Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Πληροφορικής και Τεχνολογίας Υπολογιστών

Προσαρμοστική Διαχείριση και Αναζήτηση Δεδομένων Ευρείας Κλίμακας σε Κατανεμημένα Συστήματα

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

ΑΘΑΝΑΣΙΑ Χ. ΑΣΙΚΗ

Αθήνα, Ιανουάριος 2012

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Πληροφορικής και Τεχνολογίας Υπολογιστών

Προσαρμοστική Διαχείριση και Αναζήτηση Δεδομένων Ευρείας Κλίμακας σε Κατανεμημένα Συστήματα

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

ΑΘΑΝΑΣΙΑ Χ. ΑΣΙΚΗ

Τριμελής Συμβουλευτική Επιτροπή: Παναγιώτης Τσανάκας
Συμεών Παπαβασιλείου
Νεκτάριος Κοζύρης

Εγκρίθηκε από την επταμελή εξεταστική επιτροπή την 26η Ιανουαρίου 2012

.....	.....	.....
Παναγιώτης Τσανάκας	Συμεών Παπαβασιλείου	Νεκτάριος Κοζύρης
Καθηγητής ΕΜΠ	Αναπ. Καθηγητής ΕΜΠ	Αναπ. Καθηγητής ΕΜΠ
.....	.....	.....
Τιμολέων Σελλής	Αντώνιος Δεληγιαννάκης	Ιωάννης Κωτίδης
Καθηγητής ΕΜΠ	Επικ. Καθηγητής Πολυτεχνείου Κρήτης	Επικ. Καθηγητής ΟΠΑ
	.....	
	Μανόλης Κουμπαράκης	
	Καθηγητής ΕΚΠΑ	

Αθήνα, Ιανουάριος 2012

. . . . . .

ΑΘΑΝΑΣΙΑ Χ. ΑΣΙΚΗ
Διδάκτωρ Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών ΕΜΠ

Contents

List of Figures

List of Tables

List of Abbreviations

API	Application Programming Interface
DHT	Distributed Hash Table
GIIS	Grid Index Information Service
GRIS	Grid Resource Information Services
IT	Information Technologies
MDS	Globus Monitoring and Discovery Service
OLAP	On-Line Analytical Processing
OWL	Web Ontology Language
P2P	Peer-to-Peer
PDMS	Peer Data Management System
R-GMA	Relational Grid Monitoring and Discovery Service
RDF	Resource Description Framework
RDFS	RDF Schema
SFC	Space Filling Curves
VO	Virtual Organization

Abstract

The tremendous increase of managed data by a variety of applications is a new trend observed more and more in our digital era. The ability to handle and analyse large amounts of data efficiently is a requirement posed strongly by scientific and business disciplines and the Web community. This trend leads to the adoption of distributed solutions for data management aiming at building scalable and fault-tolerant systems combining the power of multiple autonomous resources. *Peer-to-Peer networks* greatly contribute to the design of decentralized systems capable of dynamically adjusting to changes of their topology. A major class of existing Peer-to-Peer networks is the one referring to the structured overlays that implement a *Distributed Hash Table* (DHTs). The efficient lookup functionality provided by the Distributed Hash Tables has made them popular among Internet-scale applications for content publishing and sharing.

The main goal in this dissertation is the development of data management techniques for large-scale data hosted by scattered resources. In this context, novel methodologies for efficient organization, indexing, searching and updating of data are introduced. A common property of the explored data is the use of concept hierarchies, which offer the capability for organizing it at different levels of abstraction. The exploitation of concept hierarchies greatly helps in the organization and reuse of information and contributes to more effective processing of aggregate queries. The proposed techniques enable the organization of such data in a manner that preserves the semantics of the hierarchies, while they assign it among the nodes of a DHT substrate. The applied method for query processing utilizes a distributed indexing scheme allowing peers to dynamically detect the prevailing trends in incoming queries and adapt the indexing granularity. Re-indexing operations towards more general or more detailed levels can be performed individually on a per node basis for each stored hierarchy to improve the performance of the processing

and expedite the retrieval of results on variable aggregation levels. Another important aspect addressed in this approach is the support for online updates of stored items and insertions of new ones without impeding the operation of the system. Apart from the evaluation of the proposed techniques with synthetic query workloads following uniform and skewed distributions, this approach is also motivated by the use case of a Grid Information System. A fully decentralized scheme is developed that creates, queries and updates large volumes of hierarchical data on-line and can be considered as a viable solution compared to the traditional information systems comprising of centralized and hierarchical structures.

The derived methods are further extended and enhanced to enable the management of data annotated by concept hierarchies in multiple dimensions. The outcome is the implementation of all required mechanisms for a fully functional system handling data following the specific structure. A significant asset of the system differentiating it from existing approaches is the restriction for a global, rigid schema followed by all data is eliminated. The proposed scheme manages partially structured information and search strategies are described, which mainly focus on the resolution of aggregate operations over multiple dimensions. Apart from the adoption of the highly adaptive mechanisms performing re-indexing operations, a technique for the pre-computation of combinations of stored values is analysed for partial materialization of views according to the incoming queries. The introduced techniques for the management of semi-structured, multi-attribute data are also applied in the design of a system for semantic search and retrieval. The paradigm of Linked Data, which is widely used for publishing large datasets from different resources on the Web is studied. The resultant system is a distributed platform that serves the needs for integrating, indexing and querying data published in the form of Linked Data.

Περίληψη

Η αλματώδης αύξηση των δεδομένων που παράγονται και χρησιμοποιούνται από μία πληθώρα εφαρμογών είναι η νέα τάση που παρατηρείται με συνεχώς αυξανόμενο ρυθμό στη ψηφιακή εποχή μας. Η δυνατότητα της διαχείρισης και ανάλυσης δεδομένων μεγάλου όγκου είναι μια βασική απαίτηση που τίθεται τόσο από τους επιστημονικούς και επιχειρησιακούς κλάδους, όσο και από την διαδικτυακή κοινότητα. Η τάση αυτή οδηγεί στην υιοθέτηση κατανεμημένων λύσεων για τη διαχείριση δεδομένων που στοχεύουν στη δημιουργία επεκτάσιμων και ανεκτικών σε σφάλματα υποδομών. Οι υποδομές αυτές συνδυάζουν την ισχύ πολλαπλών αυτόνομων πόρων και εξασφαλίζουν την αποδοτικότερη χρήση τους. Τα *δίκτυα ομότιμων κόμβων* συνεισφέρουν στο σχεδιασμό μη κεντρικοποιημένων συστημάτων που έχουν την ικανότητα να προσαρμόζονται δυναμικά σε αλλαγές της τοπολογίας τους. Μία σημαντική κατηγορία των δικτύων ομότιμων κόμβων είναι οι δομημένες επικαλύψεις που υλοποιούν *Κατανεμημένους Πίνακες Κατακερματισμού*. Η αποδοτική αναζήτηση που επιτυγχάνεται σε αυτές τις επικαλύψεις καθιστά τους Κατανεμημένους Πίνακες Κατακερματισμού δημοφιλείς για εφαρμογές διαμοιρασμού περιεχομένου ευρείας κλίμακας.

Ο βασικός στόχος της συγκεκριμένης διατριβής είναι η ανάπτυξη τεχνικών διαχείρισης δεδομένων μεγάλου όγκου σε κατανεμημένες υποδομές. Στο πλαίσιο αυτό προτείνονται καινοτόμες τεχνικές για την αποδοτική οργάνωση, δεικτοδότηση, αναζήτηση και ενημέρωση των δεδομένων. Ένα κοινό χαρακτηριστικό των δεδομένων που μελετώνται είναι η χρήση εννοιολογικών ιεραρχιών για την δόμηση των τιμών τους σε διαφορετικά επίπεδα αφαίρεσης. Η αξιοποίηση των εννοιολογικών ιεραρχιών βοηθά σημαντικά στην οργάνωση και την επαναχρησιμοποίηση της πληροφορίας και μπορεί να συνεισφέρει στην αποτελεσματικότερη επεξεργασία ερωτημάτων σύνοψης. Οι προτεινόμενες τεχνικές επιτυγχάνουν την οργάνωση της πληροφορίας κατά

τέτοιο τρόπο ώστε να διατηρείται η σημασιολογική πληροφορία που εμπεριέχεται στις ιεραρχίες, ενώ αυτά κατανέμονται στους διαθέσιμους κόμβους ενός Κατανεμημένου Πίνακα Κατακερματισμού. Η επεξεργασία των ερωτημάτων γίνεται με τη χρήση μίας πλήρους κατανεμημένης δομής δεικτοδότησης που επιτρέπει στους κόμβους να αντιλαμβάνονται δυναμικά τις επικρατούσες τάσεις στα ερωτήματα των χρηστών και να προσαρμόζουν αντίστοιχα τη δεικτοδότηση. Επίσης, οι προτεινόμενοι μηχανισμοί επαναδεικτοδότησης είτε προς πιο γενικότερες τιμές ή τιμές μεγαλύτερης λεπτομέρειας μπορούν να εκτελεστούν μεμονωμένα απο κάθε κόμβο για κάθε αποθηκευμένη ιεραρχία, ώστε να βελτιωθεί η απόδοση της επεξεργασίας των ερωτημάτων και να επισπευσθεί η ανάκτηση των αποτελεσμάτων. Ένα άλλο σημαντικό θέμα που εξετάζεται είναι η online ενημέρωση των αποθηκευμένων δεδομένων και η προσθήκη νέων χωρίς να αναστέλεται η λειτουργία του συστήματος. Οι τεχνικές που περιγράφηκαν αξιολογήθηκαν με συνθετικά φορτία ερωτημάτων που ακολουθούν ομοιόμορφες και πολωμένες κατανομές. Επιπρόσθετα, η προσέγγιση αυτή μελετήθηκε για το σενάριο χρήσης ενός Πληροφοριακού Συστήματος μίας Υποδομής Πλέγματος. Για το λόγο αυτό αναπτύχθηκε ένα πλήρως κατανεμημένο σύστημα για τη δημιουργία, επερώτηση και ενημέρωση μεγάλου όγκου ιεραρχικής πληροφορίας προερχόμενης από αυτήν την εφαρμογή και το οποίο μπορεί να θεωρηθεί σαν μία βιώσιμη λύση σε σύγκριση με τα υπάρχοντα συστήματα που αποτελούνται από κεντρικοποιημένες και ιεραρχικές δομές.

Οι τεχνικές, που προέκυψαν, επεκτάθηκαν και εμπλουτίστηκαν, ώστε να γίνει εφικτή η διαχείριση δεδομένων που περιγράφονται από εννοιολογικές ιεραρχίες σε πολλαπλές διαστάσεις. Το αποτέλεσμα ήταν η ανάπτυξη των απαραίτητων μηχανισμών και ενός πλήρως λειτουργικού συστήματος που προορίζεται για δεδομένα με την περιγραφόμενη δομή. Ένα σημαντικό στοιχείο που το διαφοροποιεί τη προσέγγιση αυτή από υπάρχουσες λύσεις είναι η άρση του περιορισμού για την υιοθέτηση ενός αυστηρά ορισμένου σχήματος, που πρέπει να ακολουθείται από τα δεδομένα που εισάγονται στο σύστημα. Το προτεινόμενο σύστημα διαχειρίζεται μερικώς δομημένη πληροφορία και οι στρατηγικές αναζήτησης που περιγράφονται επικεντρώνονται κυρίως στην επίλυση ερωτημάτων σύνοψης σε πολλαπλές διαστάσεις. Εκτός από την υιοθέτηση των ιδιαίτερα προσαρμοστικών μηχανισμών αναζήτησης για την εκτέλεση διαδικασιών επαναδεικτοδότησης, μία τεχνική για τον υπολογισμό συνδυασμών από αποθηκευμένες τιμές αναλύεται για τη μερική δημιουργία όψεων σύμφωνα με τις εισερχόμενες ερωτήσεις. Οι τεχνικές αυτές για τη διαχείριση μερικώς δομημένων και πολυδιάστατων δεδομένων εφαρμόστηκαν για το σχεδιασμό ενός συστήματος για σημασιολογική αναζήτηση και ανάκτηση δεδομένων. Το παράδειγμα των Διασυνδεδεμένων Δεδομένων χρησιμοποιείται ευρέως για τη δημοσιοποίηση μεγάλων συλλογών δεδομένων στο Διαδίκτυο και είναι αυτό που μελετάται. Το συγκεκριμένο σύστημα μπορεί να αποτελέσει μία κατανεμημένη πλατφόρμα για την εξυπηρέτηση των αναγκών της ενοποίησης, δεικτοδότησης και επερώτησης δεδομένων που δημοσιεύονται με τη μορφή *Διασυνδεδεμένων Δεδομένων*.

Πρόλογος

> Everything that can be counted does not necessarily count;
> everything that counts cannot necessarily be counted.
>
> *Albert Einstein*

Φτάνοντας στο τέλος του διδακτορικού, προβάλλει ακόμα πιο εντυπωσιακά το γεγονός ότι υπήρξαν πολλαπλές και ραγδαίες αλλαγές στις Τεχνολογίες Πληροφορικής κατά τη διάρκεια αυτών των χρόνων. Αναπτυσσόμενες τεχνολογίες ωρίμασαν και έδωσαν τη θέση τους σε καινούργιες για να συνεχίσουν να "ρέουν" οι εξελίξεις και καινούργια τεχνολογικά επιτεύγματα να γίνονται μέρος της καθημερινής μας ζωής. Πολλές φορές, δυσκολεύομαι να θυμηθώ πως ήταν χωρίς να χρησιμοποιείς το διαδίκτυο για να ψάξεις για "οποιαδήποτε" πληροφορία ή χωρίς να βλέπεις τα e-mail σου στο κινητό σου οποιαδήποτε στιγμή, όπου και αν βρίσκεσαι. Όμως, ανατρέχοντας πίσω στα πρώτα χρόνια του διδακτορικού, οι βασικές προσπάθειες στρέφονταν στη δημιουργία κατανεμημένων υπολογιστικών υποδομών με στόχο να παρέχουν όσο το δυνατόν ευκολότερη πρόσβαση σε υπολογιστικούς πόρους και εφαρμογές. Πλέον, ένα κινητό τηλέφωνο αρκεί για να έχεις απομακρυσμένη πρόσβαση σε υπερυπολογιστές, σε αστείρευτη πληροφορία, σε όλα σου τα δεδομένα, σε εφαρμογές κοινωνικής δικτύωσης και σε πληθώρα διαδικτυακών εφαρμογών. Είναι συναρπαστικό να παρατηρείς πως εξελίσσονται οι διάφορες τεχνολογίες και πως αλλάζουν μορφή, εμπλουτίζονται ή αφήνονται στο παρελθόν, για να προκύψουν νέες, ακόμα πιο πρωτοποριακές.

Η εκπόνηση της διδακτορικής διατριβής είναι μία πολύπλοκη διαδικασία, που απαιτεί την απόκτηση γνώσεων από πολλαπλά πεδία, την εμβάθυνση σε ερευνητικά θέματα και τη συστηματική ενασχόληση. Για την ολοκλήρωση της διαδρομής αυτής ήταν καθοριστική η παρουσία

όλων αυτών που συνέβαλαν ουσιαστικά, ο καθένας με το δικό του τρόπο, και για αυτό θα ήθελα να τους ευχαριστήσω, έστω και αν είναι δύσκολο μέσα από αυτές τις λίγες γραμμές. Αρχικά, θα ήθελα να εκφράσω τις ευχαριστίες μου στον επιβλέποντα μου Καθηγητή κ. Παναγιώτη Τσανάκα που μου έδωσε την ευκαιρία να γίνω μέλος του εργαστηρίου Υπολογιστικών Συστημάτων του Εθνικού Μετσόβιου Πολυτεχνείου και να ασχοληθώ ερευνητικά στον τομέα μου. Κατά τη διάρκεια του διδακτορικού, έδειξε εμπιστοσύνη στις δυνατότητες μου και μου παρείχε την ελευθερία να επιλέξω τις ερευνητικές κατευθύνσεις που κέρδισαν το ενδιαφέρον μου. Επίσης, θα ήθελα να ευχαριστήσω τον Αναπ. Καθηγητή κ. Νεκτάριο Κοζύρη για τις χρήσιμες συμβουλές του και την αμέριστη υποστήριξη που μου έδειξε από την πρώτη στιγμή που ήρθα στο εργαστήριο. Η συμβολή του ήταν καθοριστική στη διαμόρφωση της πορείας που ακολούθησα και πάντα ήταν πρόθυμος να συνδράμει στην επίλυση θεμάτων που με απασχόλησαν. Θα ήθελα ακόμα να εκφράσω τις ευχαριστίες μου σε όλα τα μέλη της επιτροπής παρακολούθησης και κρίσης του διδακτορικού μου, που ανέλαβαν πρόθυμα το έργο αυτό.

Επίσης, θα ήθελα να αναφερθώ και στα υπόλοιπα μέλη της ομάδας του εργαστηρίου μου. Θα ήθελα να ευχαριστήσω το διδάκτορα Δημήτρη Τσουμάκο για τη βοήθεια που μου πρόσφερε και τη συνεργασία που είχαμε κατά τη διάρκεια του διδακτορικού, που συνέβαλε στο να αποκτήσω μία πιο ολοκληρωμένη αντίληψη για την αναζήτηση των ερευνητικών αποτελεσμάτων. Η παρουσία μου στο εργαστήριο μου έδωσε την ευκαιρία να συνεργαστώ με άτομα με κοινά επιστημονικά ενδιαφέροντα, αλλά και να συμμετέχω σε εποικοδομητικές συζητήσεις και ανταλλαγή απόψεων. Για όλα αυτά αλλά και την πολύ φιλική ατμόσφαιρα ευχαριστώ όλους τους συναδέλφους που συνάντησα κατά τη διάρκεια αυτών των χρόνων: τους διδάκτορες Κατερίνα Δόκα, Άρη Σωτηρόπουλο, Γιώργο Γκούμα, Αντώνη Χαζάπη, Αντώνη Ζήσιμο, Γιάννη Κωνσταντίνου, Νίκο Αναστόπουλο, Κορνήλιο Κούρτη, Βαγγέλη Κούκη, Κωστή Νίκα αλλά και τους Γεωργία Κουβέλη, Χριστίνα Μπούμπουκα, Γιώργο Βεριγάκη, Γιώργο Τσουκαλά, Βαγγέλη Αγγέλου, Βασίλη Καρακάση, Τάσο Νάνο, Στέφανο Γεράγγελο, Τάσο Κατσήγιαννη. Εύχομαι σε όλους η πορεία τους να είναι αντάξια των προσδοκιών τους.

Οι τελευταίες αυτές γραμμές είναι αφιερωμένες στα πιο αγαπημένα μου πρόσωπα, που έκαναν τα πάντα να φαίνονται εφικτά, με ώθησαν να προχωρήσω ακόμα και τις πιο δύσκολες στιγμές και μου ενέπνευσαν τη θέληση μου για αναζήτηση της γνώσης. Το μεγαλύτερο ευχαριστώ μαζί με την αγάπη μου είναι για τη μητέρα μου Γιώτα και τον πατέρα μου Χάρη που με στηρίζουν όλα αυτά τα χρόνια για να μπορώ απερίσπαστη να ακολουθώ τα όνειρα μου, τον αδερφό μου Θωμά που με τον ενθουσιασμό του με γεμίζει αισιοδοξία και το σύντροφο μου Σωτήρη, που με την υποστήριξη του και τη παρουσία του σε κάθε στιγμή του ταξιδιού αυτού το έκανε όμορφο και δημιουργικό. Σε εσάς αφιερώνω και την παρούσα διατριβή.

Αθανασία Ασίκη,

Ιανουάριος 2012

Adaptive Management and Search of Large Scale Data in Distributed Systems

CHAPTER 1

Introduction

> Intelligence is the ability to adapt to change.
>
> *Stephen Hawking*

Data management is an important field of computer science, especially with the rapid explosion of data forming a new reality in the digital world. The tremendous increase in the volume of content is a global phenomenon, affecting a variety of applications and making it one of the biggest challenges in the area of Information Technologies (IT). Data is everywhere and large amounts are produced by various kinds of applications in our everyday life and usually transmitted over long distances. A variety of sources, from more widespread ones such as computers, cameras, microphones, mobile phones to more specialized ones like sensors, scientific instruments, etc., flood our world with more information at each moment. This "data deluge" requires new solutions in data management and data mining, as well as new approaches for the underlying computational infrastructure.

Various studies have been conducted about the growth of data in various fields, including the scientific and business domain. Even in our everyday life, we produce and consume a large amount of digital content in our interaction with digital devices. Nowadays, people go about their day by communicating, browsing, buying, sharing and searching: All these are activities that produce enormous trails of data. Apart from the usage of our personal computer, other activities such as watching television or talking over the phone have engaged the digital era and contribute significantly to the size of digital information being stored, transferred and replicated. Everyday

habits have led to a new reality, where for instance only the U.S. households have already broken the *zettabyte* barrier [GRC+07].

The same phenomenon is observed in many scientific disciplines as well and the reported use cases where special effort is needed for handling data are numerous: earth and space sciences, aerospace, meteorology, climate prediction, medical imaging, genetic sequence mapping. For example, a well-known reference of massive data generation is the Worldwide LHC Computing Grid (WLCG) [WLC] hosting events registered by the detectors of the LHC experiment in CERN. The digitized summaries of data recorded by a vast number of sensors reaching an astonishing rate of some terabytes per second. This size cannot be managed in a centralized solution and thus multiple stages of filtering take place to limit it to the order of megabytes so as to be stored and processed in a large-scale, geographically distributed infrastructure. Climate research is another demanding area, as model simulation experiments, remote sensors and observation platforms produce hundreds of petabytes of data which must be tightly integrated into a "smart data infrastructure system" [AHL+11]. In general, petascale computing has already moved scientific applications in a data-centric world. Scientists are faced with enormous data that stem from the flood of new scientific instruments and simulations [BGS06]. In the meantime, the ability to economically store petabytes of data online and the advent of Internet and high-performance computing making it accessible to anyone anywhere leads to the replication, creation, and recreation of more data [BGS06]. The vision of "exascale computing" will make things even more challenging, as it will enable applications to scale up and integrate technologies into complex coupled systems for real-world multidisciplinary modelling and simulation. This new form of computing will rapidly increase the rate of generated data requiring new techniques for dealing with the volume, transfer, analysis and virtualization of massive and possibly distributed datasets [GL09].

Innovative techniques for data handling can also play a crucial role for the operation of business companies and organizations. The usual approach in industry and business is to maintain data in local storage systems, which mainly use database technologies. Nevertheless, it is often the case that relational databases which have been very successful solutions for enterprises, cannot cope with the exponential increase in the information managed by the businesses [MCB+11]. The size of data can become a prohibitive factor for its storage; for instance, health care providers discard over 90 percent of the data that they generate (e.g., almost all real-time video feeds created during surgery) [MCB+11]. "*Big data*", i.e., large datasets whose size is beyond the ability of a typical software tool to capture, store, manage and analyze, is the new challenge for enterprises and public organizations. The exploration of big data can lead to the export of prevailing trends and this knowledge can be utilized for the increase of productivity and competitiveness, e.g., such analysis takes place for forecasting the customers preferences on products. Mining for knowledge that exists in the content of the repositories of an organization by using platforms

with analytic capabilities can boost the evolution of its activities, while it points out the need for the development of new tools and platforms to resolve the encountered problems.

Nevertheless, the medium that revolutionized the access to information and changed the landscape of exploiting the power of data is the Internet and the Web. Nowadays, vast amounts of digital content is transferred through the Internet in every moment and everyone can find "infinite" information on the Web. Powerful search engines index constantly the available content so as to return results faster and improve their quality. In 2008, Google [Goo] announced that it managed to index 1 trillion unique URLs on the web at once, while the first Google index back in 1998 already had 26 million pages and by 2000 this number reached one billion [1]. The distributed infrastructure used by Google allows its application to efficiently traverse a link graph with many trillions of connections and quickly sort petabytes of data, only for preparing to answer a Google search [1]. But Google is not the only one that comes along the challenge to keep up with huge volumes of information. Social networks are well-established candidates of resources that add new content. In Facebook [fac], more than 800 million active users upload more than 250 millions of photos per day on average and install applications more than 20 million times every day [2] . In Twitter [Twi], the average number of "tweets" that people send in a single day exceeds the 140 million [3]. Finally, each of Wikipedia's about 20 million articles is edited over 30 times a month, with over 8K new articles added to the collection per day [4]. The impressive fact is that even if the size of Indexed Web amounts some billions pages, there is also a large portion of content on the Web not accessible through search engines known as the *"Deep Web"*. However, the correlation of different pieces of information raises the need for a new perspective on search systems capable of supporting more advanced types of queries perform complex combination and integration of data [CBB11].

The common reference point in the described applications from different areas is the vast amounts of data to be managed. However it must be noticed that there are also cases where storing all the generated data for further processing is not required. A lot of data in modern applications are temporary and millions of bytes are generated from applications and used for few seconds. Despite of the importance of this data during its short-lives (e.g. for the communication among digital devices), this data is finally overwritten. All these observations indicate that it is not always useful to just store data, but it is also significant to generate value by extracting the right information. Towards this direction, an increasing effort is put into the development of new methods and tools. As pointed out in [GR11], the rate of creating data about data or metadata in an automated manner is doubled compared to the growth of our digital universe as a whole.

[1] http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html
[2] http://www.facebook.com/press/info.php?statistics
[3] http://blog.twitter.com/2011/03/numbers.html
[4] http://stats.wikimedia.org/EN/TablesWikipediaZZ.htm

A common example is the constantly increasing number of photos in the social networks. To enable their search, special tools have been developed for tagging based on facial recognition; otherwise a photo will remain a useless piece of data, which cannot be shared and searched.

A widely adopted solution for data management is the creation of a central repository due to the various offered advantages. Storing the whole dataset in a single site enables a unified view making easier to develop faster methods for processing. The whole information is accessible at once and transferring data around is not needed during the resolution of queries. Nonetheless, big data prevent centralized solutions from being effective and many problematic situations appear, such as the increase of time for data load and query processing and the unavailability of the provided services as the number of requests increase.

Modern applications adopt more distributed approaches capable of combining the power of multiple autonomous resources to address such kinds of issues. Distributed computing is a constantly expanding field, making hard to define it precisely. Distributed processing may take place even on a personal computer, where different parts of an application can be assigned to different resources. This dissertation refers to distributed computing as it is defined in [TS06] and [OV11]: A distributed computing system comprises of a number of autonomous elements, either homogeneous or heterogeneous, interconnected by a computer network and able to cooperate to perform their assigned tasks. The distribution may occur on various levels, but in the current work, the distribution of process logic and the distribution of data are mainly considered. By the distribution of processing logic, it is assumed that different tasks are assigned to different processing nodes, which cooperate to complete them. Nevertheless, the main focus of the presented work is towards the distribution of data among a number of processing sites.

An important property of distributed systems is the elimination of single points of failure and possible bottlenecks in the system. During the last years, the problem of unavailable nodes offering crucial due to load or hardware malfunctions is often investigated [SC11]. Moreover, distributing the processing and retrieval of data among multiple nodes results in more efficient data access and higher throughput, while adverse cases of either extremely heavy-loaded or completely idle nodes can be avoided.

Another aspect addressed by distributed systems is the efficient placement of data among the participating nodes according to the needs of an application, especially if we take into account that a large number of applications are inherently distributed, i.e., web-based applications or scientific applications. Distributed data management allows more flexibility in the placement of data according to the nature of each application; a requirement that was listed as a major challenge of "exascale" computing as well [GL09]. Replication also makes decentralized systems powerful, as far as data availability and robustness is concerned. Since a distributed system consists of dispersed nodes, the approach of replicating a piece of data to different nodes can be adopted: The existence of multiple replicas of a data item on independent nodes increases the probability of

at least one replica being available despite of node failures, unreliable network connectivity and limited bandwidth.

Finally, it is worth mentioning that decentralized approaches are capable of handling gracefully the increase in the volumes of data and numerous requests of the services provided by the hosted application. Also, new resources can be added easily to the system without needing to reconstruct it from the scratch. The scalability is one of the strongest features of distributed approaches that have made them popular. In the context of this thesis, scalability issues concerning the volume of data, the size of the network and the number of clients are investigated. What is also important is to provide a coherent system capable of managing an increased number of data items and provide its services with a unified manner regardless of the number of nodes. Users and applications do not need to be aware of the exact physical location of data and they behave, at least conceptually, as if all data were stored locally.

Various categories of large-scale distributed architectures extending their functionality beyond the server client model have been proposed in the bibliography. Peer-to-Peer (hence P2P) and Grid computing can be considered some of the most widely adopted models so far. Some fundamental concepts appear in the majority of the distributed models, while others differentiate one from another. In the rest of this dissertation, the P2P and Grid Computing models are discussed, as the basic distributed models studied and exploited in this thesis.

Peer-to-Peer computing: P2P technologies can be used to build scalable network overlays. The nodes participate in the overlay in an autonomous and unsupervised manner and can act both as servers and clients at the same time. The resultant overlay is able to dynamically adjust itself to node arrivals and departures following self-organized procedures. P2P systems can provide good substrates for large-scale data sharing and content distribution. Moreover, building data-intensive applications on top of P2P overlays is advantageous due to their assets such as efficient routing, redundant storage, fault-tolerance, scalability and load balancing in large-scale environments. Given the distributed nature of these architectures, a major question is how to organize the content among the nodes to ensure its availability during dynamic changes of the overlay and heavy load and at the same time to enable its efficient search. The existing approaches are mainly towards two directions: either data items are placed at random nodes or in a deterministic manner determined by the used protocol. A major class of P2P networks is the one of structured overlays implementing a ***Distributed Hash Table (DHT)***. In this case, a scheme is constructed that tightly controls the data placement and assigns identifiers to data items, which are mapped to identifiers of nodes based on some distance metric [BKK+03]. The use of a distributed index makes DHTs popular due to the fact that enables efficient lookups of the stored

values. P2Ps have been used for a wide variety of Internet-scale applications, e.g. content distribution (Bittorent [Bit], eMule [eMu]) and communication networks (Skype [Sky]).

Grid computing: Grid computing is mostly oriented towards high performance computations. A Grid platform can be viewed as a federation of heterogeneous computational and storage resources, shared among different groups of users. In existing infrastructures, clusters interconnected with high-throughput networks and belonging to different administrative domains are the main structural components of the system. The platforms offer primarily computational power and storage space to users according to statutory policies among the consumers and producers of the services. The coordination required for the sharing of resources is achieved through the existence of centralized structures, such as brokers, that maintain information (e.g. number of running jobs, number of waiting jobs in queues, etc.) about the whole infrastructure so as to be able to schedule the submitted tasks.

1.1 Problem Description

Distributed data management contributes to the scalability, robustness, fault-tolerance of applications handling large volumes of data. Nevertheless, various issues arise that need to be encountered in the design of a distributed platform, while the type of the application and the structure of the data to be hosted play a significant role in the resultant architecture.

In this thesis, the problem under investigation is the efficient management of large volumes of hierarchical and multidimensional data which can be partially structured. The techniques to be developed refer to computational environments comprising of interconnected commodity nodes, which may belong either to geographically scattered resources or more centralized infrastructures such as data centers. The addressed issues are the efficient integration, organization, indexing, query processing, retrieval and updating of data described by concept hierarchies. Also, these aspects are explored for multidimensional datasets, which can partially structured.

Concept hierarchies are often met in data mining and OLAP applications analyzing large datasets to export patterns and trends [HK00]. In most cases, the results of such processing are intended for decision making systems, business intelligence tools and platform for data mining analytics. As a result, the processing of complex forms of queries including multiple levels of aggregation is required. The use of hierarchies enables the ordering of attributes that may exist in a database and thus hierarchies greatly help in the organization and reuse of information. A *concept hierarchy* or *taxonomy* defines a sequence of mappings from more general to lower-level concepts [HK00]. Concept hierarchies are important because they allow the structuring of information into categories, thus enabling easier searches and creation of synopsis. The selection and ordering of concepts are usually provided by application or domain experts. For example,

a concept hierarchy for the attribute *location* may include the levels: `Country - City - Street - ZipCode`, where the level of `Country` corresponds to the most general level. As the volume of data increases and the extraction of useful information becomes more and more difficult, concept hierarchies can significantly contribute to the organization of data and deliver fast insights due to the summarization property incorporated in their structure. Apart from the exploitation of hierarchical structures in databases and data warehouses, other use cases can be considered as well, especially when the sharing of information is performed among machines, applications or services. In this case the lack of structure cannot be handled and a hierarchical classification is a machine-processable form encoding a lot of information and ensuring interoperability. According to this perception, metadata attributes can be structured hierarchically. A more generic paradigm is taxonomies, which are widely used for data integration in the Semantic Web.

The multidimensional data model is also an integral component of analytic processing. In this model, multiple attributes are mapped to *dimensions* describing one (or more) numerical value(s) called *fact(s)*. The problem faced with this model is that the increase in the number of dimensions results in an exponential growth in the number of views that can be computed. The complexity of a multidimensional data model increases more, if it is assumed that each dimension is further annotated by a different concept hierarchy. Apart from the wide use of the multidimensional data model in OLAP applications for the creation of *data cubes*, it can be also applied in many other cases. For example, the metadata attributes describing photos (e.g. location, name, date, topic, photo type, etc) enabling their search can be considered to form a multidimensional space, where each point represents a specific photo. Typical queries in such an application may include searching photos with a specific tag in a given location or for a specific topic or for the combination of both attributes. This example also points out the fact that information about all dimensions may not be always available, i.e., not all photos are described by their location or their type. Semi-structured data [Bun97] with self-contained or loose schema are studied in various applications, especially in Web related ones. The handling of such data is described as one of the forthcoming challenges to be faced by the database community in [AAB+08]. It is underlined that the interest of managing heterogeneous data (namely structured, semi-structured and unstructured) spread over many repositories has gradually taken over the focus on traditional databases consisting of well-defined schemata for structured business data.

A representative example combining the properties of such data can be met in the Web of Data [BHBL09], which presents a revolutionary opportunity for deriving insight and value from data by enabling seamless connections between different data sets. By adopting practices such as the ones introduced in the *Linked Data* initiative [Data], a major opportunity is given to link pieces of information distributed across the Web and therefore connect different sources into a single global data space resulting in the change of the way that we exploit data in the Web. The

accomplishment of Internet-scale interconnected resources implies that their data are characterized by high degrees of heterogeneity and span numerous topical domains relative to people, companies, films, music, locations, etc. This paradigm highlights the need for supporting and integrating structured and semi-structured content classified into different categories and published by different sources. As a result, new approaches arise to address the challenges met in extracting structure from such data and developing methods for effectively querying and deriving insight from outcome of its integration.

In the context of this dissertation, the targeted applications manage data of the described types and the following major categories can be distinguished from the perspective referring to their requirements of the underlying architecture that hosts them.

- Applications that are traditionally implemented in centralized systems with a need to extend and adapt them to decentralized architectures.
- Applications that are inherently distributed, for example a variety of web-based and scientific ones.

In the design of a distributed system for data management, various issues need to be tackled, which are usually emerging from the nature of the application itself. Two significant factors investigated in this thesis are the data structure and the types of queries to be resolved by the resultant system. The data structure influences its organization among the participating nodes and how it is stored in each node. The type of the queries indicates the processing strategies to be applied and any additional, needed information to be maintained. For instance, top-k queries require additional ranking information of the possible answers to calculate the final result.

Some essential questions need to be answered during the process of designing a distributed system regarding the already discussed issues and examined in the approaches presented in this thesis: *Which scheme is more efficient to organize the available nodes and their communication? How is data distributed and assigned to the participating nodes? How is access provided to data so as to be easily searched and reused? How can the integration of data from different resources occur? How can data items be efficiently indexed taking into account the properties of their structure, their among relationships and the targeted queries? Which are the types of queries resolved by the developed system? Which strategy is the most appropriate for effective query processing? How often is data updated and how much processing is acceptable during updates? Are there any synchronization issues to be considered during updates?*

The communication framework of the systems proposed in this thesis is built with the utilization DHT-based overlays. Various research works highlight the fact that P2P technologies can provide competitive solutions for designing distributed systems hosting data-intensive applications ([LCP+05], [ATS04],etc.) . The implemented systems presented in this thesis are not tightly

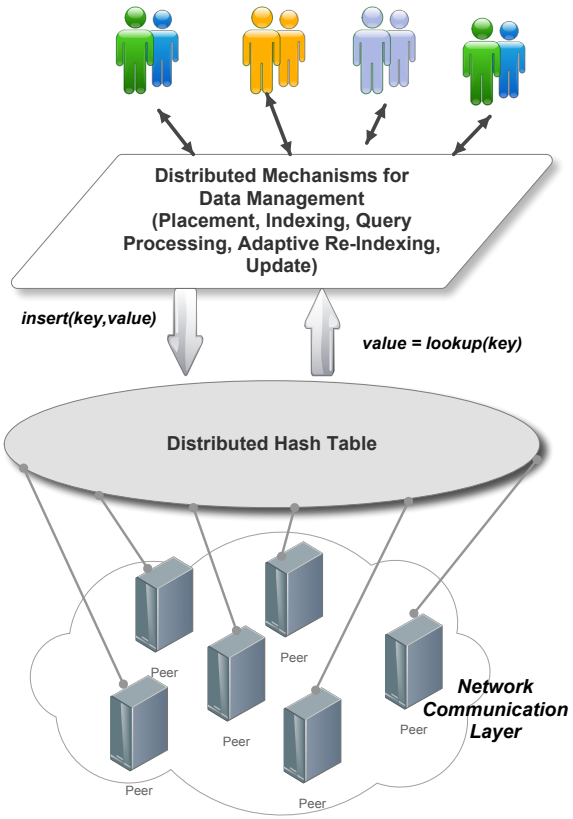


Figure 1.1: *An abstract overview of the assumed layers in the presented architectures presented of the current thesis.*

bound to a specific DHT protocol, as any implementation providing the basic functionalities for inserting and looking up objects can be used. The obtained substrate is used as the basic layer for structuring the participating nodes and message routing. DHTs have been chosen due to their ability to organize the participating peers in a consistent manner defined by the implemented protocol and offer logarithmic lookup complexity on the number of peers. Thus, DHT overlays do not suffer from high bandwidth consumption and intrinsic limitations imposed by the lack of structure. Figure 1.1 exhibits an abstract overview of the assumed layers in the architectures described in my dissertation. The nodes are organized in a DHT-based overlay and the supported functionalities for the management of the stored content are built on top of this overlay.

The construction of DHT-based overlay implies that data items are assigned to nodes according to a specific logic determined by the corresponding protocol. Each data item is mapped to an identifier called *key*, which is then used to determine the responsible for this item. If the key of a data item is known during its lookup, then this item can be easily located in the overlay. Nevertheless, this property of DHTs introduces also some limitations. In more complicated types of queries, the searched keys may not be known in advance and has to be discovered on

the fly. As queries become more complex, additional mechanisms extending the DHT primitives are required.

DHTs are based on random assignment of data items to avoid overloaded nodes. Nonetheless, the random placement of data may negatively affect the performance and the cost of resolving some types of queries. For instance, range queries require an ordering of the data items so as to avoid issuing consecutive queries for each value within the range separately, as this will result in high bandwidth consumption. The same remark is valid for the case of aggregation queries as well, where the aggregation function is calculated over a set of facts "grouped by" a specific attribute. In this case, the structure of the data can contribute to more effective placement and indexing of the data, that is by exploiting a common characteristic of its structure to group them and assign them to the same node. This would be very helpful for the case of aggregation and range queries, which are the main categories of queries that focus my attention in this thesis apart from point queries.

In general, the use of aggregation functions in queries is a popular and powerful feature, especially for facilitating decision making where data need to be accessed, combined and summarized [WJOT09]. The usefulness of aggregate queries can be obscured by the fact that long processing is required for their resolution. To answer such a query, several items of the database need to be accessed for the calculation of the aggregation function. As the volume of stored data increases, the processing of this queries becomes slower and more expensive and thus centralized solutions are left aside and replaced by parallel and distributed ones.

The big size of data makes the indexing of all stored values and all possible combinations among them expensive. Especially in multidimensional data, the number of possible combinations increases exponentially with the number of dimensions. The hierarchical structure of the assumed data inherently supports the grouping of values and this property is exploited for the placement of items grouped by the same value (in the chosen level) in the same node. The selection of a single level to index can be proven very effective, especially if the indexed level can be adjusted according to the demands of the users. This solution is adopted in this work: the indexing is highly adaptive to the trends observed in the queries, while it takes place in an online mode without requiring time-consuming processing or re-insertion of the entire data.

As a consequence of selecting only a single level of the hierarchy, the lookup operation of the DHT locates only the values corresponding to this level without having to contact all the nodes of the overlay, i.e., "flood" the query among all nodes. For this reason, additional indexing capabilities are required to avoid flooding, but at the same time to encumber the system with reasonable maintenance and update cost. Bearing in mind that in the targeted applications, such as web-based and OLAP applications, the recognition of repetitive patterns in the queries is a common fact, the adaptiveness of the indexing can serve the effectiveness of the system. For example, in a web-based application when an item becomes popular, then the majority of the queries target this

item. In this case, the indexing of the popular items is adequate to provide faster responses and reduce the communication cost. Hence, main emphasis is given on the development of mechanisms for adaptive re-indexing in this thesis apart from creating indices. It is also intended that these mechanisms do not to deprave the DHT of its features, causing unexpected side-effects, i.e., by overloading only a few nodes or return inaccurate results.

1.2 Contribution of the Thesis

In this dissertation, novel methodologies are introduced to enable the implementation of scalable mechanisms for efficient integration, organization, indexing, search and update in large-scale distributed repositories. These mechanisms are integrated in the design of fully-functional platforms facilitating the management of data characterized by hierarchies, multiple dimensions and self-describing semantic information. To motivate the subsistence of the proposed systems, real-world applications have been considered and the systems have been appropriately customized and evaluated.

The general principles that unify the systems proposed in my dissertation are:

- The placement and indexing mechanisms focus on hierarchical and multidimensional data, while they can support both tightly structured and semi-structured data in an efficient manner.
- The forms of queries under investigation, such as aggregate and range queries require more complicated processing and thus mechanisms performing more advanced operations than simple lookups are established.
- Powerful, distributed indexing schemes leveraged by the data structure and the types of targeted queries are developed.
- Flexible mechanisms achieving the adaptation of the indexing scheme of a distributed infrastructure are introduced.
- The processing of data for constructing and updating the maintained index scheme does not require any offline procedures to take place.
- The scalability of the proposed systems as the volume of data and the number of resources increases is pursued.

The main contributions of my thesis can be enumerated as follows:

- Efficient methods are proposed for organizing, indexing, searching and updating data following concept hierarchies. The placement of data is performed in a hierarchy preserving

manner that maintains the semantic information included in the concept hierarchies. The indexing approach is self-adaptive to the incoming queries and flexible mechanisms are proposed to enable a query-driven re-indexing scheme. The resultant scheme is capable of detecting trends in a query workload and presents a highly adaptive behaviour by adjusting the imposed indexing in an automatic manner, so as to increase performance and decrease latency for answering queries on variable aggregation levels. Nodes participating in the system decide individually on the level of indexing according to the incoming queries and perform the required re-indexing operations only for their piece of data that these operations are decided to be beneficial. Moreover, a method for online updates of the hierarchical data is introduced. The difficulty to overcome during updates is that the distribution of items among many nodes and the lack of centralized structures require additional effort to decide the node that an item should end up and which maintained information should be updated. For this reason, a distributed catalogue is implemented for maintaining the required information. All these methods have been combined and implemented in a fully-functional system built on top of a structured P2P overlay, so as their efficiency and achieved performance to be evaluated.

- The usefulness of this scheme is motivated by customizing it to serve as a high-performance, distributed information system. The described scheme creates, queries and updates records containing information about the state of a grid infrastructure (e.g., available processing units, storage space, etc.) and can serve as a viable solution compared to the traditional centralized information systems for such kind of infrastructures. Experimental results on a real accounting application using the Grid Information Service show that the proposed distributed architecture outperforms traditional approaches by eliminating offline processing and other performance bottlenecks.

- Extending the work on concept hierarchies, a fully operational system is presented for storing and processing data items described by a set of attributes, i.e., multiple dimensions, which are further annotated by concept hierarchies. A significant feature is that the demonstrated mechanisms do not depend on a predefined, rigid schema and support partially structured data without requiring mediator nodes for translating the schemas. For the efficient resolution of queries, the described system utilizes a “conceptual” chain of DHTs rings storing information in a hierarchy preserving manner. The adaptiveness of the system is still maintained as one of its novel features. Besides the existence of mechanisms for adaptive re-indexing, another distinguishing property of the system is the proposed strategy for pre-computing combinations of queried values, so as to materialize partial views for future usage.

- The techniques introduced for the management of multidimensional, hierarchical data are applied in the implementation of a system for storing and searching semantic data.The use case of semi-structured data used for making available an increasing number of datasets on the Web, namely *Linked Data* is studied. The resultant system is a distributed platform that serves the needs for storing, indexing and querying data published in the form of Linked Data.

1.3 Outline of the Thesis

The rest of the dissertation is organized as follows:

In *Chapter 2*, an overview of the existing indexing and search methods in P2P systems is demonstrated. An indexing technique based on Space Filling Curves for multidimensional data is described and utilized to implement a distributed and scalable Grid Metadata service.

Chapter 3 introduces adaptive methods for distributing and searching data described by concept hierarchies. The types of hierarchical data are studied and methodologies are presented for storing, indexing, querying and updating such data in a system built on-top of DHT-based substrate. Much of my attention is focused on the algorithms that decide about the re-indexing operations and the actions that should be performed during the re-indexing operations to adjust the level of granularity appropriately. The implementation of the described methodologies and their interaction in a system aiming to accommodate hierarchical data are presented. Moreover, the use case of a distributed Information System is investigated and it is analyzed how this application can be imported in a system with the aforementioned properties. This chapter also includes an extensive experimental evaluation of the performance of such a system using synthetic workloads or real data coming from an existing Grid Information System.

The problem of partially structured, multidimensional data is explored in *Chapter 4*. The assumption that the inserted data can be organized in a hierarchical manner is still considered as a fact. A first encountered issue is the distribution and placement of such data on the available nodes, so as multi-term queries containing values from one or more dimensions to be answered. The methods for resolving point and aggregate queries are analyzed. The description of the adjustments in the re-indexing operations is given so as to function in a system that multiple dimension are indexed separately and interlinked. Also, in this chapter a query-driven partial materialization of views based on the pre-computation of possible combinations of the values contained in a query is explored. The chapter continues with the Linked Data paradigm being studied. The description of the architecture of a system follows for the management of semi-structured web data published by heterogeneous resources and described by ontologies in the form of Linked Data. The performance of both systems is evaluated in the sections of this chapters.

Chapter 5 concludes the work accomplished in this dissertation and summarizes some further extensions being studied.

CHAPTER 2

Data Management exploiting Peer-to-Peer Technologies

Peer-to-Peer technologies have been widely adopted as scalable and fault-tolerant solutions to build distributed systems. In practice, the most popular application of P2P systems is their usage for global-wide file-sharing. Nevertheless, P2P techniques have been intensively studied by the research community in other fields. Various works have been carried out for their applicability in distributed database systems, distributed indexing and search engines. In this chapter, an overview of existing P2P systems focusing mostly on structured P2P systems is provided. Basic concepts of structured / unstructured P2P overlays and existing protocols are analysed, while emphasis on DHTs is given. Other major topics described in this chapter are proposed techniques for DHT-based indexing schemes and distributing indexing of multidimensional data. Moreover, an overview of the basic concepts of Grid computing is provided. Finally, an indexing scheme based on Space Filling Curves is proposed for indexing metadata descriptions of annotated content.

2.1 Peer-to-Peer Overlays

Peer-to-peer computing has been widely accepted as a robust, easily deployable and self-organizing paradigm on which fully distributed and scalable applications can be built. The self-organizing nature of P2P networks is illustrated in their ability "to automatically adapt to the arrival, departure and failure of nodes" [RD01]. Another significant feature of such systems is

the lack of centralized structures to control fundamental procedures of their operation. Each node participating in the overlay is referred as a *peer* and can act both as a client or a server in an autonomous manner. Peers contribute their resources to the system, which can be used in a cooperative mode for distributed processing and storage. A P2P overlay is capable of operating unsupervised with no central coordination and dynamically adjusting itself to unadvertised node arrivals and departures. Although various completely decentralized schemes exist, P2P systems with some centralized structures have been also proposed in the literature, especially during their early days. In the centralized P2P approaches, a central server (or multiple instances of this server) is used for the core functionalities. For example, in the case of Napster [SGG03] this central point serves as a directory service containing the locations of the stored data items and is queried about the nodes responsible for the items to retrieve. Nevertheless, the existence of central points maintaining information about the whole overlay usually leads to solutions that do not scale, since one node gets all index traffic creating a single point of failure. In general, central points with crucial information need to be avoided in distributed architectures, if possible.

Distributed applications using a P2P architecture have become a very active field of research, mostly because of the scalability offered by the decentralized control model. The most popular applications deployed on top of P2P networks are about large-scale and geographically distributed file sharing and some representative examples are Gnutella2 [Gnu], Kazaa [KaZ], Bittorent [Bit], eMule [eMu], iMesh [iMe]. Nevertheless, some of their prominent features such as the unsupervised and dynamically adjusting operation, the scalability and the fault-tolerance provided by such overlays have made P2P technologies attractive for the development of many Web and data-intensive applications. In most cases, a P2P overlay is built on top of the physical network topology implementing various operations utilized by applications coming from different disciplines. For example, P2P technologies have been utilized for the construction of networks for content delivery and streaming, for the establishment of social networking, for providing distributed search engines and for the creation of communication networks. Skype [Sky] is one of the most well-known P2P applications in the communication domain with millions of users.

For the accomplishment of a specific task in a P2P overlay, a peer needs to communicate with other peers by exchanging messages through the available communication channels represented as *links*. A significant posed question is how to organize the peers and their among links in the overlay. Although various implementations of P2P overlays exist as discussed in [LCP+05] and [ATS04], two basic classes of P2P overlay networks are distinguished: Unstructured and Structured.

Unstructured overlays do not require the organization of the connections by a specific structure [KXZ05] and there is neither a centralized directory nor any precise control over the network topology or file placement [LCC+02]. Unstructured overlays rely mainly on randomized algorithms for their construction. In these approaches, each peer maintains a list of neighbors

and an overlay network that resembles a random graph is constructed. As described in [RM06], the lookup of a stored data item ends up to be flooded across the network. In more detail, a query is forwarded from a node to its neighbor and each visited peer evaluates the query locally in its own content. To avoid the cost of redundant traffic, searching methods based on constrained broadcast and random walks have been proposed [TR06]. Unstructured overlays do not enforce any coupling between topology and data items, even though the resultant topology may present some properties. Thus, an observed weakness regards queries for not widely replicated content; these queries are sent to a large fraction of peers leading to excessive bandwidth consumption. Moreover, there are no guarantees that remote or unpopular data items will be found, due to the limit of lookup horizon, while locating relevant items can become problematic as the network grows. A *hybrid* approach is also followed in this category of overlays, where more "powerful" peers are considered and called *superpeers*. These peers act as "brokers" and regulars peers are attached to them, when they join the network. All communication from and to a regular peer proceeds through that peer's associated superpeer.

Structured overlays impose a strict topology, namely the set of connections among P2P members are tightly controlled by the implemented protocol and data items are not placed in random nodes but at specific locations that will make subsequent queries easier to resolve [LCC+02]. The structured overlays are less flexible and extra cost is required for the maintenance of the topology, especially during arrivals and departures of nodes (e.g. updating the routing tables). Nonetheless, the search performance is improved since the overlay is constructed following a deterministic procedure and the location of a data item can be defined precisely. The most common approach to build a structured P2P overlay is based on building a *Distributed Hash Table (hence DHT)* as described in the following Section.

2.2 Distributed Hash Tables

The basic approach followed for the creation of a structured overlay is to organize the overlay through a *Distribute Hash Table* (*DHT*), namely a hash table whose entries are distributed among different peers. The DHT abstraction implements inherently some basic functionalities for the routing of messages among the nodes and provides a general-purpose interface for location-independent naming upon which a variety of applications can be built.

All items - namely nodes and resources (e.g. data items) - to be inserted in a DHT are assigned a unique identifier (ID), referred as *key*. In most cases, the key is assigned randomly from a large identifier space, such as 128-bit or 160-bit space. A DHT protocol implements an efficient and deterministic scheme, where a data item is uniquely mapped to the identifier of a node based on some distance metric [BKK+03], making this node "*responsible*" for the specific data item. The well-defined structure is the one that enables efficient discovery of data items, if their keys

are known. To ease the network construction task and to effectively balance the data load, most structured peer-to-peer systems use uniform hash functions (like cryptographic functions) for assigning identifiers to peers and resources. The usage of uniform hash functions ensures that keys are assigned randomly to nodes and resources and thus the load is uniformly distributed over the key-space.

When looking up a data item in a DHT-based overlay, the network address of the node responsible for this data item is returned and the node is located based on the properties of the DHT, since peers lack of global knowledge about the network. This is accomplished by routing a lookup request closer and closer to the responsible node, until it is found. A node maintains information (i.e., the NodeID of the peer and the IP address) about a small portion of other nodes, called "*neighbours*" in its *routing table*. Each node is responsible for storing keys and managing operations for resources close to its own ID. A lookup operation can be initiated by any peer and terminates to the node responsible for the specific key. The number of visited peers is small and the DHT overlays present logarithmic search path lengths compared to the network size in most cases, namely any data object can be located in $O(\log N)$ overlay hops on average, where N is the number of peers in the system. The idea behind the routing of a lookup for a specific key is that any node receiving a query for a key must be able to forward it to a node whose ID is closer to the key being looked up, until the node responsible for this key is found.

In [AAG+05], a generalization of the existing approaches is presented and a reference model is proposed. In this work, it is also recognized that the main requirements in the construction of an overlay are the following ones: An appropriate *identifier space* needs to be chosen and a method to be provided for mapping the resources and peers to this space. Moreover, the structure of the logical network needs to be defined and the identifier space needs to be effectively managed by the peers. Finally, a routing strategy and a maintenance strategy must be implemented by the adopted protocol. It is important that all these requirements need to be addressed in a manner that enables the following features of the overlay [AAG+05]:

- ***Efficiency*** of the routing and maintenance mechanisms.
- ***Scalability*** as the number of peers and resources increase without significant performance degradation.
- ***Self-organization*** during nodes' arrivals and departures without requiring any centralized control.
- ***Fault-tolerance*** to the node departures, as these environments are dynamical and error-prone.
- ***Cooperation*** among the peers to achieve effective routing, exchange information, etc.

These properties of the DHT-based overlays make them popular among the P2P networks: The aggregation of enormous storage space and computational resources is achieved, while the lookup cost is minimized. As it has been already described, the DHT-based protocols for the construction of structured overlays implement at least two basic mechanisms (or provide all the required functions for their implementation) in accordance to the put/get interface of the *Hash Tables*:

- *insert*(*key,value*): A key is mapped to the inserted value with the use of a hash function and the (*key,value*) pair ends up to the responsible node (or nodes if replication is enabled) for the key. Also, the same procedure is followed for inserting a node in the overlay, which is also mapped to an identifier. When a node is inserted in the overlay, then some procedures need to be followed for the maintenance of the structure of the overlay.
- *lookup*(*key*) : A (*key,value*) pair can be retrieved from the overlay, when a lookup message is routed to the peer responsible for this key.

The procedure followed during the lookup operations guarantees their effectiveness but it also imposes limitations for the adoption of DHTs overlays by various applications. The reason is that the use of the hash functions eases the construction of the overlay and manages to balance the data load but limits the use of data-oriented overlays to simple exact identifier lookup capabilities, where the key of a stored data item must be known before issuing a lookup request. More complex queries, such as aggregation, multi-attribute, range, join and similarity queries cannot be resolved only with the use of this simple lookup functionality and thus the access to data becomes highly ineffective. The processing of such types of queries requires the implementation of additional indexing structures and techniques. Therefore, data processing based on their semantics cannot be successfully tackled by conventional DHT-based systems.

As applications demand more complicated types of queries, capable of searching the data content and exploiting inter-relationships between data, research efforts have focused on developing corresponding algorithms and mechanisms. The resolution of such types of queries is mainly encountered with the following basic approaches met in the bibliography:

- The proposed overlays rely on an existing DHT protocol and either modify/replace the hash function or add additional indexing structures on top of the DHT and introduce new methodologies for the resolution of queries. The proposed indexing schemes are adapted to the data structure and usually focusing on the efficient resolution of a specific type of query.
- The distribution of the dataset among the peers does not occur with the use of a hash function and thus these overlays do not directly utilize any existing DHT protocol.

The methods described in this thesis result in the design of architectures belonging to the first category. The goal is that the DHT overlay is utilized to address issues about scalability, robustness, load balancing and replication, while new methods are proposed for the efficient resolution of complex queries, mainly focusing on aggregation and range queries and taking advantage of properties in the structure of data. Aggregation queries can be correlated with range queries as well. In the proposed system, ranges can be also implicitly introduced through the support for hierarchical aggregation. Nevertheless, the transformation of an aggregate query to a range query is not always straight-forward or feasible, especially when the queried values are not enumerable.

DHT Implementations

Various DHT protocols exist with different organization schemes for the data items and the key space and follow different routing strategies. Some of the most popular implementations are Chord [SMK+01], Pastry [RD01], Kademlia [MM02] and CAN [RFH+01]. These protocols differ mainly in the shape of the identifier space and consequently the function that measures the distance between IDs in the specific virtual structure [LCC+02]. In this section, some of the basic types of DHT-based protocols are summarized. Emphasis is given to the DHT-based overlays related to the used overlays in the proposed systems of this thesis.

Chord [SMK+01] uses consistent hashing [KLL+97] to map keys to nodes responsible for them. The utilized hashing function balances load among nodes with high probability. All identifiers including both identifiers for nodes and data items are placed clockwise around a circle. Each peer maintains routing tables (containing *finger tables*) containing information about $O(\log N)$ other nodes in a network with N nodes. In the "Chord ring", a key k is assigned to the first node whose identifier is equal or follows k in the identifier space and this node is called the *successor node* of key k. Each node needs to definitely know its current successor node on the identifier circle, so as a lookup for a specific key can be passed around the circle via these successor nodes and get resolved with this simple approach. Figure 2.1 shows an identifier circle consisting of ten nodes storing five keys and the path followed during the lookup of a key. The lookup request is passed around the nodes of the Chord ring via the successor pointers, until the responsible node is encountered. A more scalable solution for locating the node responsible for a queried key is achieved by maintaining more information in the finger tables apart from the successor node. So, up to m entries are maintained in the finger table of a node, where m is the number of bits in the ID space. The $i-th$ entry in the table at node n contains the identity of the first node s that succeeds n by at least $2^{(i-1)}$ on the identifier space. The notion behind this approach is that a node knows more about its close nodes than its distant nodes. During a lookup operation, the involved peer forwards the lookup message to a node included in its finger table whose identifier is highest but not greater than the searched key. Since each node is aware of nodes at

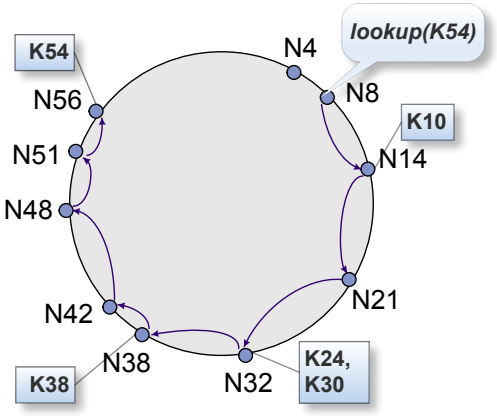


Figure 2.1: *The followed path during a lookup for the key K54 starting from node N8 in a Chord ring. The query is forwarded simply via the successor pointers.*

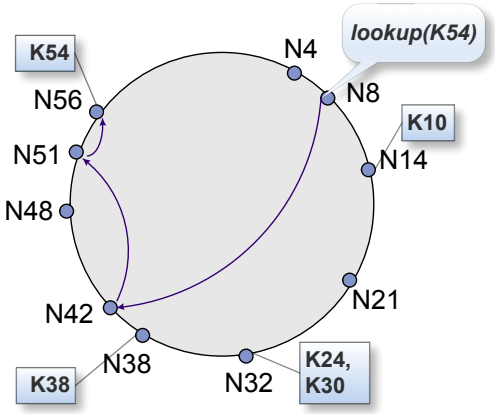


Figure 2.2: *The followed path during a lookup for the key K54 starting from node N8. The information stored in finger tables is used for the query forwarding.*

power-of-two intervals around the ring, each node can forward the query at least half way along the remaining distance. The number of nodes contacted until the successor node in a N-node network is found is $O(\log N)$. Figure 2.2 exhibits the resolution of the same lookup as in Figure 2.1. The more scalable solution using the information stored in the finger tables for the routing of the lookup is depicted in this Figure. The arrivals and departures of new nodes require the updates of the finger tables.

Pastry [RD01] is another self-organizing overlay using a circular namespace, where the position of a node is defined by its identifier (hence *NodeId*). The NodeId is assigned randomly to a node and ranges from 0 to $2^{128} - 1$. Pastry is similar to Chord but its routing is based on address prefixes, namely a message is forwarded towards nodes that share successively longer address prefixes with the destination. This is accomplished by considering that each NodeId and key is a sequence of digits with base 2^b, where b is a numerical parameter (its typical value is 4). As described in [RD01], a node forwards the message in each routing step to a node whose NodeId shares with the key a common prefix that is at least one digit longer than the prefix that the key shares with the present NodeId. If such a node does not exist, then the message is forwarded to a node whose NodeId shares a prefix with the key as long as the current node, but it numerically closer to the key than the present NodeId. Under normal operation, Pastry can route a message to the closest node for a given NodeId or key in less than $\lceil \log_{2^b} N \rceil$. In case of concurrent node failures, the delivery of a message is guaranteed unless $\lfloor |L|/2 \rfloor$ nodes with adjacent NodeIds fail simultaneously, where $|L|$ is a configuration parameter.

This routing procedure is accomplished by using information maintained in the *routing table*, the *leaf set* and the *neighborhood set* of a node. The routing table of a node contains $\lceil \log_{2^b} N \rceil$

rows with $2^b - 1$ entries each. The n-th row of the routing table contains a respective entry for a node whose NodeId has the same prefix for the first n digits, but whose $n + 1$th digit has one of the $2^b - 1$ possible values other than the $n + 1$th digit of the present NodeId. If a node with such a NodeId does not exist, then the routing table entry is left empty. The leaf set is the set of nodes with the $l/2$ numerical closest larger NodeIds and the $l/2$ nodes with numerical closest smaller NodeIds, relative to the present node's NodeId. The neighborhood set is a set of l nodes that are near the present node according to the proximity metric and it is used during the addition of a new node or a recovery process.

Kademlia [MM02] is a symmetrical DHT-based overlay that uses an XOR metric to measure distances between the leaves of a binary tree. Both data items and nodes are assigned unique identifiers from a unified address space. Items to be inserted in the DHT are *(key, value)* pairs, where the value is the data item to be stored to the node with the closest ID to the key. The notion of distance between points in the identifier space is defined by the XOR metric. The Kademlia protocol locates data items based on their keys, only if their exact key is known in advance. The query messages are routed in the overlay according to the information that each node maintains for other peers. This information is acquired by the messages that a peer receives. The symmetric feature of the XOR metric allows Kademlia participants to receive lookup queries from precisely the same distribution of nodes in their routing tables. Kademlia appeals as an appropriate selection due to its simple routing table structure and its consistent algorithm throughout the lookup procedure.

The basic operations supported by the Kademlia protocol [MM02] are:

PING: This RPC probes a node to see if it is online.

STORE: At first, the node initializing the STORE operation calculates the key for this data item, usually a hash over its content. Then, it searches the network in several steps, until the closest nodes to that key are discovered. Afterwards, the data item is stored in these nodes.

FIND_NODE: This operation returns the *kappa* closest nodes to the asked node ID, where *kappa* is a system parameter.

FIND_VALUE: When a node of the Kademlia network is instructed to lookup a value in the network, it issues α parallel queries (FIND_NODE) to the *kappa* closest nodes it is aware of. It continues the process until the *kappa* closest nodes to the target key are found. Then, the node issues a FIND_VALUE RPC to retrieve the value. The system wide parameter *kappa* also specifies the number of copies maintained for each data item, and controls the size of routing tables in peers. Both α and *kappa* variables are set at each participant node and affect only local service performance.

A variety of structured overlay networks exists as well, implementing different structures or routing strategies, etc. For instance, CAN [RFH+01] deploys a *d-dimensional* Cartesian coordinate space, which is dynamically and completely partitioned among all the peers of the overlay. Each node is responsible for a distinct and individual zone of the d-dimensional space and maintains a routing table with $O(d)$ entries and any node can be reached in $O(dN^{1/d})$ routing hops. Two nodes are neighbors if their zones share a *d-1* dimensional hyper-plane. Each data item is assigned a unique point of the d-dimensional space with the use of a uniform hash function and it is assigned to the node responsible for the zone that this point lies in. Tapestry [ZHS+04] follows also a prefix-based routing similar to Pastry, but differs in its approach to achieve network locality and to replicate objects. P-Grid [Abe01] is a structured overlay that implements a distributed search tree. Each node holds a part of the overall tree and its position in the overlay is defined by its path in the tree and is responsible for the data items whose key begin with the bit representation of its path. In P-Grid, the queries are resolved based on prefix matching, while the key-space is dynamically partitioned among nodes depending on the load resulting in even distribution of the keys among the nodes even for non-uniform load distributions.

2.3 Indexing Techniques in Peer-to-Peer Networks

Various research works investigate how P2P systems can gain the strengths of data management in semantics, data transformation and data relationships, which could enable them to provide complex queries capabilities. The survey of Risson and Moors [RM06] presents a summary and comparison of search methods in P2P systems. A large number of existing works emphasizes on the creation of powerful indexing schemes and how these schemes can be adapted to specific types of queries, such as aggregation queries, range queries, similarity queries, structure queries, conjunctive queries, etc. Indexing techniques either followed in traditional database systems or combining the properties of the underlying overlay and the targeted type of queries have been proposed. At first, an overview of the works aiming at providing database functionalities in a distributed environment is presented.

PIER [HHB+03], [HCH+05] proposes a distributed architecture for an Internet-scale distributed database aiming at supporting multiple operators such as selection, projection, union, join, group-by, etc. A DHT-based overlay is used for query routing, so as the system to scale up to thousands nodes connected over the Internet. Each tuple to be inserted in the PIER overlay is self-describing containing its table name, column names and column types. On top of the DHT overlay, a secondary indexing scheme is built creating a tree among the nodes. The purpose of this index is to reduce communication cost by gathering tuples to a single node during the resolution of aggregates and joins. Thus, the nodes are organized in trees; a node involved in the computation of such queries computes its local result and forwards it one hop closer to the root

level. Each node also supports a local Scan method allowing the query processor to view all the objects that are present to the local node.

Piazza [TH04] is another Peer Database Management System (PDMS), where each node maintains its data modelled in XML. This work focuses mainly on achieving better performance of the search strategies based on reformulations of queries, pruning of redundant optimizations and pre-computing of semantic paths. The proposed approaches enable nodes to operate in a completely decentralized fashion, utilizing the standard lookup operations to refine their local knowledge.

PeerDB [OTZ+03] is a full-fledged data management system that facilitates sharing of relational data without predetermined shared schema. A local database for the data management exists in each node of PeerDB, while a "Local Dictionary" stores the associated metadata for each created relation and another dictionary structure stores the metadata of objects that are sharable to other nodes. In PeerDB, agents are introduced, which are responsible for the processing of queries. In a first phase, nodes are contacted and return prospective answer metadata. In a second phase, the user selects the relevant metadata and the selected resources are contacted directly and a reformulated query matching to relation name and attributes of the node is posed to the local database.

The pSearch [TXD03] system is built on top of a CAN overlay accommodating documents indexed according to their semantic vectors produced by the Latent Semantic Indexing (LSI). The specific system cannot handle dynamic collections of documents, since the semantic vectors have to be defined a priori and thus the documents of newly-joined peers, with terms that are not encapsulated in the existing vector, cannot be indexed by them.

Mercury [BAS04] is a network consisting of multiple circular overlays, where each overlay corresponds to a different attribute. The nodes are partitioned into groups called attribute hubs and each hub is responsible for a different attribute. The nodes of a hub are organized in an order-preserving manner and each node is responsible for a certain range of a certain attribute. This scheme enables the resolution of range queries which require an ordering of the stored data. A query for more than one attribute is considered as a conjunction of predicates and is routed to all relevant hubs. Since the elimination of the hash function may result in non-uniform partitioning of data among nodes, Mercury emphasizes on load balancing, using random sampling to avoid problems with skewed data distributions.

All these approaches offer solutions for the creation of distributed database systems for storing and sharing structured and heterogeneous data. The works based on structured overlays provide better search efficiency but impose more maintenance overhead. In both cases, the above systems adopt a global indexing strategy for all the tuples stored in the network and this fact results in a costly procedure as the volume of data and the number of schemas and attributes increase. The indexing and maintenance process of large-scale data ends up to be very costly

in terms of bandwidth consumption and time, especially when high update and churn rates are observed.

A more dynamic solution for the creation of a peer-based data management system that indexes only a portion of the stored tuples is PISCES [WLOT08]. The specific system follows a partial indexing strategy for a subset of the stored tuples based on criteria such as query frequency and update frequency. Each node maintains its own database and it also participates in a structured network like BATON [JOV05] and CAN [RFH$^+$01]. The partial index is built upon approximate information gathered by a histogram based approach about the total number of nodes, the total query number, the query distribution and nodes' arrivals and departures. Nevertheless, this approach targets mainly to support queries regarding relational data and does not handle the special case of hierarchies over multidimensional datasets, especially when emphasis is given on the interlinking of data.

Apart from the more generic systems aiming at the sharing of structured data, there are also various research efforts focusing on the design of P2P overlays oriented towards the efficient resolution of a special category of queries and the proposed indexing techniques are intended to serve mostly a specific purpose. The followed approaches target either at introducing hash functions that maintain an ordering of the data items among the nodes or building additional structures, which can be applied on top of existing structured overlays.

An approach to influence the placement of relational objects in a DHT overlay by introducing a hashing function is presented by Gupta et al. [GAA03]. The authors develop a P2P architecture for computing approximate answers for complex queries expressed in the form of SQL statements. The tuples of a relational database are partitioned horizontally and assigned to nodes in a locality preserving manner with high probability using *Locality Sensitive Hashing*. The query processing occurs by locating partitions relevant to the query and approximate results are returned. The quality of the results depends on the complexity of the hashing function, which may also result in load balancing problems.

In [AX02], an effort to support a distributed information system for computational grids on top of a CAN overlay is presented. Each attribute describing a resource is indexed in a different DHT overlay. The adoption of a hash function based on the Hilbert Space Filling Curve tries to achieve the mapping of ranges of values to nearby CAN zones. Thus, this approach focus on the resolution of range queries, which are initially routed to the node responsible for the middle point of the range and then recursively propagated according to three proposed and evaluated 'flooding' strategies. If a query involves more than one attributes, then the query is resolved for each attribute separately and the returned results are concatenated in a database-like "join operation", increasing the cost of resolving a more complex query.

Towards the efficient processing of queries that cannot be resolved with simple DHT lookups, a common strategy is the adoption of additional indexing mechanisms on top of the DHT-based

overlays. An indexing structure that has been mainly used for prefix searches is tries. Tries are a generalization of trees for storing and processing strings in which there is a node for every common prefix. In tries, the actual data is stored in the leaf nodes and thus lookups are resolved by finding the leaf whose label is a prefix of the queried value. In various works, tries are distributed among the nodes of a DHT-based overlay. The Prefix Hash Tree (PHT) [RRHS04] is a trie-based structure indexing binary strings based on their common prefix. The main type of queries supported in this system are one-dimensional range queries. Each node maintains a pointer to each immediate left and right node and a range query can be resolved either in a linear or binary manner. A range query is performed by various DHT lookups and the query cost is data dependent. Datta et al. [DHJ+05] incorporate the trie in the network itself by suitably organizing the routing tables of a P-Grid overlay. The exact match searches are resolved by the mechanisms of P-Grid, while two strategies are proposed for the resolution of range queries. The min-max traversal strategy starts querying one of the bounds contained in the range. The respective node returns the relative data and forwards the query to a peer responsible for the next partition of the key space, until a peer for the other bound of the range is encountered. Another strategy is implemented as well, where the range query is split into smaller ranges, which are posed concurrently. The *Distributed Lexical Placement Table* (DLPT) [CDT07] is a dynamically constructed trie on top of a DHT-based overlay for indexing, enabling service discovery in grid environments.

Other types of trees are also exploited for creating additional distributed indexing structures. P-Trees [CLGS04] are decentralized indexing structures, maintaining parts of semi-independent B+ trees on top of a Chord ring. The created trees are used for the routing of equality and range queries along their paths. Nevertheless a query cannot be guaranteed to terminate – for example, if a node crashes due to complex cross-structure management. The Range Search Tree (RST) is another tree for the indexing of ranges of values built on top of a DHT overlay, such as Chord. Each level of the tree corresponds to a different granularity of data partitioning, while content is registered with all or various levels of the RST and the corresponding physical nodes of the DHT.

An approach based on multi-level bloom filters was proposed in [KP04]. In this work, the described structures are used to summarize hierarchical data and support regular path expression queries. The studied data are XML trees contained in documents and the multi-level filters are used to summarize the documents stored locally in a node and the documents of its neighboring nodes, while nodes with similar filters (and thus similar content) are linked together. These summarization structures are exploited during the routing of path queries over distributed collections instead of searching the actual documents.

A more relative scheme targeting the online processing of aggregate queries is described in the Distributed Online Aggregation (DOA) Scheme [WJOT09]. In this case, the calculation of results is approximate and multiple iterations take place: in each iteration a small set of random samples are retrieved from data sites and assigned to available nodes for processing. The random

sampling among the distributed nodes may bias the calculated result and premises that each node must provide an interface for picking random sample from its database. Apart from this fact, all data needs to be mapped to a global schema, but this is not feasible usually and introduces complexity to the system.

2.4 Indexing of Semantic Data in Peer-to-Peer Overlays

The Resource Description Framework (RDF) is a widely accepted standard in the Semantic Web and has been widely adopted for the representation of semi-structured data. Many approaches combine techniques from the relational databases to build large centralized repositories that index and query such data, as it has been discussed in [HSTW11]. The most representative categories in centralized approaches are the triple stores and vertically partitioned tables. In most cases, the triple stores such as Virtuoso [Vir], 3store [HG03] and RDF-3X [NW10] store RDF triples in a simple relational table. The approach in [AMMH07] proposes a physical organization based on vertically partitioning tables. In this scheme, the triples are spread over multiple two-column tables. Each tuple of a particular table contains a subject and an object column for a particular property that connects them. This approach clearly outperforms simple or naive processing solutions that perform multiple joins on a huge RDF table or *property tables* that cannot handle multi-valued attributes (i.e., a value such as subject linked to multiple values).

Nevertheless, centralized solutions do not scale [MAYU05] and more advanced indexing schemes are enforced on top of triple stores to boost up performance [AMMH07, WKB08]. In Hexastore [WKB08], the previous method, deemed strictly property-oriented and behaving suboptimally for general queries, is amended. The proposed index structure comprises of six distinct indices that treat subjects, properties and objects equally. Each of the indices focuses on a single element of the triplets and defines a prioritization between the other two elements. These indices in effect materialize all possible orders of precedence of the three RDF elements with a 5-fold worst-case storage increase. Both systems in [AMMH07, WKB08] build on similar notions and try to alleviate the scalability issue in existing central RDF stores by improving complex query performance. While they both have particular pitfalls (search is limited to only a few properties for [AMMH07], strictly main-memory evaluation and extreme storage overhead for [WKB08]), they only offer efficient centralized triple management.

As the growing volume of data cannot be handled efficiently in centralized solutions, various distributed schemes have been proposed in the literature. As far as structured P2P systems are concerned, the majority of efforts focuses on distributing RDF data among multiple peers. RDFPeers [CF04] was one of the first efforts to store triples on top of a MAAN overlay [CFCS04]

by hashing the subject, the object and the property and insert the same triple with three different keys in the overlay. ATLAS [KKK+10] uses also RDFPeers for querying RDF data. GridVine [ACMHP04] follows a similar approach to RDFPeers for inserting RDF triples in a P-Grid overlay [Abe01] and storing them in the local database of the node. Another platform for supporting index-based path queries is IMAGINE-P2P platform [ZSL+05], which builds a semantic overlay on top of a structured P2P overlay that provides object location and management services. Triples containing values and the relationship that correlates these values are stored in the semantic overlay, where trie structures are deployed. Path queries are decomposed to the subqueries that are sequentially processed and resolved along the index paths defined by the relationships among the semantic objects stored in a Chord ring.

All these efforts in P2P networks may easily lead to overloaded nodes with poor performance for popular triples (that is, the node responsible for the key of rdf:type). Also, they cannot exploit the semantic information included in this kind of data and they are forced to build additional semantic layers or interfere to the organization of the overlay according to the semantics of data (e.g. [KSHS08], [ZHDR09]) or extend the RDF model. These modifications add more complexity and increase the cost for maintenance of the external structures, especially during update procedures. Also, all these approaches encounter all the problems of distributing query processing, where large volumes of data need to be fetched in the query initiator and joined before they proceed to the evaluation of the rest of the parts of the query.

2.5 Handling Multidimensional Data in Peer-to-Peer Overlays

2.5.1 The Multidimensional Data Model

The multidimensional data model is a model exploited in data warehouses and applications for *On-Line Analytical Processing* (hence OLAP). The purpose of this model is to serve the resolution of complex queries quickly, since the whole analytic task takes place on-line. This model views data in the form of a *data cube* (or *cube*) defined by dimensions and facts, where facts are numerical measures [HK00]. Dimensions are the perspectives or entities that identify and categorize data and form the edges of the cube. Organizing measures in a data cube requires that these measures have the same shape, i.e., they have the exact same dimensions. A typical example is the data cube that maintains the *Sales* of a company. In this case, the dimensions may be `Location`,`Time`, `Customer` and `Product` and the described measure is the total amount of sales.

In relational databases, the N-dimensional data are modeled as N-attribute domains. The cube operator is the N-dimensional generalization of simple aggregate functions [GCB+97] and performs the computation of the aggregation function for all possible combinations of grouping attributes. The relational implementation of a cube is typically a star schema or a snowflake

schema. In star schemas, data are organized into *dimension tables*, *fact tables* and *materialized views*. The fact table is a large central data table containing the bulk of the data with no redundancy. A dimension table represents a dimension and contains a set of attributes, which may form either a hierarchy (total order) or a lattice (partial order). The snowflake schema is a variation of the star schema, where some dimension tables are normalized, thus splitting the data into additional tables so as to reduce redundancies.

It is common in data warehouses for data to be organized at different levels of aggregation with the use of hierarchies. A concept hierarchy defines a sequence of mappings from a set of low-level concepts to higher-level, more general concepts. In the multidimensional data model, dimensions may contain multiple levels of abstractions defined by concepts hierarchies, thus allowing users to view data at different levels of aggregation. In [HK00], the following OLAP operations for the multidimensional data model are defined as follows:

- ***Roll-up*** is the operation that performs aggregation on a data cube either by climbing up a concept hierarchy of a dimension or by dimension reduction. In dimension reduction, one or more dimensions are removed from the given cube.

- ***Drill-down*** allows the navigation from less detailed data to more detailed data, either by stepping down a concept hierarchy of a dimension or by introducing additional dimensions.

- ***Slice and dice*** operation performs a selection on one dimension of the given subcube resulting in a subcube.

- ***Pivot (or rotate)*** is a visualization operation that rotates the data axes in a view in order to provide an alternative presentation of the data.

The computation of a cube is a procedure that involves the scanning of the original data, so as to apply the required aggregate function to all groupings and generate the relational views with the cube content. Due to the fact that the increase of dimensions results in an exponential growth of the size of the data cube, various methods have been developed for the computation of data cubes in centralized systems. In the MOLAP approaches (e.g. [ZDN97]), a data cube is stored in a multidimensional array. The main disadvantage of this method is that cubes are sparse in most real-life applications with a large number of empty cells; a fact that results in the inefficient management of the storage space. Another major category is ROLAP cubing methods. The Bottom-Up Cube (BUC) [BR99] is a method for the computation of the cube in a bottom-up fashion, starting with the calculation of the *ALL* value. It also computes only those group-by tuples with an aggregate value above some predefined minimum support threshold. Various other methods

have been based on BUC, e.g. CURE [MI06] which uses a similar execution plan, while it support hierarchical data and removes all forms of redundancy for efficient space utilization. A third category is the Graph-Based methods, which use tree-like structures for cube construction and efficient storage. Dwarf [SDRK02] is a highly compressed structure and achieves that by eliminating prefix and suffix redundancies. In [SDKR03], the initial Dwarf structure has been extended so as to support aggregate queries on every level of a dimension's hierarchy. QC-trees [LPZ03] is another compressed structure presenting similar redundancy reduction capabilities and supporting roll-up and drill-down operations. Nevertheless, these approaches present increased complexity as far as the construction of their structures is concerned.

2.5.2 Indexing techniques in Peer-to-Peer Overlays

As the use of data following the multidimensional data model increases more and more in various applications, the users tend to issue queries requiring complex processing (e.g., aggregation queries, similarity queries, range queries). The simple lookup mechanisms of traditional P2P protocols are not adequate and show poor performance for multi-attribute (or multidimensional) queries. For example, a query about all the products of a company created by a specific manufacturer and sold in a specific time period cannot be answered efficiently by the simple DHT lookup operation.

The indexing of multiple attributes for answering such types of queries requires that the generated keys in a DHT overlay enclose such semantic information. A solution is to use the values of the indexed attributes (or dimensions) in the generation of keys. In the case of hierarchical multidimensional data studied in this thesis, the result of this solution would be to insert the same fact multiple times in the overlay; one time for each level of the hierarchy and for each possible combination among dimensions. The multiple insertion of the same fact is a followed approach in DHTs to ensure load balancing and robustness. Nevertheless, a large storage overhead would be introduced in the case of multidimensional hierarchical data, where the number of possible combinations is not small and increases exponentially to the number of dimensions.

Another approach is the placement of data among the nodes in such a manner that such types of queries to be routed to the responsible nodes with low communication cost and bandwidth consumption. The exploitation of Space Filling Curves (hence SFC) for the construction of functions that take into account the values of the attributes for the generation of IDs fall into this category. In these approaches, it is assumed that the attributes form a multi-dimensional space, which is commonly mapped to one dimension and vice versa using a SFC. The property of SFCs exploited in this approach is that there is high probability of the creation of locality-preserving orderings. For this reason, SFCs have been also used for indexing of multidimensional data in traditional applications such database systems [Bay97, TH81, LK00], image processors. The mappings

produced by a Hilbert SFC exhibit better clustering properties as shown in [MJFS01], namely the locality between objects in the multidimensional space is best preserved in the linear space. P2P indexing schemes using SFC-based function can also be found in the literature.

Squid [SP04] is a Chord-based overlay accommodating multi-attribute data, while it supports partial keyword, wildcard and range searches. Each data item is described by a set of d-attributes and is mapped to an ID by a function that maps the attribute values to a Hilbert index. If a query includes value ranges, then the query is transformed through the Hilbert mapping into a series of one-dimensional interval queries, which are resolved in the Chord ring. To avoid a flooding-like search, a recursive procedure is followed with the use of an embedded prefix tree: At each search step, longer prefixes of candidate clusters are looked up. If a node is responsible for the IDs with the prefix in question, then it queries its local database. In the opposite case, it initiates new lookups for more specific sub-clusters. Despite the recursive searching of SFC clusters and the pruning of nodes which do not contain relative data elements, the whole procedure imposes relatively large computational and messaging costs and problems related to load balancing accrue. To avoid these problems, Squid relies on the fact that the d-dimensional keyword space is sparse and so the data items are assigned to peers roughly in the same way.

CISS [LLK+07] is another framework that utilizes SFCs for searching multidimensional data in the context of massively multiplayer online games and P2P catalog systems. In this work, a hierarchical naming structure for catalogs is also considered. A locality preserving function that uses the Hilbert SFC produces mappings from multiple bit keys to a one-dimensional key and it is used for the clustering of objects among the nodes, so as efficient data updating and multidimensional range query routing to be achieved. CISS focuses on the resolution of queries including ranges of attribute values in a 2-dimensional space. The queried ranges are mapped to candidate key ranges, and the node responsible for the first key of a candidate cluster is looked up. Afterwards, this node forwards the query to succeeding peers, until all relevant objects are retrieved. The authors of CISS adopt the forwarding-based routing strategy to optimize query processing and avoid congestion nodes that they believe that are introduced in Squid by the prefix based routing.

An SFC-based function for producing single-dimensional indices is also used in SCRAP [GYGM04] and the produced keys are assigned to peers of a SkipGraph network [HJS+03]. In this scheme, range queries over multiple attributes are studied and the routing of the queries for candidate SFC clusters is implemented through SkipGraph. Nevertheless, the number of calculated ranges may be very large and definitely depends on the refinement level of the SFC curve. ZNet [SOTZ05] proposes a similar approach to SCRAP; the main difference is that the multidimensional space is dynamically partitioned using a Z-curve. The nodes may be responsible for continuous ranges of Z-addresses of different lengths according to the order of the curve for the

specific interval. A range query is initially forwarded to the nodes assigned with the shorter prefixes, before being further refined by computing the next recursion of the curve for producing more specific candidate intervals.

The exploitation of SFCs for the assignment of one-dimensional keys to objects described by multiple attributes aims at the placement of data in a locality preserving manner, thus achieving the clustering of "similar" objects. Nevertheless, it is assumed in these approaches that appropriate encoding schemes exist to map the attribute values to a specific number of points in each dimension. For different types of attributes (e.g. numeric, date, string type), different encoding schemes need to be developed. These schemes pose also the need for knowing or having at least a good estimation of the possible values of the attribute, otherwise they become highly ineffective for the mapping procedure. At the same time, another posed restriction by this method is that all dimensions have equal cardinalities. The cardinality of each dimension increases for higher approximations of the curve, but this results in the increase of the complexity of the calculation of the mappings, especially when the effort is to define the mappings of ranges of values in the multidimensional space to a series of ranges at the level of the SFC. Reducing the number of possible coordinates of the dimensions may lead to a completely unbalanced assignment of items to nodes.

If an SFC-based function is exploited for data with multiple, hierarchical attributes, then each level of each hierarchy can be considered as a different dimension. In this case, highly dimensional spaces will occur and the SFC-based functions will loose the desired properties. Another solution is to handle the hierarchy inside the encoding function and adopt an approach similar to CISS. Besides of the added difficulty for creating such an encoding, the query resolution is not effective as well. If the whole path from the most generic level is not defined in all dimensions or at least the values for the most levels are not given, then numerous, non-contiguous SFC segments are determined. This is a computationally intensive and time consuming procedure, which is also not effective, as the query is actually flooded across the network.

Various works also exist that define their own data organization schemes for indexing multidimensional data in structured overlays. Apart from SFC-based functions, some works define their own functions for assigning data objects described by multiple attributes to peers. For instance, the properties of a Kautz graph are exploited by algorithms for single and multi-attribute hashing that preserve entire and partial order in the structured overlay called Armanda [LCLC09]. Each peer in the overlay is responsible for a specific interval of an attribute's adjacent values and searches for ranges common prefixes are enabled. Another major category of systems hosting such types of data includes extensions of existing DHTs, e.g., Chord or CAN. MAAN [CFCS04] extends Chord to support multi-attribute and range queries. It implements a locality-preserving hashing function and one different registration for each of the resource attributes is performed. MAAN supports only predetermined schemata with a fixed number of

attributes. MURK [GYGM04] implements a distributed kd-tree in order to partition the data space to rectangles and assign them to participating nodes. Each node is aware of its neighbors and queries are forwarded along these links. A distributed quadtree is implemented for indexing and querying of spatial data in [THS07]. The centroid of each quadtree block is hashed and inserted in a Chord overlay. In all these types of aforementioned approaches, the effort is towards the elimination of the random assignment of data in a structured overlay and the preservation of an ordering of the stored values. These works mainly focus on the effective resolution of a specific type of queries and require additional techniques to acquire properties inherently supported by a DHT, such as logarithmic routing, recovery after node arrivals and departures, load balancing and replication.

2.6 Grid Computing

Grid computing [FK99], [FK03] focuses on studying distributed infrastructures, where users share geographically distributed resources integrated under a common middleware. It adopts the principles of the software model called *Service Oriented Architecture* (SOA) and provides seamless access to services deployed in a distributed manner. The basic characteristic of these systems is the existence of explicitly defined rules and policies to enable flexible, secure and coordinated resource sharing among dynamic virtual collections of users, named *Virtual Organizations* (also referred as VO). A Virtual Organization is as a set of individuals and/or institutions defined by highly controlled sharing rules, with resource providers and consumers defining clearly and carefully just what is shared, who is allowed to share and the conditions under which sharing occurs [FK99].

The grid middleware is responsible for hiding the heterogeneity and complexity of the underlying physical infrastructure and provides uniform access to the users. Apart from that, the middleware comprises of services critical for the operation of the infrastructure, such as the coordination of the use of resources, monitoring and accounting of the infrastructure, data management, job execution and progress monitoring and other services related to the operation of the system. Although the execution of applications and the storage of data may occur among multiple sites, the design of some of the basic functionalities is constrained to follow a centralized approach. To avoid flash crowd situations and achieve better service availability and response time, these service are replicated among multiple instances. Nevertheless, this strategy has been proven insufficient in many cases and more distributed approaches are required.

Grid infrastructures have been widely used for the execution of complex scientific and business applications. These applications are characterized by a need for complex computations on large and distributed datasets, as well as their storage and dissemination across multiple sites. Depending on the scope to serve, grid systems can be characterized mainly as *Computational*

Grids or *Data Grids*. In [FKT01], a Computational Grid system is defined as an environment consisting of one or more hardware- and software-enabled environments that provide dependable, consistent, pervasive and inexpensive access to high-end computational capabilities. In these environments, most emphasis is given on the seamless resource sharing aspects in a collaborative virtual organizational world and the majority of the resources provide mainly computational power. *Data Grids* are wide-area distributed infrastructures of heterogeneous resources capable of managing immense amount of data. The core services for accessing heterogeneous storage resources, storing, transferring and searching large datasets are described in [CFK+00]. Existing grid testbeds may combine features from both categories so as to serve the needs of the users.

Nevertheless, the data-intensive oriented Grid systems have been recognized as candidate platforms, where DHT-based mechanisms can be introduced to provide efficient and scalable services for discovery, search and transfer of the stored data. Some of the existing critical services can be only executed as single instances since they need information about the whole infrastructure for their operation. The design and the deployment of such systems can take advantage of methodologies enforced in P2P networks for the management of large amounts of data and the increase of availability, when the amount of stored data and the number of performed operations increase. These methodologies have been exploited to transform services executed in a centralized manner into scalable, reliable and concrete services.

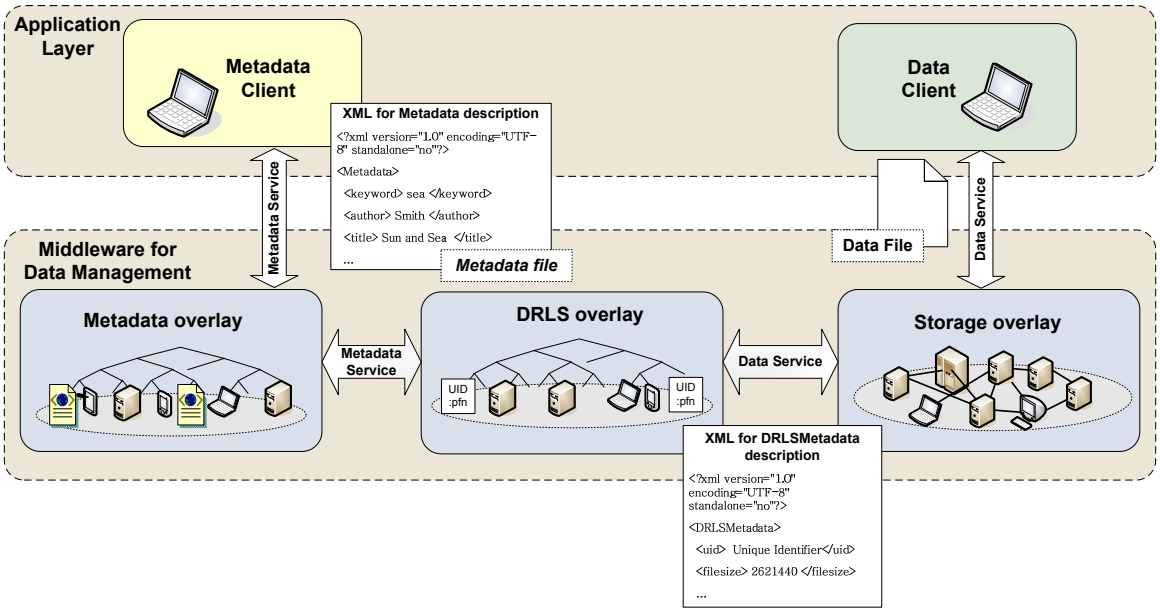


Figure 2.3: *Overview of the architecture of Gredia data middleware*

2.7 Exploiting Space Filling Curves for a Distributed Metadata Catalog

The properties of Space Filling Curves (hence SFCs) can be exploited for data items described by multiple dimensions, while the semantics of these values are maintained. As it has been already discussed, various works in the bibliography examine on how to efficiently handle such data and perform fast and accurate exact match and range lookups.

This section describes a technique that exploits the properties of SFCs for the design of a scalable and distributed Metadata service in the Gredia platform [GRE]. The Gredia platform provides a Grid-based infrastructure that allows developers, professionals and users to share annotated multimedia content. An overview of the middleware architecture developed for the Gredia platform is shown in Figure 2.3.

The Metadata service hosted by the *Metadata* overlay provides a search mechanism for the annotations of the multimedia content stored in the Gredia platform. Each data file containing the multimedia content is described by a set of metadata attributes following a predefined metadata schema designed according to the needs of the users. The multimedia content is annotated along multiple dimensions as it deemed important to characterize the file and make it searchable upon. The metadata values of each data file are included in its *metadata file*. Users should be able to perform efficient point and range queries over the multiple attributes that index each file.

The role of the Metadata service is very important for the search and retrieval of the actual content. The approach followed in Grid systems is the creation of centralized catalogues for each VO, which maintain the information about the stored data of the users providing restricted functionality. As the number of users and the volume of data increase, this solution fails to serve effectively the requests of the users. With the proposed scheme, meta-data are stored among the nodes participating in a DHT overlay built according to the Kademlia protocol and the number of visited peers is dynamically minimized and consequently so is the bandwidth utilized to collect answers. Moreover, fault-tolerance is automatically handled by the underlying DHT's inherent replication mechanism, without influencing the effectiveness of the query engine.

The approach followed to enable the resolution of point and range queries for multi- dimensional data exploits the fundamental property of SFCs. An SFC continuously maps a compact interval to a d-dimensional space and vice versa. SFCs preserve locality, so that points in the 1-dimensional space are mapped to close points in the d-dimensional space.

The SFCs are utilized in the *Metadata* overlay in order to influence the data placement in the DHT without changing the routing algorithm of the DHT protocol. The goal is metadata files with relative attributes to end up in nodes with close IDs, so as to reduce flooding over the network for range queries.

In the proposed multidimensional indexing scheme, the set of d attributes to be indexed is considered to form a d-dimensional space. Each point in this space represents a combination of values for these indexed attributes. The points of the d-dimensional space are mapped down to a single dimension by a SFC, such as the Hilbert curve or the Z-curve. The result is the partitioning of the d-dimensional space into 2^{kd} cells, which in turn are mapped through the SFC to 2^{kd} points of a single dimension. In more details, the d-dimensional space is divided in d-cubes that are mapped to intervals of the SFC. Without loss of generality, each hypercube is considered to be a point of the d-dimensional space and the corresponding interval in the SFC to be a point of the single dimension.

A basic property of SFCs is their recursive generation. Initially, the d-dimensional space is subdivided into 2^d cells mapped into the First Order Curve, where k is considered to be equal to one. In the next recursion, each cell is subdivided 2^d more times and is mapped into the Second Order Curve. The number of recursions is indicated by the factor k called *approximation order*. This factor defines the number of space partitions and thus the precision of the algorithm.

The derived values in the single dimension represent the keyset of the Kademlia-based DHT overlay. Each key is kd bits long and each node in the overlay manages data in one contiguous range of the SFC. The mapping of a combination of values to a specific key is done by the SFC module, which applies the hash function based on the SFC to produce the key used in the DHT

overlay. The produced key maintains the information included in the given values and the mapping takes place among the values of the d-dimensional space to the single dimension and vice versa.

The SFC module may use either the Hilbert curve or the Z-curve. The Hilbert SFC presents the best clustering properties [MAK03], [MJFS01]. The mapping of a point in the d-dimensional space to its position in the SFC and vice versa are not simple and the difficulty increases analogously to the number of dimensions. The generation of the SFC can be performed non-recursively by using circular shift and exclusive-or operations on bytes according to the Butz algorithm [But71]. Based on this algorithm, the mapping procedures from the d-dimensional space to one dimension and vice versa have been implemented. The traversal of the SFC is done by generating the curve recursively. Hilbert SFC can be approximated in a higher order by a combination of First Order Curves appropriately oriented. The calculation of the mappings using the Z-curve is simpler and the mapping is calculated by interlacing the binary digits of each dimension index.

A problem to tackle when deploying the proposed multidimensional indexing scheme has been the segmentation of each dimension according to the number of possible values for each attribute. Since the SFC-based method for enabling range queries over multiple dimensions presumes constant-bit indices, mapping functions of the possible attribute values to the available values in each dimension are needed. The types of attributes that have been recognized are *string*, *date* and *categorical*.

The categorical attributes are easily manageable, since their range of possible values is known in advance and can be directly mapped to the available index space. Although range queries are not a common case for categorical types, the imposed ordering at the dimensional level has been exploited to enable the retrieval of metadata files characterized by more than one category. In this case, the query is rephrased by the metadata module according to the ordering convention followed when encoding. It is possible that multiple distinct values are joined in one or more ranges when the search query reaches the overlay.

For string and date types, a simple character-coding scheme has been developed to produce bit representations. However, the number of encoded characters that comprise the dimensional index depends on the number of bits available for the attribute at the overlay level. Taking into account that the number of index bits in each dimension is equal, the cardinality of the SFC is dominated by the most numerous attribute. However, instead of scaling all attribute representations up, an approximation order of the used curve that requires a reasonable amount of bits – depending on the application – has been selected and then encoded values are respectively "cropped". The encoding results in alphabetical ordering of strings on the corresponding dimensions. The goal is to avoid using a large number of bits, so searches refer to more restricted search spaces, thus end up to less nodes and are served more efficiently. Further filtering may follow in case that the string value contains more than the allowed characters. To avoid increasing the

number of mapped attributes excessively and thus the complexity of calculating the mappings, the most critical attributes needs to be chosen for indexing according to the search frequency of the users. Naturally, the rest of the attributes can still be included in the metadata file for application-level selection (i.e., attributes defined by the MPEG-7 standard [MPE7] for the multimedia content).

The use of the SFC module does not influence the procedure for the insertion of a metadata file in the overlay. A key is generated according to the values of the indexed attributes, which maintains the semantic information of these values. Finally, the file ends up to the node responsible for its key.

The retrieval of a metadata file requires knowing the exact key used during the insertion of the metadata file in the DHT. This means that the native lookup procedure is very restrictive. If the combination of the indexed values for a specific file are known in advance, then the SFC module generates the key used during the insertion phase of this file and the DHT lookup returns the requested file.

Nevertheless, this search functionality is not usually adequate and the DHT mechanisms need to be enhanced to support more complex queries. For example, when at least one value of the indexed attributes is not defined, then the whole range of values in this dimension is searched. A range in one or more dimensions can be narrowed down by specifying a starting and an ending value for the corresponding attribute(s). Taking advantage of the properties of SFCs, the ranges in the multi - dimensional space can be converted to a series of corresponding ranges at the level of the SFC. Nevertheless, it needs to be considered the fact that a simple query even with only one specified range in one dimension may produce numerous non-contiguous segments on the curve, while some of the segments may even be single identifiers.

The resolution of range queries results in a procedure consisting of two basic phases. In the first phase, clusters of SFC points answering the query are determined. In the next phase, lookup operations for these cluster(s) start. Determining the SFC segments may be complex and computationally intensive, thus the calculation of the candidate ranges of IDs can be done using the feedback of the visited nodes. Assuming that churn is not an issue for the target applications, each node is aware of the node with the closest ID in the network and maintains an index towards it. Each time that a query for the starting ID range arrives to a node, it scans its keys and the locally stored metadata files and replies with the ones that answer the query. It is indicated that the indexed value is stored to a local database, so the node that receives a query can easily find the metadata files corresponding to the query with a simple local database lookup. If the node is responsible only for a subrange, then it forwards the query to the next node. The forwarding of the query continues until objects within the ID range are retrieved. Afterwards, the next ID range that answers the query and belongs to nodes that have not been already visited is calculated and the lookup procedure continues as it has been described.

It has been shown that the data placement and the indexing of the stored values can be used to build an advanced query engine for complex queries, while the computational cost and management overhead can be minimized by exploiting the inherent DHT mechanisms leading to scalable and fault-tolerant solutions.

CHAPTER 3

Adaptive Methods for Distributing and Searching Concept Hierarchies

Chapter 3 presents a system for processing bulk hierarchical data in a DHT-based overlay. Concept hierarchies greatly help in the organization and reuse of information and are widely used in a variety of information systems applications. The goal to achieve is to efficiently store and query data organized into concept hierarchies and dispersed over a DHT substrate. In addition to the fact that the DHT substrate can transparently handle node churn, replication, reliable distributed storage, etc, the proposed techniques offer adaptive indexing according to the granularity of the incoming queries and online updating with low cost and no downtime. The proposed mechanisms for re-indexing allow peers to decide individually on the level of indexing according to the granularity of the incoming queries. Roll-up and drill-down operations are performed on a per-node basis in order to minimize the required bandwidth for answering queries on variable aggregation levels. The methods used in the proposed system are applied to distribute the execution of a Grid Information System. In this case, a fully decentralized scheme is developed that creates, queries and updates large volumes of hierarchical data on-line and is considered as a replacement for the traditional centralized and strictly indexed information systems. Extensive experimental results support this argument that the proposed techniques proves very efficient in skewed workloads, both over single and multiple hierarchy levels at the same time. The resultant system adapts to sudden changes in popularity and effectively stores and updates large amounts of data at very low cost.

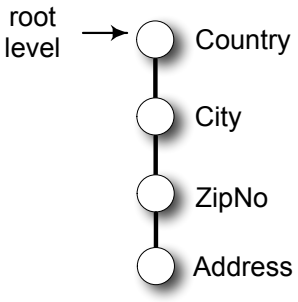


Figure 3.1: *A concept hierarchy for* Location.

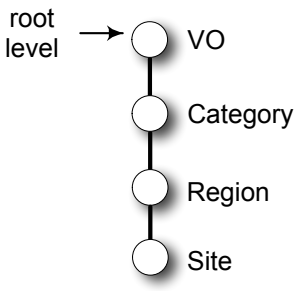


Figure 3.2: *A concept hierarchy with the VO as the root level.*

3.1 Introduction

The P2P networks apply as a distributed and resilient solution for the management of large volumes of data. For this reason, the adoption of P2P overlays is indicated for large-scale distributed systems that host large amounts of data providing similar functionalities to the databases systems.

In most applications, data are usually determined by multiple characteristics (or *dimensions*), which can be structured with different levels of granularity through the use of concept hierarchies. A concept hierarchy (or *taxonomy*) defines a sequence of mappings from more general to lower-level concepts. For example, Figure 3.1 depicts the ordering for the *Location* concept from the more general level, which is the `Country` towards the most detailed one, namely the `Address`. In various applications, concept hierarchies with partial ordering are met, like the concept hierarchy for *Time* shown in Figure 3.3. It is considered that the proposed system does not support this category of concept hierarchies. In case that the application implemented on top of the proposed system generates data following concept hierarchies with partial order, then a rearrangement of the levels is needed so as the levels to be fully ordered, as shown for the *Time* in Figure 3.4. The mappings of a concept hierarchy are usually provided by application or domain experts. The organization of information according to a sequence of mappings from more general to lower-level concepts is important for its search and reuse. The data can be searched at different levels of summarization, while aggregation is supported by the structure of data.

While there has been considerable work in sharing simple relational data using both structured and unstructured overlays (e.g., [HHB$^+$03, OTZ$^+$03]), no special consideration has been given to data supporting hierarchies. Assuming that a DHT-based overlay is used to build a large-scale, distributed system, the problem to tackle is the development of the required techniques and mechanisms to manage and process data with concept hierarchies.

The distinctiveness of inserting tuples containing values for each level of a concept hierarchy is that they need to be stored and indexed so as queries for any level of the hierarchy to be allowed.

Nevertheless, the DHT overlays support lookups only for the stored keys. The selection of the value of a level to be used as the key of the stored tuple would result in a system that answers effectively queries referring to the chosen attribute level, whereas queries concerning other levels of the hierarchy demand global processing. Nevertheless, a solution of multiple insertion of each tuple by hashing every hierarchy value is not viable: As the number of levels increase, so does the redundancy of data and the storage sacrificed for this purpose. While exact-match queries would be answered without global processing, this scheme fails to encapsulate the hierarchy relationships. Thus, a hybrid solution is followed: The values of a reference level are hashed and used as keys in the DHT overlay, while an adaptive indexing scheme is built for the rest of the levels.

Another significant feature supported by the introduced techniques is the adaptiveness of the resultant system implementing them to the query workload, in terms of adjusting its indexing structures to the incoming queries to achieve better performance and load balance. The adaptiveness of the system depends on the ability of deciding adequately in an on-line, distributed manner about re-indexing operations. The trends in the queries are monitored on a per node basis so as the most queried levels for all instances with the same value in the most general level to be found. The re-indexing of inserted tuples to the most popular levels increases the efficiency of the query resolution and contributes to more effective bandwidth consumption. The re-indexing is achieved with *roll-up* operations towards more general levels of the hierarchy and *drill-down* operations navigating to lower levels of the hierarchy with increased detail.

The above requirements have been taken into account and the proposed techniques can be enforced to a system that efficiently stores, queries and updates data organized in concept hierarchies. A DHT overlay is chosen as a reliable substrate over which data are stored, indexed and queried, thus eliminating all possible bottlenecks created by the hierarchical or centralized approaches. The data producers individually insert data items to the systems. Queries are still answered, while incremental updates are efficiently processed. The described solution takes the query granularities into account, adjusting the indexing structure to favor performance. Second, the proposed system preserves all hierarchy-specific information and data are stored in tree-like data structures and maintain indices to related keys. The insertion of data enables the resolution of more complex queries referring to correlations among different levels of the hierarchy such as: "*Which* `sites` *correspond to* `VO` *'Biomed' ?*" or "*What* `category` *does* `region` *'AsiaPacific' belong to ?*".

The application that has been studied and applied on top of the proposed system is a distributed and efficiently operational grid *information system*. Grid computing allows for coordinated resource sharing and problem solving in virtual organizations (*VOs*). A VO is a group of users from multiple institutions who collaborate to achieve a specific goal. The vision of grid computing is to provide a network of resources which, acting like a single supercomputer, offers

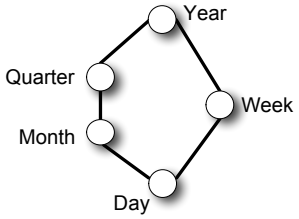


Figure 3.3: *A partially ordered hierarchy for* Time

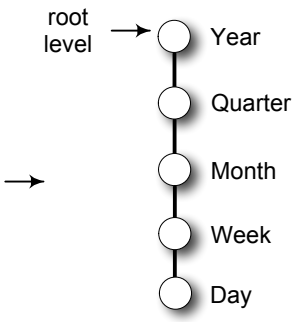


Figure 3.4: *A fully ordered hierarchy for* Time

resources that are easily accessible. In order for jobs to be adequately served by the most appropriate resources, the information system stores all needed information about the characteristics of the available resources over time.

There exist a number of systems that accomplish the tasks of an information system (e.g., [MDS], [RGM], etc). Nevertheless, they either feature a central information repository or a hierarchy of aggregation sites that introduce both scalability (single points of failure) and performance (processing burden on a single site) issues. Work in the area of distributed databases offers a variety of systems which can be used to disseminate and query this information. Nevertheless, these schemes cannot be used to maintain the semantics of the hierarchy and efficiently retrieve views of the data at different granularities. This is very important for applications such as a large scale information system, as queries naturally target different levels of detail: Historic queries usually require grouping by the highest hierarchy levels (e.g., group-by `VO` or group-by `Year`), whereas online queries are naturally directed towards more detailed levels (e.g., group-by `Site` or by `Day`).

In order to overcome these deficiencies, a fully distributed and adaptive architecture is proposed. Let us assume that the system's database contains a location dimension that relates to the `VO` location information (see Figure 3.2). Monitoring information is described by attributes (*facts*) such as the number of running jobs, number of waiting jobs, available storage space, total storage space, etc. Common accounting queries could be "*Give me the average CPU time*" or "*Give me the minimum available space in Gbytes*", presented to the user grouped by a VO value.

3.2 Notation

Let the data items stored in the system be in the form of *tuples* containing values for all levels ℓ_i of a concept hierarchy with L levels. They also contain a numerical fact of interest (e.g., `CPU Time, Available Memory`, etc) or the location of actual data. The uppermost level of the hierarchy

(ℓ_0) is called *root level* and its value *root key*. It is also defined for this work that $\ell_a < \ell_b$, where ($a, b \in [0, L-1]$), if and only if ℓ_a is higher in the concept hierarchy (namely closer to ℓ_0) than ℓ_b. The values of the hierarchy levels are organized in tree structures, one per root key. Without loss of generality, it is assumed that each value of ℓ_i has at most one parent in ℓ_{i-1}. During the insertion of a tuple, a level of its hierarchy is chosen and its hashed value serves as its *key* in the underlying DHT overlay. This level is referred as *pivot level* and to its value as *pivot key*. Finally, the highest and the lowest pivot levels of the hierarchy for a specific root key are called *MinPivotLevel* and *MaxPivotLevel* respectively.

3.3 Data Insertion

The key for each tuple is the result of a hash function applied on the value of the selected pivot level. Tuples are assigned to the node with ID numerically closest to the generated keys, according to the standard DHT operations.

In the proposed system, both initial insertions as well as incremental updates are handled in a unified manner. A completely distributed catalogue containing all the root keys and their corresponding pivot keys is introduced. Each root key is stored in the node responsible for it along with the list of pivot keys that have been already inserted. The root key is also aware of the MaxPivotLevel used during tuple insertion containing its value.

The procedure followed during the tuple insertion is as follows: The root key for this tuple is generated and a lookup for this key takes place in the DHT overlay. If the root key exists, the tuple ends up in the node responsible for it. An *insertion* is considered to be the procedure followed in case that the root key does not already exist in the overlay. Otherwise, the *update* procedure is followed (see Section 3.5.1).

In case of an insertion, the tuple or the group of tuples with the same root key arrive at the node responsible for it. The node selects a pivot level (either a random or a predefined one) and the pivot key(s) of the tuple(s) is (are) calculated for the appropriate pivot level. Each new pivot key is added to the list of pivot keys. Finally, each tuple is stored in the node with the ID closest to its pivot key.

Each peer organizes the tuples in trees that preserve their hierarchical nature. As a consequence, each distinct value of the pivot level corresponds to a tree that reveals part of the hierarchy. When a new tuple arrives at the node responsible for it, the node searches its keys. If no tuples for this pivot key have been stored, a new tree with a single branch is created. In the opposite case, a new branch is added below the value of the pivot level with the new values of the remaining levels.

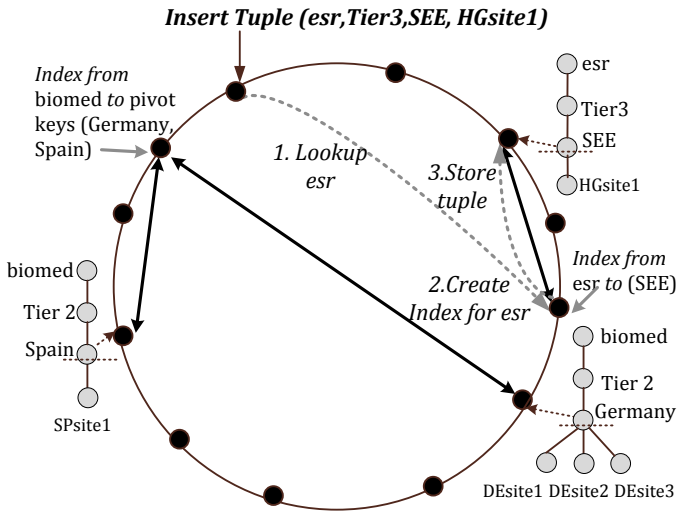


Figure 3.5: *Insertion of a new tuple with its root key not already stored in the overlay.*

An example of insertion is shown in Figure 3.5. A graphic convention is that solid lines represent existing indices, while dotted lines correspond to logical steps followed during the described procedure. Let us assume that tuples follow the `VO` hierarchy depicted in Figure 3.2 with ℓ_2 (`Region`) as the globally defined pivot level for initial insertions. A tuple with root key *'esr'* is inserted in the overlay. Since the specific root key does not already exist, a new *index* is created in the corresponding node. Afterwards, `Region` is selected as pivot level and the tuple is forwarded to the node responsible for this key, which creates a new tree with only one branch for this tuple.

3.4 Data Lookup and Soft-state Indices

Queries concerning the pivot level are exact match queries and can be answered by the DHT lookup operation. Queries for any other level cannot be resolved unless flooded across the DHT. Towards the exploitation of the knowledge acquired by flooded queries, *soft-state bidirectional indices* are introduced to the proposed scheme. When a node answers a flooded query, it checks whether a roll-up or drill-down is necessary. If this is not the case, the query initiator starts the procedure of creating an index, as soon as it receives the complete answer. It inserts the result of hashing the requested value in the DHT along with IDs of nodes having the actual tuples. The tuple holders also mark the specific value as *indexed*. The next time that a query for this key is initiated, the lookup operation locates the node holding the index, finds out the IDs of nodes with relevant tuples and retrieves them.

The created indices are soft-state, in order to minimize the redundant information. This means that they expire after a predefined period of time (Time-to-Live or *TTL*). Each time that an existing index is used, its TTL is renewed. This constraint ensures that changes in the system (e.g., data location, node departures, etc) will not result in stale indices, affecting the validity of the lookup mechanism. Moreover, after the number of indices has reached a limit I_{max}, the creation of a new index results in the deletion of the oldest one(s). Overall, the system tends to preserve the most "useful" indices, namely the ones directed towards the most frequently queried data items

The nodes with actual tuples of the indexed value need to know the existence of an index, in order to erase it after a re-indexing operation. The bidirectionality of the indices is introduced only to ensure data consistency, despite of them being soft-state. During re-indexing operations, the locations of stored tuples change and indices correlated to these tuples need either to be updated or erased, preventing the existence of stale indices. It has been chosen to erase them, so as to avoid increasing the complexity of the system. Detailed information for an existing index is not essential for the node, where the tuples are stored. A simple mark for each indexed value is adequate in order to erase its index, if it is needed. In this case, some redundant operations for erasing expired indices may occur. If there are no memory restrictions and local processing is preferable to bandwidth consumption, indexed values can be marked with a timestamp. Every lookup for an indexed value renews the TTL in both sides of the index and only valid indices are erased during re-indexing operations.

In the example of Figure 3.6, a query for *'SEE'* is resolved directly by the lookup operation of the DHT protocol. Lookups for values of the root level are processed utilizing the indices created during insertions. Nevertheless, any query for any level other than the pivot level or the root level ends up with no results. For example, a query for data items described by *'Tier2'* does not contact the two nodes with the corresponding trees before the creation of the index. The next step is the flooding of the query and the nodes storing the keys of *'Spain'* and of *'Germany'* are reached. The query initiator, which is now aware of the existing pivot keys, creates indices storing the information about these pivot keys to the node responsible for the value *'Tier2'*. This node now has an index pointing to the node *'Spain'* and another to node *'Germany'*. In the future, queries for *'Tier2'* will be answered without flooding, utilizing the created soft-state indices. As shown analytically in Figure 3.6, a query for the `Category` *'Tier2'*, after the creation of the indices, reaches the node responsible for this key, which in turn forwards the query to all relative nodes directly. The nodes storing the trees with the queried value return only the relevant tuple(s).

Since root indices are created during the insertion of tuples, an optimization for the flooding scheme can be applied to reduce the number of messages required for flooding, taking advantage of their existance. When a flooded query is forwarded from a node to its subsequent neighbor in the overlay, the covered range of IDs of the visited node (hence *CoveredIdRange*) is registered.

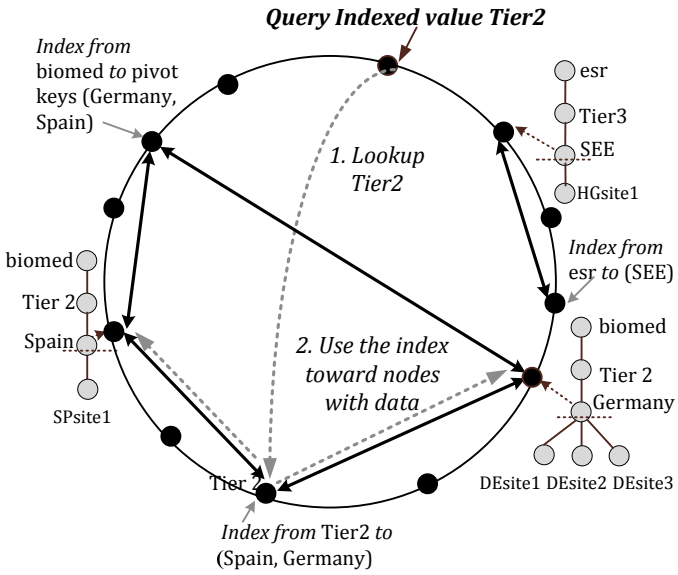


Figure 3.6: *Lookup using soft-state indices for a value belonging to a non-pivot level.*

According to the described approach of storing the data, if the query regards a value belonging to a level below the pivot level, the corresponding tuples can be retrieved by only one node. Therefore, the forwarding of the flooded query terminates, as soon as the node with the queried value is contacted. In case of a query for a value above its pivot level, the forwarding of the query cannot be terminated when the first contacted node responsible with a tree containing this value is found. Nevertheless, this node can answer to the initial node with its corresponding tuples and it can forward the query to the node responsible for the root key. Upon the delivery of the flooded query to the node with the root index, the node examines the CoveredIdRange of all visited nodes and sends the query in parallel only to the nodes storing the pivot keys, that are candidates to answer the flooded query and are not included in the CoveredIdRange. According to this strategy, the information stored in the root key is utilized and the visiting of all nodes may be avoided during flooding. For example, let us suppose data tuples organized as shown in Figure 3.5. A query for *'Tier2'* is flooded and thus it can be forwarded from a node to its subsequent neighbor clockwise. In this case, the node with the pivot key *'Germany'* is reached at first. The specific node forwards the query to the node responsible for the root index as implied by the described optimization. Eventually, the node with the root index sends the flooding query only to the node responsible for the non-visited pivot key *'Spain'*.

3.5 Re-indexing Operation

A major goal of the described system is to dynamically adapt on a per node basis to online queries, so as to increase the ratio of the non-flooded queries. In order to achieve this goal, two re-indexing operations regarding the selection of pivot level are introduced: *Roll-up* towards more general levels of the concept hierarchy and *drill-down* to levels lower than the pivot level.

The idea behind individual re-indexing of stored tuples is based on the fact that each node has a global view of all the queries directed towards a value above the pivot level, but only a view of the queries directed towards only to its values for the levels below the pivot level. Therefore, it has sufficient information to decide if a drill-down will favor the increase of the exact match queries for its values. On the other hand, a node has to cooperate with other peers that store a value of a level $\ell_i < pivotlevel$ in order to decide if this level is more appropriate.

The re-indexing of the data tuples (through a choice of a different pivot level) is performed on a per-tree basis, requiring no global coordination. Each node collects information using the incoming queries and finds out if the pivot level of a tree remains its most popular level. Otherwise, the node proceeds with the re-indexing of the tuples of this tree. The popularity of the levels of a tree is estimated based on their average rates of incoming queries (hence InQ) over a time period. A node maintains one record per tree with these rates during a restricted time-frame W. This parameter should be properly selected to perceive variations of query distributions and, at the same time, stay immune to instant surges in load.

In more detail, the mechanism works as follows: A node may check if a re-indexing is required based on the objective to achieve. The implemented strategy implies that a node decides whether a roll-up or drill-down is required when it answers a flooded query or when a number of queries for indexed values have been received. While the main objective is the increase of the queries answered without flooding, this strategy targets to the increase of exact match queries as well.

The number of queries for indexed values triggering a node to examine a possible re-indexing may vary and has an impact to the adaptiveness of the system. A small value indicates that the potential of re-indexing is examined more often and thus more re-indexing operations may take place. Nevertheless, if a decision has erroneously been made, it can be easily corrected. However, during re-indexing operations, existing indices are deleted and this may have a negative impact on the system. In the opposite case, the system tends to depend more on the effectiveness of the indices. It has been observed that re-indexing operations are necessary, when popular values belong to levels with several distinct values. Indices perform better for higher levels with less values, since there is a high probability for repeated utilization of an index.

A node decides if a re-indexing operation will favor the increase of non-flooded queries based on the '*popularity*' of each level following the procedure described in basic steps in Algorithm 1.

Algorithm 1 Decision Algorithm in the node answering a query

pivotlevel: current pivot level
ℓ_q: the queried level of the flooded or indexed value
NotPivotKey: the flooded or indexed value
InQ_{tot}: rate of incoming queries for the tree with *NotPivotKey*
InQ_{ini}: initial minimum rate to allow re-index operations
$InQ_{l_{pop}}$: rate of incoming queries for the most popular level
action: the decided action
$\ell_{pop} \Leftarrow FindMostPopularLevel$
if $\ell_q > pivotlevel$ **then**
  if $(InQ_{tot} > InQ_{ini})$ AND $(\ell_{pop} > pivotlevel)$ AND $(InQ_{\ell_{pop}} > thr \times InQ_{tot})$ **then**
    Drill-down to l_{pop}
    $action \Leftarrow NoAction$
  else if *NotPivotKey* is NOT indexed **then**
    $action \Leftarrow CreateIndex$
  else
    $action \Leftarrow NoAction$
  end if
else if $\ell_q < pivotlevel$ **then**
  if $(InQ_{tot} > InQ_{ini})$ AND $(\ell_q = \ell_{pop})$ AND
  $(InQ_{\ell_q} > thr \times InQ_{tot})$ **then**
    $action \Leftarrow PositiveToRollup$
  else if *NotPivotKey* is NOT indexed **then**
    $action \Leftarrow CreateIndex$
  else
    $action \Leftarrow NoAction$
  end if
end if

A *thr* parameter is used to indicate if a re-indexing operation is required. The following criterion defines if a re-indexing to a level ℓ_q is allowed:

$$InQ_{\ell_q} > thr \times \sum_{i=0}^{i=L-1} InQ_{\ell_i}, \ell_q \neq pivotlevel, thr \in [0, 1]$$

In the described algorithm, two possible cases are considered to indicate the necessity of a re-indexing operation: **The queried level ℓ_q lies lower in the hierarchy than the pivot level of the tree ($\ell_q > pivotlevel$).** Only one tree stores the values of a level below the pivot level. Therefore, the specific node is aware of the exact popularity of these values and feels 'confident' to decide if a drill-down is needed. If the most popular level ℓ_{pop} of the tree lies below the pivot level and the defined criterion is valid for its InQ, then a drill-down to this level is decided. After the decision for drill-down is made, the node finds all the distinct values of the new pivot level

Algorithm 2 Decision Algorithm in the querying node

l_q: the queried level of the flooded or indexed value
NotPivotKey: the flooded or indexed value
action: the required action by involved nodes {*action* = *PositiveToRollup* if at least one node is possitive to roll-up}
if *action* = *PositiveToRollup* **then**
  Gather statistic information
  Calculate *InQ* for each level
  $\ell_{pop} \Leftarrow FindMostPopularLevel$
  $MaxPivotLevel \Leftarrow FindMaxPivotLevel$
  if $(\ell_q = \ell_{pop})$ AND $(InQ_{\ell_{pop}} > thr \times InQ_{tot})$ **then**
    Roll-up to l_{pop}
  else if $(\ell_{pop} \geq MaxPivotLevel)$ AND $(InQ_{\ell_{pop}} > thr \times InQ_{tot})$ **then**
    Group-Drill-down to ℓ_{pop}
  else if *NotPivotKey* is NOT indexed **then**
    Create Index for *NotPivotKey*
  end if
else if *action* = *CreateIndex* **then**
  Create Index for *NotPivotKey*
end if

and hashes them one by one, sending the new groups of tuples to the corresponding nodes. The already gathered statistic information is sent along with one randomly selected group, in order to maintain information about the query distribution for the values contained in the drilled-down tree within W. Any existing indices for any value of this tree are removed. If a drill-down is not needed, the node includes in its answer to the initiator the fact that the queried level is lower than the pivot level, hence it can carry on with the creation of the soft-state index and expedite the process.

The queried level ℓ_q lies higher that the pivot level of the tree ($\ell_q < pivotlevel$). In this case, there are more than one trees with this value needed to participate in a possible roll-up to this level. Otherwise, lookups for this value will not return complete results. If the threshold criterion is satisfied for the ℓ_q, then the node is positive to the potential of adopting this level as pivot level for this tree. This step is indicative of an imbalance and the query initiator is informed about this. The query initiator decides for a re-indexing operation according to the procedure described in Algorithm 2. If the query initiator is aware of at least one node willing to roll-up to this level, it starts a procedure to confirm the local intuition by using statistic information provided by all the nodes having answered the query. After receiving the tuples containing the number of InQ per level, it calculates the total value of InQ per level.

The calculation of the total InQ per level is not straightforward. Queries concerning an ℓ_i for any $\ell_i \geq pivotlevel$ end up only in one node and are thus counted once for statistic purposes.

The same property is not valid for queries requiring values of higher levels than the pivot level. These queries reach more than one node and are counted in all of them. During the gathering of statistic information for roll-up decisions, the problem of multiple counting of such queries in the calculation of the InQ for each level needs to be solved. The complexity in the calculation of the overall InQ increases since more than one pivot levels may exist for the involved trees in the re-indexing procedure. For example, let us assume that the state of trees with the *'biomed'* as their root key is the one shown in Figure 3.7(b). In this case, the value of InQ for *'Tier2'* is sent twice by the nodes with the trees of *'Tier2'*. To avoid this situation, a path containing the values for all levels in $[0, pivotlevel]$ is sent along the statistic information to the querying node, so as to make the correct decision. Through this procedure, the querying node is also informed for the MinPivotLevel and MaxPivotLevel of all existing trees containing the queried value (hence *NotPivotKey*).

If the InQ of ℓ_q is more than *thr* of the total number of InQ, then the initiator messages the involved nodes to roll-up the corresponding trees to this level by re-inserting their tuples. If the re-indexing criterion for l_q is not fulfilled and since statistic information has been collected, the querying node examines if a drill-down to a level $\ell_i \geq MaxPivotLevel$ (the equality is for the case that all the involved trees do not have the same pivot level but some of them are already in the MaxPivotLevel) is dictated by the collected statistics. The intention is to take advantage of the fact that the querying node has now a more global view of the InQ per level. It is possible to find a level $\ell_i \geq MaxPivotLevel$ to be the most popular but this tendency not to appear in the partial views of the involved nodes. In this case, the query initiator informs the involved nodes that a drill-down to this level is needed. This procedure is called `Group-Drill-down`, since more than one nodes participate in the drill-down. All the trees with the queried value in ℓ_q drill-down to the new pivot level. If the new pivot level is equal to the MaxPivotLevel, the trees already in the MaxPivotLevel do not perform any action. If a re-indexing operation is not needed, no action is taken other than the creation of a soft-state index for this value.

Lock mechanisms are activated during the time that a re-indexing decision is being made. The purpose of this locking is to avoid examining the same re-indexing possibility multiple times for concurrent lookups on specific trees. Locks are revoked after the completion of an ongoing procedure or after a short period of time. The steps described in Algorithms 1 and 2 are performed, only if the corresponding locks are inactive. Otherwise the described procedures are not performed and only the query is answered.

Examples of the described re-indexing operations are applied in the trees of Figure 3.7 with root value *'biomed'*. The two trees of Figure3.7(a) are stored in different nodes of the DHT overlay and are considered the initial state before any re-indexing operation. It is supposed that a query for *'Spain'* triggers a drill-down operation. The result is shown in Figure 3.7(b). On the other hand, a query for *'biomed'* may result in a roll-up to the root level depicted in Figure 3.7(c) or

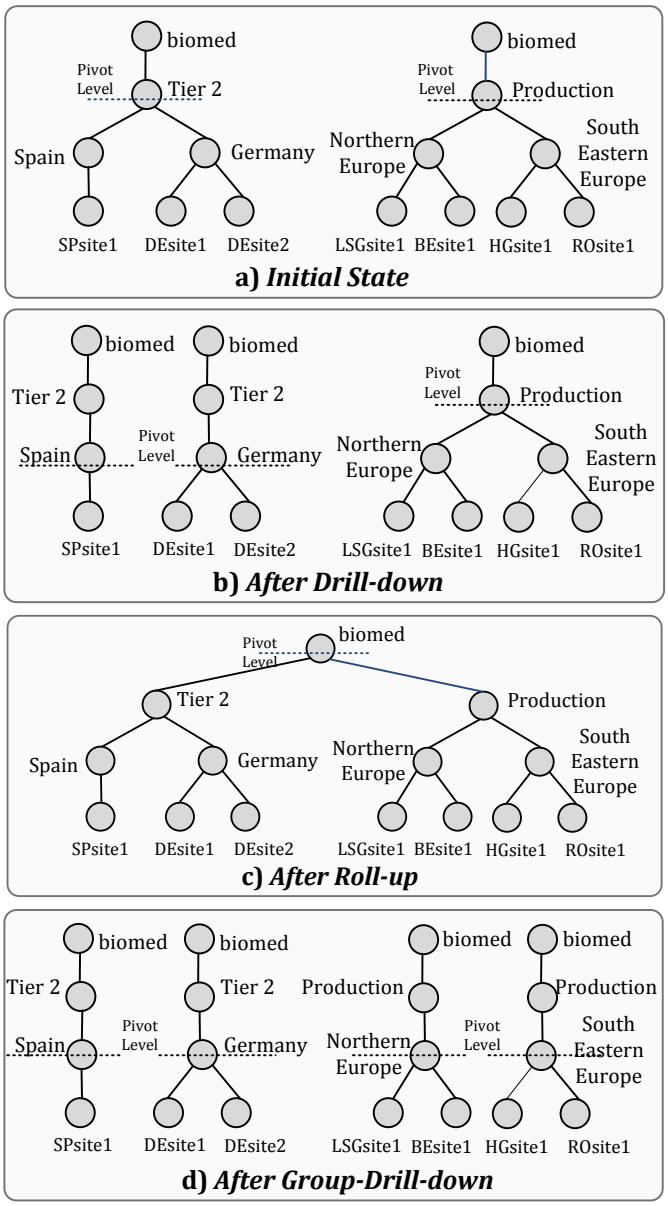


Figure 3.7: *Examples of drill-down and roll-up operations*

a Group-Drill-down to the `Region` level depending on the total InQ per level. The Group-Drill-down results to the trees of Figure 3.7(d) and differs from the simple drill-down, since all the trees drill-down to ℓ_2.

3.5.1 Updates

An *update* is the procedure followed during the insertion of a tuple, when its root key already exists in the overlay. The update procedure comprises of two consecutive phases: the insertion of the tuple and the updating of any existing indices for the values of the tuple. The insertion phase presents minor differences compared to the insert procedure.

The updates of the existing datasets is a more complicated procedure. During the insertion of new tuples, it is critical to select the correct pivot level so as to ensure the correctness of the lookup operations. The selection of the pivot level is not simple, since the pivot levels of the stored trees of a specific root key may vary due to performed re-indexing operations.

The following assumptions are made:

- If a new tuple contains a pivot key, then this key should be used during insertion. Otherwise, lookups for this value will return only the tuples, that existed before this operation.

- Even if none of the values belonging to the specific tuple have been used as pivot keys, they may have already been stored in the network. The selection of such a value as pivot key would result in the discovery of the new tuple only in a later search. Therefore, it is considered that the pivot level be equal to the MaxPivotLevel in this case. Re-indexing operations would take over to find the most appropriate pivot level of this tuple.

The difficulty of updates increases because the information about the stored pivot values and the MaxPivotLevel is distributed over the overlay. A distributed catalogue has been implemented by creating an index among the node responsible for the root key and the nodes with the pivot keys, namely a record with the root key and the corresponding pivot keys. This enhancement allows online updates with the system continuing to efficiently serve requests of the users.

During a new tuple insertion, a lookup operation for the root key is performed. The responsible node is contacted and it finds out if any value of the tuple corresponds to a pivot key. In this case, the tuple is stored to the responsible node for the pivot key and its new values below the pivot level are added as a new branch to the existing tree. In the opposite case, the hashed value of the MaxPivotLevel is considered as the pivot key of this tuple during its insertion in the overlay. The existence of trees with equal values above the pivot level is not excluded by this assumption and neither is the existence of corresponding indices. These indices should be updated so as indexed lookups to return complete answers. The node storing the tuple initiates lookups

for each ℓ_i which lies among the the root level and the pivot level ($0 < \ell_i < pivotlevel$) and the corresponding indices are informed about the new tuple.

Examples for the possible cases during an update are depicted in Figure 3.8. The node holding the index for the root key *'biomed'* concludes that the none value of the tuple *'(biomed, Tier2, Sweden, SWsite1)'* corresponds to an existing pivot key. Moreover, it is aware that the MaxPivotLevel for its trees is the `Region` and thus the tuple is inserted with *'Sweden'* as its pivot key. The new pivot key is also added in the list of pivot keys for this root key. However, the value *'Tier2'* is already indexed. During lookup for the value *'Tier2'* according the update procedure, the responsible node is discovered and a new index among this node and the node with actual data is created. During the update for tuple *'(biomed,Tier2,Spain,SPsite2)'*, the node with *'biomed'* index proceeds in the insertion of tuple with *'Spain'* as pivot key, resulting to the creation of a new branch in the existing tree. The indexed value *'Tier2'* is not affected and no further action is taken.

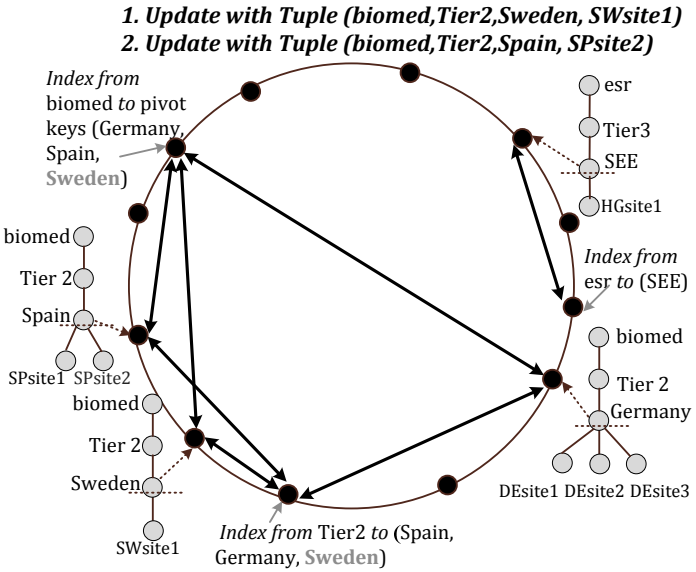


Figure 3.8: *Updates for an already existing and a non-existing pivot key*

3.6 Discussion

In this section, some aspects of the system related to its parameters as well as optimization issues are discussed.

3.6.1 Memory Requirements

The architecture of the proposed system requires the storage of additional information for the implementation of the proposed algorithms. In more detail, each node in the overlay may, relative to the hashing function as well as the query workload, store the following:

- Data: The inserted data are considered as tuples containing a value for each level of the hierarchy and the corresponding described fact value(s). When a tuple arrives to the node responsible for its pivot key, it is inserted in a tree-structure containing the specific pivot key, so as to avoid the storage of duplicate values for levels above the *pivotlevel* and facilitate the search procedures during lookups. For each stored tree, at least one tuple with L numerical values for statistic purposes is maintained, one per level of the hierarchy.

- Root Indices: Each time that a different root key is inserted in the overlay, indices are created towards its pivot keys. Moreover, when a new key is inserted for an existing root key, the corresponding index is added.

- Soft-state indices: The flooded queries result in the creation of soft-state indices, which are utilized for faster resolution of future queries. The maximum number of allowed indices is defined for each node through the I_{max} parameter. The required space for soft-state indices is $O(I_{max})$.

3.6.2 Parameter Selection

The introduction of various parameters in the proposed system influences its performance relative to their values. The threshold value (*thr*) plays an important role to the number of the performed reindexing operations. On one hand, a large threshold allows less reindexing operations and therefore increases the utilization of the soft-state indices. This case favors query workloads where the upper levels of the hierarchy are at most queried while the number of different values is limited. Thus, the probability of duplicate queries being issued is larger. On the other hand, a smaller *thr* value results in more frequent reindexing operations depicting the changes in the query trends. In general, the drawback of more roll-up and drill-down operations is the bandwidth consumption for the movement of the involved tuples and the invalidation of the existing soft-state indices. However, these operations are highly effective, when the values of the lower levels of the hierarchy are more popular. In this case, the contribution of indices to the performance of the system is not adequate. Therefore, lower values of the *thr* parameter are required for query workloads directed mainly towards lower levels of the hierarchy and targeting numerous different values.

Soft-state indices help the system improve its performance. Nevertheless, the maintenance of index consistency may be problematic in a dynamic system that adapts according to user preferences. For this reason, the TTL parameter is introduced: Indices which have not been used for a period larger than TTL, are not considered as useful and removed. This constraint ensures that changes in the system (e.g., data location, node departures, etc) will not result in stale indices, affecting the validity of the lookup mechanism. The amount of indices can be also calibrated according to the system capabilities. While memory becomes a cheaper commodity by the day, the plain size of data discourages an "infinite" memory allocation for indices. Therefore, after the number of indices has reached a limit I_{max}, the creation of a new index results in the deletion of the oldest one. If an indexed value occupies more than one index, then all of them are erased for consistency reasons. Calibrating I_{max} for performance without increasing it uncontrollably entails knowledge of the stored data (e.g., how skewed each hierarchy is). Overall, the system tends to preserve the most useful" indices, namely the ones directed towards the most frequently queried data items.

The window parameter W represents the number of previous statistics maintained by each node for the calculation of average rates of incoming queries. A large value of W will fail to perceive load variations, whereas a very small will have a negative impact, since the average rate of incoming queries will not be adequate to lead to re-indexing decisions or may lead to erroneous ones. In order to estimate its value, $W = O(1/\lambda)$ is set, i.e., the size of the window is connected with the query interval. The more frequent the requests, the smaller W can be and vice-versa.

3.6.3 Consistency

Another aspect to be considered is the various strategies to follow, so as to ensure data consistency during the various operations. In a highly dynamic environment, as the one described, where updates and re-indexing operations occur online, special care should be given to ensure the consistency of data during these operations, against some additional communication and maintenance cost. A remarkable point regarding consistency issues is the validity of root indices for the correct execution of updates. The root keys should be constantly aware of all the existing pivot keys and confirm their existence after a period. Therefore, it is advisable that root keys issue periodical messages to verify the validity of the information about their pivot keys.

Moreover, it is also important to ensure the participation of all relative trees in the roll-up and drill-down operations and the effectiveness of the locking mechanisms during re-indexing, so as consequent lookups on the re-indexed data to return complete results. During these operations, it can be observed that lookups fail to return complete results due to the movement of data. In order to face this miss, the data are cached at the initial nodes as well, until the re-indexing procedure is completed. Once the tuples are re-inserted and the root key completes the updates

of its indices towards the new pivot keys, the old trees are removed from their predecessor nodes after the expiration of a TTL. Lookups for values being re-indexed are also delayed to a node that receives the re-indexed data until the completion of the transaction. The root key aware of the ongoing operations on its data, is also responsible to forward a non-exact match query either to the nodes initially responsible for the data or the newly-responsible ones. The creation of soft-state indices for any value included in the involved trees is not allowed during the re-indexing period.

Finally, special care is taken during the creation of the soft-state indices to avoid stale indices. Apart from the TTL restriction, indices encountering any problem during their creation are removed.

3.7 Experimental Results

3.7.1 Simulation Setup

A comprehensive simulation-based evaluation of the proposed scheme is now presented. The performance results are based on a heavily modified version of the FreePastry simulator [Fre], although any DHT implementation could be used as a substrate. By default, a network size of 256 nodes is assumed, all of which are randomly chosen to insert tuples and to initiate queries.

In the most part of the simulations, synthetically generated data are used. The inserted data is a tree with each value having at most one parent. Each distinct value of ℓ_i has a constant number of children in ℓ_{i+1}. By default, the data comprise of 100k tuples, organized in a 4-level hierarchy (see Figure 3.2) with one numerical fact (e.g., `CPU_time`). The number of distinct values per level are $|\ell_0 = 100|, |\ell_1 = 1000|, |\ell_2 = 10000|$ and $|\ell_3 = 100000|$. The level of insertion is, by default, ℓ_1, unless stated otherwise.

For the query workloads used in the experiments, a two-stage approach is considered:at first the level that the query will target according to the *levelDist* distribution is identified; the requested value is then chosen from that level following the *valueDist* distribution. In the following experiments, the Zipfian ($p_i \sim 1/i^{\theta}$) distribution for *levelDist* is used, while a bias is expressed inside each level using the uniform, 80/20, 90/10 and 99/1 distributions for *valueDist*.

Generated queries are issued at an average rate of $1\frac{query}{time\_unit}$, in almost 50k time units total simulation time. The results for queries on a single dimension with multiple levels of hierarchy are presented. The default *thr* value is set to 0.3, which is a large enough value to avoid very frequent re-indexing attempts. Simulations with different values of *thr* around this default show small qualitative difference. The default value of W, which controls how quickly the system can adapt to changes, is set to 1000 time units. Finally, a practically infinite value of TTL (indices never expire) is assumed.

In this section, it is mainly intended to demonstrate the performance and adaptability of the proposed system under various conditions. The goal is to show that the system proves highly efficient under a variety of data and load distributions and can quickly adapt to sudden changes in skew without any modification to the default parameters. Specifically, the percentage of queries which are answered without flooding (*precision*) is mainly measured.

3.7.2 Performance Under Different Levels of Skew

In the first set of experiments the behavior of the proposed system is identified under a variety of query loads. Specifically, the number of queries directed to each level are varied by increasing the θ parameter in the *levelDist* distribution. For each value of θ, values inside each level using four different distributions are chosen.

In Figure 3.9, queries are skewed towards ℓ_0. As θ increases for *levelDist*, the performance of the described method improves: Re-indexing is performed sooner as more queries take place and the exact matches due to the chosen pivot level increase. By increasing the skew for *valueDist*, remarkably high precision rates (close to 100%) are observed, because both the ratio of popular queries and the density of queries for certain tuples increase. Another point that plays a big role is the limited number of distinct values of ℓ_0. Obviously this is quite small compared to the last level, thus enabling soft-indexing and faster re-indexing. For a set θ value, the method performs justifiably better as the distribution becomes more skewed: More queries exist for fewer distinct values. Finally, it is noticed that the larger the θ value, the smaller the difference in precision among the different *valueDist* distributions.

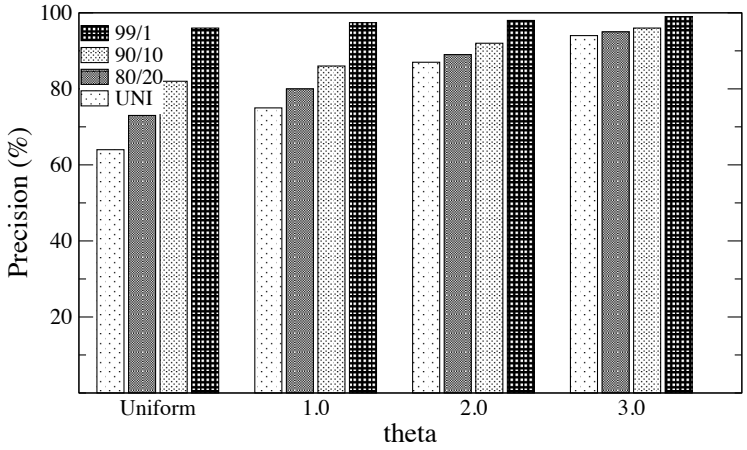


Figure 3.9: *Precision when skew directed towards ℓ_0*

Figure 3.10 shows results where the query load favors ℓ_3. Again, it is noticed a similar trend in performance as *valueDist* becomes more biased and the proposed method shows high precision

values, albeit reduced compared to the previous case. It is shown that the precision for the same *valueDist* dis tribution decreases as θ increases: This is due to the fact that ℓ_3 has a considerably larger number of values. By increasing the number of queries for those values, it is increased (relative to the choice of *valueDist* of course) the probability of queries to non-indexed values. Nevertheless, the decrease is smaller as *valueDist* becomes more skewed.

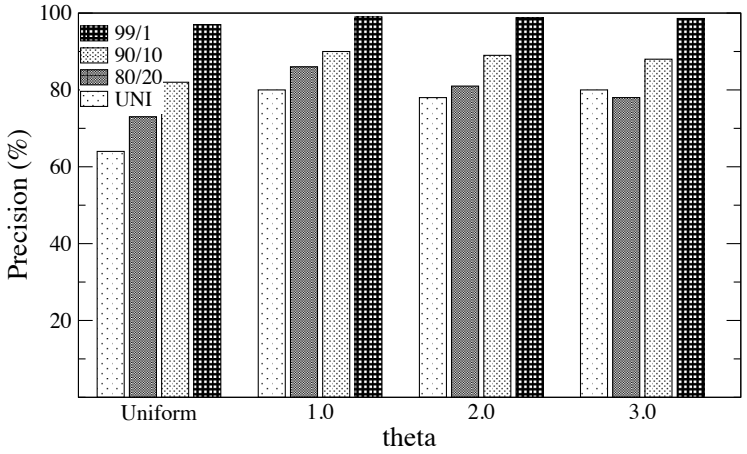


Figure 3.10: *Precision when skew directed towards ℓ_3*

3.7.3 Testing against Multiple Bias Points

In the next experiment, the proposed method is tested against a more challenging type of workload (*MULTI*): While different levels receive an equal number of queries, nevertheless a *different* part of a data tree from each level is targeted. Specifically, all levels are divided in quarters and target (using different values of *valueDist*) one quarter per level so that no quarter is related with any other in the data tree. This is a very challenging workload, as it forces the method to store different data at different levels of granularity. Table 3.1 summarizes the results, where besides the precision, the cost in number of re-indexed tuples as well as the number of total roll-up and drill-down operations are documented.

The proposed technique proves extremely efficient in all four workloads, achieving very high precision (between 92% and 100%) at low cost: The largest number of operations occur when it is uniformly queried the different values in which case 75% of the tuples are re-hashed and re-inserted from re-indexing operations. As the *valueDist* becomes more biased, the number of re-indexing calls decreases. This clearly demonstrates that the proposed method adjusts its operation according to the need: The number of trees being re-indexed is proportional to the number of unique trees that are popular. This is a highly desirable property since for most applications both dynamic and highly skewed loads are anticipated.

Table 3.1: *Performance comparison for the MULTI workload over different values of valueDist*

valueDist	**precision (%)**	**#roll-ups**	**#rolled-up trees**	**#drill-downs**	**re-inserts (%)**
uniform	92.0	25	250	500	75
80/20	94.3	25	250	171	42
90/10	95.2	25	250	51	30
99/1	99.5	1	10	6	1.6

3.7.4 Performance in Dynamic Environments

The adaptiveness and performance of the proposed system in a dynamic environment is examined by this set of experiments. The query distribution encloses a sudden change in skewness from level ℓ_0 towards ℓ_3 and vice versa in the middle of the simulated queries.

Figure 3.11 demonstrates the behavior of the system when the query load shifts from ℓ_0 towards ℓ_3. The results show that, in all cases, the system increases its precision due to the combination of re-indexing operations and soft-state indices and the majority of questions are answered by exact match lookups. The precision reaches over 90% for $\theta = 2.0$ and over 80% for $\theta = 1.0$ before the change in skew. In the transitional stage, the flooding of the queries increases but the system rapidly manages to recover and regain its performance characteristics (after at most 5% of the queries). The steep decrease in precision happens at the exact time of the shift in the workload: A much larger number of distinct values belong to ℓ_3, thus the existence of useful indices is less probable. The contribution of soft-state indices is not sufficient to handle the query load until drill-down operations take place. In this stage, the larger the value of θ, the larger the decrease in precision and the faster the recovery: As it is shown in Figure 3.12, where the query loads for *valueDist* 90/10 are considered, both exact match and indexed lookups are fewer for $\theta = 2.0$. This happens because queries are more skewed towards ℓ_3 and benefit even less from the already rolled-up trees. However, as θ increases, drill-down decisions are taken faster, favoring the increase of the exact match queries that answer the majority of the requests.

The precision of the algorithm is tested against a sudden shift from ℓ_3 to ℓ_0 for various workloads and displayed in Figure 3.13. During the steady stages of the simulation, similar trends to the ones of one directional skew are observed and the system presents high performance over 80% for all workloads. As the *valueDist* becomes more biased, higher precision is accomplished, since the number of popular values shrinks and drill-down operations are performed faster increasing the adaptiveness of the system. After the change in the direction of skew, less queries are flooded for the $\theta = 2.0$ workloads (behavior that contrasts to the previous experiment). Figure 3.14 demonstrates a more comprehensive view of the system after the change in skew. Indices

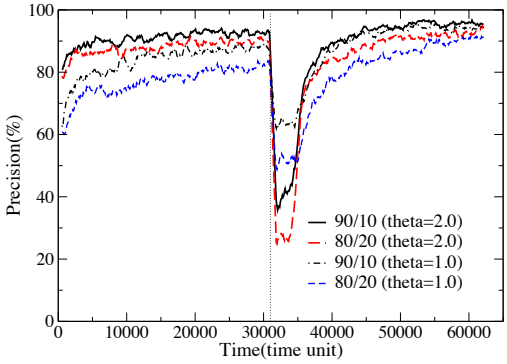


Figure 3.11: *Precision over time for various workloads, when skewness changes from ℓ_0 to ℓ_3*

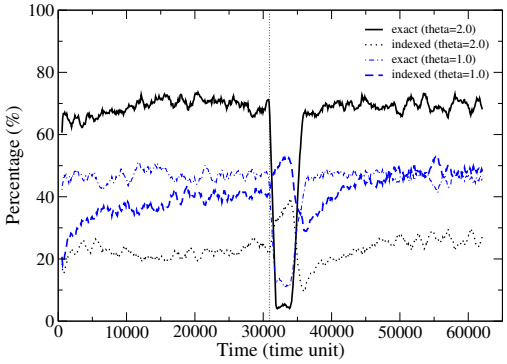


Figure 3.12: *Precision (split in exact and indexed queries) over time for valueDist 90/10, when skew is directed from ℓ_0 to ℓ_3*

take over to serve lookups immediately. Due to the smaller number of distinct values in higher levels, indices perform well. However, the consecutive roll-ups destroy the existing indices and the performance of the system is influenced negatively. The system regains its performance by the rapid increase in the exact lookups.

The comparison of results among the two shifts of the workload reveals that the soft-state indices are capable to preserve the high precision of the system in case of a skew towards higher levels due to the limited number of different values. On the contrary, the adaptiveness of the system significantly depends on the re-indexing operations, when lower levels of the hierarchy are the most popular. Nevertheless, in both cases, the system needs bounded time to reorganize its indexing mechanism and achieve high performance.

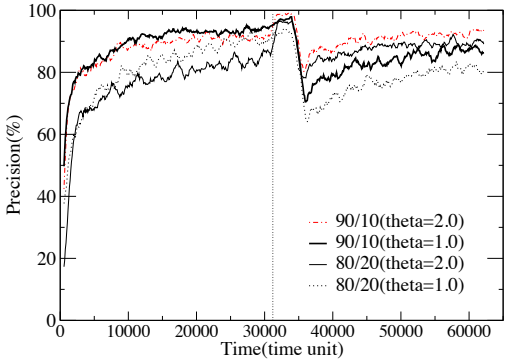


Figure 3.13: *Precision over time for various workloads when the skewness of workload changes from ℓ_3 to ℓ_0*

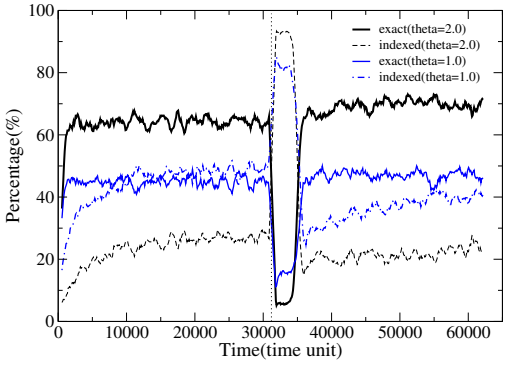


Figure 3.14: *Precision (split in exact and indexed queries) over time for valueDist 90/10, when skew is directed from ℓ_3 to ℓ_0*

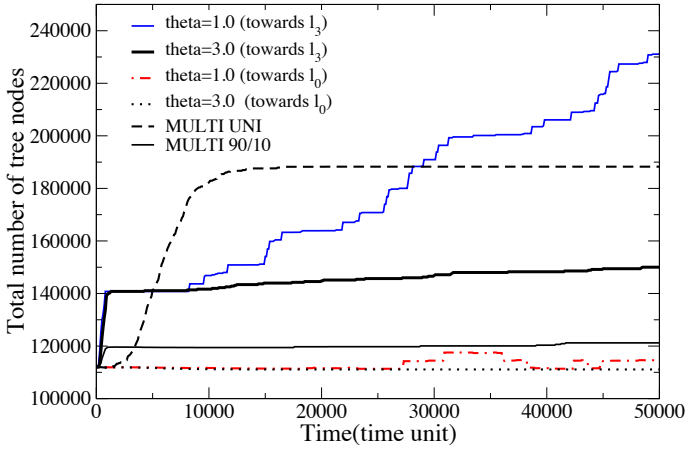


Figure 3.15: *Total number of tree nodes over time for workloads skewed towards l_0 and l_3 and* valueDist *90/10 and* MULTI *workloads with uniform and 90/10* valueDist

3.7.5 Storage Load for Different Number of Nodes

The inserted tuples in the system are stored in tree structures with the same pivot key to avoid the storage of redundant information for values above the pivot level. However, these values may exist in more than one trees stored in different nodes of the overlay depending on the selected pivot level. The total number of tree nodes receives its minimum value, when the root level is chosen as *pivotlevel* and the maximum number, when the lowest level of the hierarchy is the *pivotlevel* of all trees (≈111K and 400K nodes respectively for the inserted dataset). Each node of the tree (hence *tree node*) represents the value of a tuple for this level. The total number of tree nodes during the simulation time is presented in Figure3.15. The increase in the number of tree nodes indicates that drill-down operations take place, while a decrease depicts the effects of roll-up operations. Thus, the total number of tree nodes over time demonstrates the evolution of executed re-indexing operations. Queries directed towards ℓ_3 for $\theta = 1.0$ lead the system to reinsert its tuples slowly to lower levels of the hierarchy, which tend to be the most popular. Since the difference in the popularity amongst the levels is not excessive, the drill-down operations happen during the whole time of the simulation. In case of the respective workload for $\theta = 3.0$, the increase of tree nodes is steeper and it is completed during a short period at the initial stages of the simulation. The most popular trees re-insert their tuples to lower levels quickly and no major variations are noticed during the rest of the simulation time. The workloads directed towards ℓ_0 cause no significant variations in the total number of tree nodes. In both cases of *levelDist* for these workloads, the number of tree nodes is preserved close to its initial value, since the total tree nodes with ℓ_1 as pivot level does not differ notably compared to the tree nodes with ℓ_0 as pivot level. Finally, a comment about the *MULTI* workloads follows, where roll-up and

drill-down operations coexist. When the queries target the values uniformly inside the levels, the re-insertion of tuples continues for a longer period and for a larger number of tuples than in the respective workload with *valueDist* 90/10. The popularity of levels changes for different group of trees in the *MULTI* workloads and thus the insertion of tuples to the most appropriate level occurs in a more concurrent way than in the workloads skewed towards ℓ_3 for $\theta = 1.0$.

The total number of soft-state indices in the system appears in Figure 3.16. The creation of soft-state indices is influenced by the evolution of re-indexing operations. According to the described strategy, a soft-state index follows the flooding of a query as a supplemental solution to the adaptiveness of the pivot level. Therefore, the steep inclination in the curves of the workloads directed towards ℓ_3 is justified considering the fact that the number of different values in the lower levels is larger than in the upper levels. In the workloads with higher levels being more popular, the increase of indices occurs at a lower rate. The efficiency of the re-indexing operations becomes more evident in the case of the *MULTI* workload following a uniform distribution inside the level. New indices are temporarily created, which are removed as soon as roll-up and drill-down operations take over and the system adapts to the trends of the incoming queries.

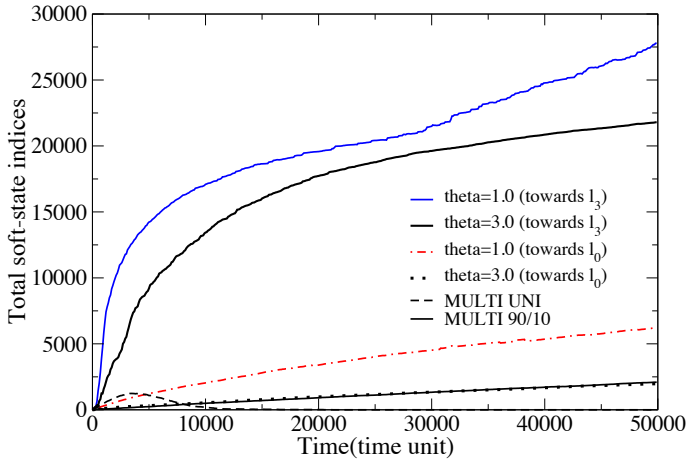


Figure 3.16: *Total number of soft-state indices over time for workloads skewed towards* ℓ_0 *and* ℓ_3 *and* valueDist *90/10 and* MULTI *workloads with uniform and 90/10* valueDist

The different objects stored in the nodes of the overlay can be divided in the following categories: the tree structures storing the values of the selected hierarchy, the root indices which are created during the insertion of a new root value and are aware of their pivot keys, the soft-state indices and the statistical information maintained for the re-indexing decisions. The experimental results of Figures 3.17-3.19 demonstrate the average number of these objects per node or the items that determine their volume. These values are the average results during the whole simulation time, derived by the measurements in regular periods. Thus, they incorporate the various

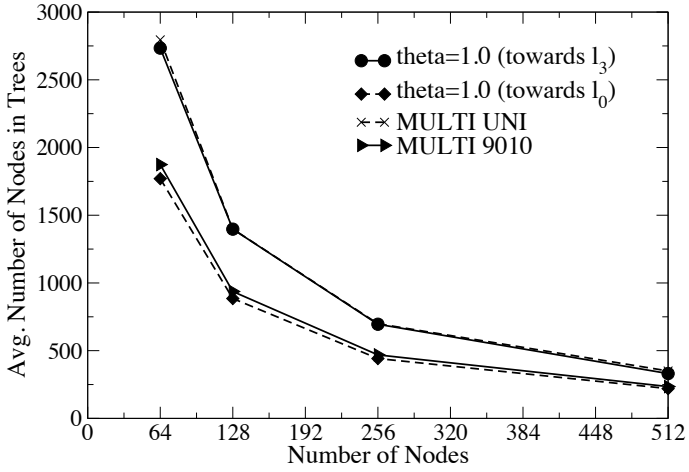


Figure 3.17: *Average number of tree nodes per node*

changes caused by the roll-up and drill-down operations. The *MULTI* workload with uniform distribution and the skewed workload towards ℓ_3 with $\theta = 1.0$ have the maximum number of tree nodes, which conforms to the results of Figure 3.15, where the curves of these workloads reach the highest values. It should be noticed that the presented values for the *MULTI* workload is the maximum observed values for this type of workload, since the uniform *valueDist* is studied. Therefore, all the trees of the dataset re-insert their tuples to the appropriate level. If the distribution of queries alters to 90/10, then the average number of tree nodes decreases significantly. The average number of tree nodes for the workload skewed towards ℓ_0 remains low. The average number of pivot keys (see Figure 3.18) behaves in analogous manner to the tree nodes for all the workloads. The number of pivot keys is correlated to the stored statistic information and the number of root indices. For each pivot key, a tuple with L levels is maintained. Moreover, if the average rate of incoming queries is calculated with the method of a sliding window with n slots, the number of maintained tuples should be multiplied with this factor. The number of existing keys also equals to the number of root indices, since a separate index is maintained for each pivot key.

The evaluation of the average number of indices is shown in Figure 3.19 and verifies the fact that the more effective the re-indexing operations are, less indices are created. The *MULTI* workload requires the less indices, since the trees adapt their pivot levels appropriately. The average number of soft-state indices is larger for workloads skewed towards ℓ_3, and this is a result of various factors. Many trees perform drill-down operations to lower levels of the hierarchy and the tuples are more dispersed among the nodes of the overlay, and thus an indexed value occupy more indices. Moreover, the number of distinct values is larger and increases the probability of

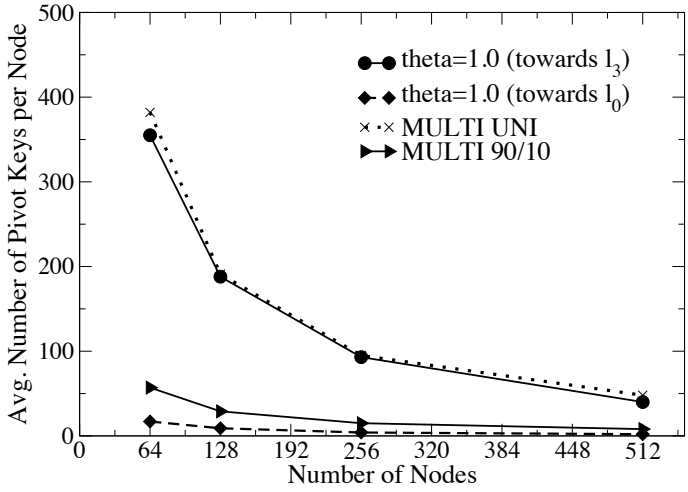


Figure 3.18: *Average number of pivot keys per node*

a non-indexed value to be queried followed by the creation of a new index. The opposite conclusions are valid for the workloads directed towards ℓ_0.

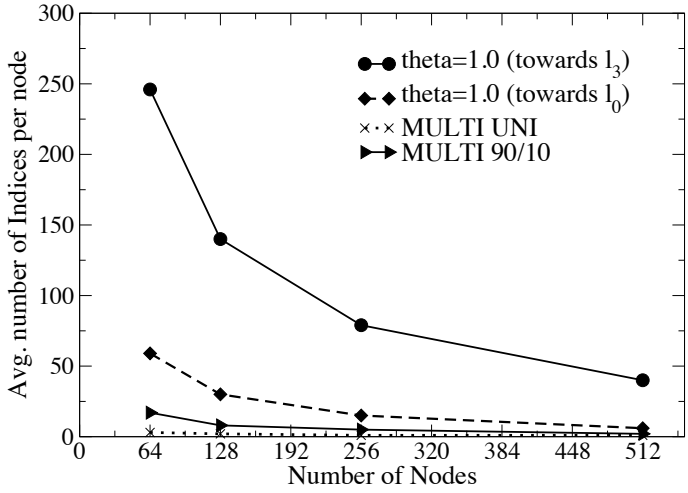


Figure 3.19: *Average number of soft-state indices per node*

The precision of the system is stable independently of the number of nodes participating in the overlay, with an average 1% deviation. Moreover, the number of nodes does not influence the total number of pivot keys and tree nodes, since these values depend mainly on the distribution of the query workload. The number of soft-state indices is affected by the number of nodes in the overlay: less nodes increase the probability of more than one tree containing the same indexed value to be located in the same node and thus only one index to be created. The distribution of

these objects among the nodes is better as the number of nodes increases. According to the exposed results, when more nodes participate in the overlay, each node is responsible for a smaller average number of items.

3.7.6 Effect of the I_{max} Parameter

The I_{max} parameter is configured on a per node basis and defines the maximum number of soft-state indices that a node may store. The creation of a new index results in the removal of *'oldest'* ones, namely the ones not utilized for the longest time, if I_{max} is exceeded. The idea behind this strategy: if the indices to be removed were useful, then they would have been renewed. Otherwise, their removal would not have a negative impact in the performance of the system.

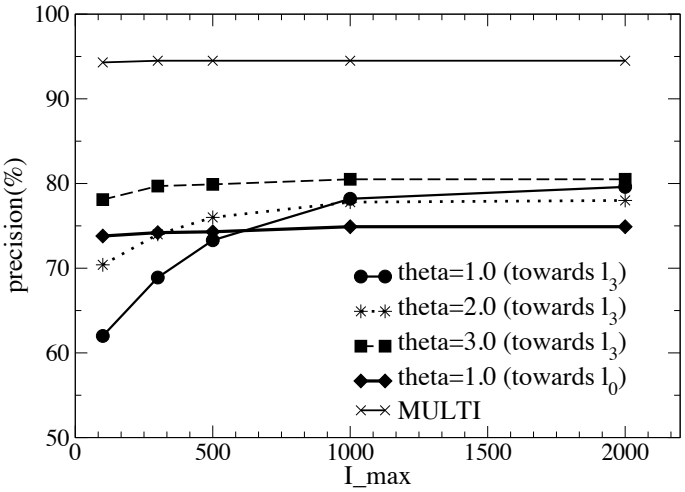


Figure 3.20: *Precision for various UNI workloads*

The I_{max} value has a varying impact on the performance of the system depending on the type of the workloads. The utilization of the indices increases when the query workload does not indicate to the system an evident direction for a re-indexing operation, namely in case that more than one levels are almost equally popular. The more uniform the distribution of queries, the more indices are created.

Figure 3.20 shows the achieved precision for various θ, when the skew is directed towards ℓ_0 and ℓ_3 respectively and the queries target uniformly the values inside a level. The workload directed towards the root level does not seem to be affected by the I_{max} variation, even though the *valueDist* is uniform, which is the most demanding case for indices. This is explained by the permanent existence of the root indices, which are not taken into account during the calculation of the I_{max} value. Moreover, the possible values to be queried become fewer in the higher levels of the hierarchy and therefore less storage space is required for the creation of soft-state indices.

In the opposite case of query workloads directed towards ℓ_3, the increase of the I_{max} value may influence the precision by almost 10%, especially in the case of $\theta = 1.0$, where values from almost all levels are queried and the re-indexing operations are not adequate to cover the demand of values in all levels. The *MULTI* workload is a test case for the system, where the proposed re-indexing algorithms favor its performance at most. The re-indexing decisions depict clearly the query trends and the precision is based on the correction of the pivot level. Therefore, the variation in the value of the I_{max} does not affect the achieved precision.

3.7.7 Performance for Hierarchies with Different Number of Levels

The number of levels may influence the precision of the system as well. The reason is twofold. More levels lead to more possible candidates for a re-indexing decision and increase the probability that queries are resolved with the use of soft-state indices or flooding. In this experiment, 4-level, 6-level and 8-level hierarchies were used. The number of inserted tuples remained 100k and the default insertion level is ℓ_1. In order to maintain the number of total tuples stable, the widths of the trees in each level were narrowed down. The distribution of the query workload, which includes around 50k queries, is generated using $\theta = 1.0$ for *levelDist* and a biased 90/10 distribution for *valueDist*. The presented results refer to query workloads skewed towards the highest and the lowest levels of the hierarchy.

Table 3.2: *Precision for different number of hierarchy levels*

Levels	towards l_0 (%)	towards l_3 (%)
4	86	91
6	80	77
8	78	77

Table 3.2 depicts the achieved precision for each one of the described workloads. The increase in the number of levels results in a decrease in the precision for both type of workloads, as expected. A significant reduction in the number of roll-up operations can be noticed for the experimental results in the query workloads directed towards ℓ_0, as well as a decrease in the drill-down decisions for the query workloads directed towards ℓ_3. Moreover, in both cases a trend of increased re-indexing operations in the opposite direction of the skewness of the workloads appeared. This behavior is justified by the fact that more levels exist and the re-indexing decisions may be more ambiguous requiring corrective re-indexing operations, so as the system to adapt in the incoming queries. It has been also observed that the number of queries answered with soft-state indices increase, especially for workloads directed towards ℓ_0. Moreover, the difference in

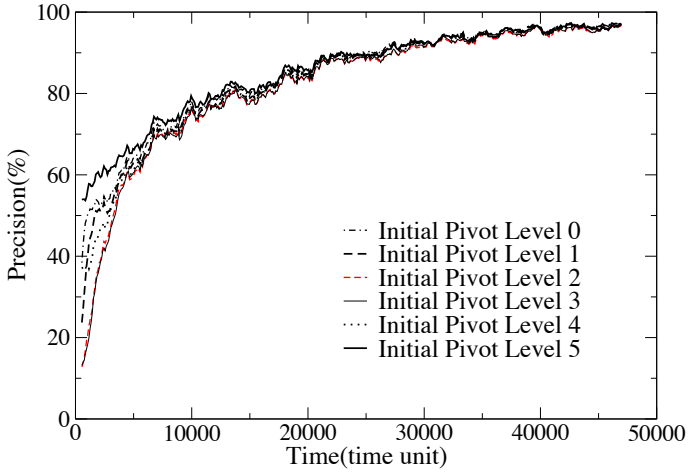


Figure 3.21: *Precision over time for the APB workload for different initial pivot levels*

the achieved precision is more remarkable for the opposite direction of skew. The number of limited distinct values in the upper levels favor the re-usability of indices. As shown in the results of Table 3.2, the decrease in the precision is less, when the skewness is towards ℓ_0.

3.7.8 Performance for Dataset of the APB Benchmark

The adaptiveness of the system is also tested using another category of realistic data. For this reason, query sets by the APB-1 benchmark [apb] were generated. APB-1 creates a database structure with multiple dimensions and generates a set of business operations reflecting basic functionality of OLAP applications. For the executed experiments, the `product` dimension is utilized, a steep hierarchy of 6 levels (the bottom level contains 90% of the members). In more detail, the number of distinct values per level are $|\ell_0 = 50|, |\ell_1 = 150|, |\ell_2 = 800|, |\ell_3 = 3050|, |\ell_4 = 6950|$ and $|\ell_5 = 93050|$. Another characteristic of the specific dataset is that the number of children per node is not constant, as in the synthetically generated datasets of the previous experiments. The query load is skewed towards the lower levels of the hierarchy and 75% of queries refer to values of the ℓ_4 and ℓ_5.

The results are depicted in Figure 3.21, where the precision over time for various initial levels of insertion is shown. It is remarkable that the system adapts to the query load and presents similar performance despite the selection of the level used as pivot level during initial insertions, thus the re-indexing operations, mainly drill-down operations towards ℓ_4 and ℓ_5 and soft-state indices, serve to the incremental precision, which reaches values near 100%.

3.7.9 Updates

In order to measure the cost of incrementally updating the dataset used in the experiments, it was randomly selected the 90% of the tuples, executed each of the described query workloads in 3.7.2 and finally updated the dataset by inserting the remaining 10% of the tuples. It is noted here that the workload plays an important role in the update process as it affects the indexing levels of the stored tuples and, therefore, the update cost (as tuples may have common attributes with existing ones). The cost in messages for storing each of the initial tuples is one lookup message so as to locate the root key and one insertion message to store the tuple. Further lookup messages are not needed, since no other indices than the ones among the root keys and corresponding pivot keys have been created yet. The selected pivot level for the initial tuples is, by default, ℓ_1. The conducted experiments regard the update cost in terms of additional lookups operations to inform existing indices about the appearance of the new tuples. In these set of experiments, we modified the inserted dataset. Table 3.3 contains the average number of lookups per insertion for updating the soft-state indices. This cost can be considered as negligible when the skew is towards high levels of the hierarchy. The maximum documented cost for skewed workloads towards ℓ_3 and uniform *valueDist* is close to 2. The less skewed the distribution the bigger the possibility of soft-index existence in levels other than the popular one. As skew increases, this cost also diminishes.

Table 3.3: *Number of average lookups for updating indices per insertion for different values of* valueDist

valueDist	***Skew towards*** ℓ_0		***Skew towards*** ℓ_3	
	$\theta = 1.0$	$\theta = 2.0$	$\theta = 1.0$	$\theta = 2.0$
uniform	≈ 0	≈ 0	1.99	1.98
80/20	0.01	≈ 0	1.8	1.16
90/10	0.14	≈ 0	0.39	0.25
99/1	≈ 0	≈ 0	0.01	0.02

3.7.10 Other Experimental Results

In this section, other conducted experiments are briefly described. An experiment by varying the number of concurrent queries per time unit for the presented workloads was executed. In this experiment, the queries of the initial workloads of Section 3.7.2 are divided into group of queries. Each group was posed to the system in the start of the time unit. The experimental results for workloads skewed towards ℓ_0 and ℓ_3 are shown in Figure 3.22. The measured precision presents

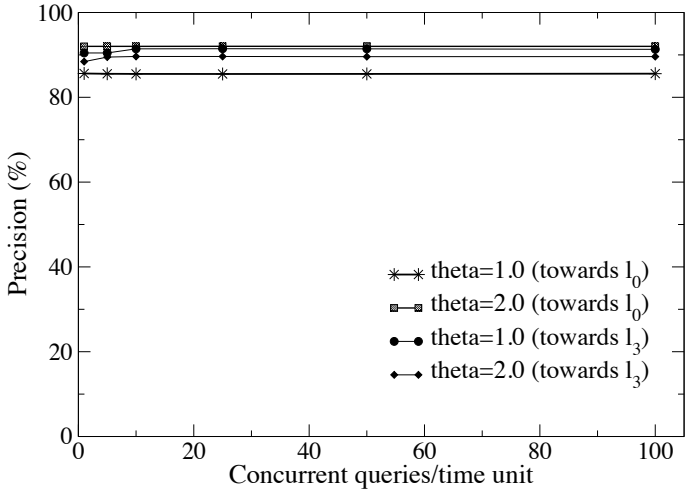


Figure 3.22: *Precision for concurrent queries for valueDist 90/10*

negligible variation, thus showing that the performance of the system remains consistently high while the system scales to a considerable number of concurrent users.

Moreover, experiments conducted with up to 1K nodes showed little qualitative difference. Simulation results for different values of threshold showed that the fluctuation of the precision for $0,2 \leq thr \leq 0,4$ is at most 2%. For values $thr \geq 0,5$ and uniform workloads, the fluctuation reaches 10%. For $thr \leq 0,3$, the initial number of queries to allow re-indexing should increase in order to avoid redundant operations. Experiments for various data distributions with different number of distinct values per level showed no qualitative differences. Another important observation is that by varying the default pivot level the steady-state performance of the proposed algorithm is not affected, since the re-indexing operations and soft-state indices provide the appropriate, adapted indexing structures.

3.8 A Distributed Grid Information System

A motivating scenario for the usefulness of the proposed system can be found in the collection of information produced by information services in Grid environments. Grid computing resources and services advertise a large amount of data, which are used by various users across multiple administrative domains. This information is not intended only for event handling, as in traditional monitoring systems for networks and cluster computing. The produced information by various mechanisms such as cluster monitors (Ganglia [Gan], Hawkey [Haw]), services (GRAM, RLS [glo]), queuing systems, etc, is organized and provided to various applications, such as accounting systems, schedulers, portals, etc. For this reason, the key to the design of Grid Information Services is to identify the information that is required and to determine how to best make it available.

In todays Grid Monitoring Architectures (MDS, R-GMA), data concerning the state of the infrastructure are collected by a combination of various monitoring systems on a resource base and organized by *information producers* (or *providers*). In the previous generation of monitoring architectures, the information producers were organized in a hierarchical structure and published their data, which finally were collected and stored in a central LDAP-based database. In more modern approaches, the information producers are known to the system by subscribing themselves to an *Index service* (MDS) or a *Registry* (R-GMA). Information consumers ask this structure for the location of the producers and contact all of them in order to acquire the needed data. Moreover, various aggregator services exist that collect information (via subscription, polling or execution) from information producers using a common configuration mechanism to specify the type of data and the collected information. For example, VOs maintain such services to collect VO-wide resource information by collecting data from the Information servers running at many sites. Due to the large volume of data and their usefulness to various services, it is strongly believed by the author that a solution for efficient storage and indexing of the produced information, which dynamically adapts to requests of users or services, contributes to the operability and the performance of a Grid infrastructure.

There exist multiple systems and architectures proposed to implement the information system component. The most common is the *Globus Monitoring and Discovery Service (MDS)* [MDS]. A *Grid Index Information Service (GIIS)* provides an aggregate directory of lower level data stored at multiple *Grid Resource Information Services (GRISs)*. The hierarchical structure that can be composed between GIISs enables complete information retrieval by querying the top level GIIS. However, MDS has shown not to be a solution for large-scale production because it does not scale: Multiple client requests quickly lead to an overload of the top level GIIS [ZFS07]. Another schema used especially for accounting and publication of user-level information is the *Relational Grid Monitoring and Discovery Service (R-GMA)* [RGM], which supports complex

type of queries allowed by relational databases. R-GMA presents information as a single virtual database containing a set of virtual tables, nevertheless the bulk of data need be transferred offline to a centralized database after a period of time, with all the performance drawbacks that this entails. The major drawback in all these methods is the fact that none of these architectures scale as the number of data collectors increase [ZFS07]. Moreover, they all assume an offline (or periodic at best) data migration phase to a more central location where global information can be available. In contrast, the presented solution is a completely decentralized system where all data are continuously available and indexed at the requested granularities for fast retrieval.

The architectures of such information systems can be divided into the following structural categories according to the existing technologies and real paradigms implemented in existing infrastructures:

- Centralized solution: The produced information by various monitoring tools is published in a central database. The users query this database in order to retrieve the information of their interest. A usual technique is to replicate the database or maintain such a database per VO, so as to lighten the heavy load in flash crowd situations and to avoid single point of failures.

- Distributed solution: A producer–consumer model is implemented in a distributed manner resulting in the implementation of a virtual database. The producers register themselves within the Registry and describe the type and structure of the provided information. The consumers contact the Registry to find the locations of the producers and afterwards contact them directly to obtain the relevant data, as shown in Figure 3.23 (*distributedCP* model).

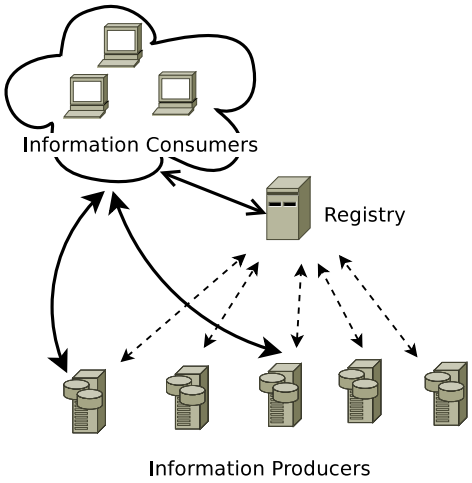


Figure 3.23: *An Information System according to the distributedCP model*

The described system is a complete solution for the organization and storage of such information. The proposed scheme can be integrated in a Grid environment providing a fully decentralized solution. In this architecture, the nodes hosting the information producers and used for information - related purposes are organized in a DHT-based overlay, as shown in Figure 3.24. The routing of messages among these resources is performed according to the DHT protocol and no centralized structures are required. The P2P overlay introduces a scalable solution, while data and query load balancing is achieved. A concept hierarchy is defined for the various levels of aggregation that characterize the produced data. Each numerical fact is described by the corresponding concept hierarchy. No off-line collection and processing of data is required, since online updates are supported. The re-indexing mechanism enables the summarization of data according to the incoming queries. Moreover, the Registry or Index service with the locations of service instances is implemented in a distributed manner. While in existing distributed approaches a consumer queries a central point to find all the relative producers, this procedure is eliminated by the self-organized nature of the described system. The data are dynamically aggregated and stored in a distributed manner according to the preferences of the users achieving reduction of processing and communication cost.

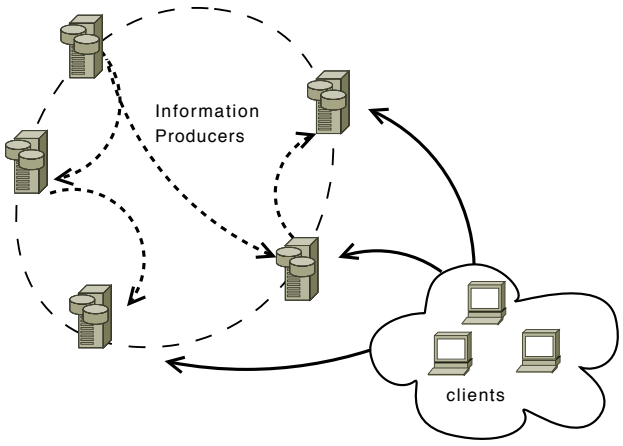


Figure 3.24: *The proposed architecture for the Information System*

An example for the usage of the described integration can be considered for the service providing information for accounting purposes of the EGEE Accounting Portal [ege]. This service uses APEL [et.05], which is a log processing application to filter data produced in each site. Afterwards, R-GMA producers collect data from sites and streams them to a centralized database. The central database collects millions of records per grid job and stores them offline. Due to the enormous volume of data, only summarized views of the various metrics such as Number of jobs, Normalized CPU usage, SumCPU, CPU efficiency, etc, are computed offline and become available to the users. The right balance between quantity and timeliness of information on one hand

and associated costs on the other should be considered for this centralized solution. A simply hierarchy describing this kind of data could be the VO hierarchy used in the previous examples. In Section 3.9, a more detailed concept hierarchy in the experiment for the specific use case is considered and the corresponding data are distributed among the nodes of a P2P overlay.

3.9 Evaluation of the Proposed Grid Information System

The proposed system has been tested for a use case scenario that can be applied to a Grid Information system and tested against existing solutions. In the described experiment, it has been considered that the organization of data and the provided functionality of the EGEE Accounting Portal developed by CESGA [ege] as a reference point, using real queries and data in the simulations. The portal gathers the accounting data of all sites participating in the EGEE and WLCG infrastructures as well as from other sites belonging to other Grid Organizations collaborating with the EGEE infrastructure. The data are further analyzed to generate statistical summaries that are available to the users. The numerical facts of a user's interest are various metrics such as Normalized CPU time, Number of Jobs, etc, being collected by tools for monitoring data of grid resources. In the case of the specific portal, the collected data are further processed and inserted in an offline database. The large volume of the collected values restricts their availability to the users. Aggregated views serving accounting purposes are generated and stored in a central database for presentation. In the proposed approach, a similar notion has been adopted. The monitoring of resources and the collection of values is not investigated, supposed that the "traditional" grid tools (e.g. APEL) are being utilized. Therefore, it is assumed that the numerical facts of various metrics are published and processed by a local service on a per site basis. Afterwards, the published information can be stored in the corresponding Information Servers that may exist in each site. The goal of the specific experiment is to show that the collected information regarding the grid resources can be organized dynamically so as to adapt to the query workload favoring the distribution of load among the nodes. The query load per Information Server is compared to the centralized and distributedCP solution.

The description of the numerical values of the metrics follows a hierarchy of 7 levels based on the presentation of data in the aforementioned application. The hierarchy levels are shown in Figure 3.25.

The specific hierarchy has been used for all categories and projects, so as to maintain a common and general description. In order to assure that a value does not belong to more than one trees with different values in the root level, the data have been parsed and uniquely mapped to integer values taking into account the values between the VO level and each level until the examined level. This assumption can be justified, for example the Number of Jobs differs when the

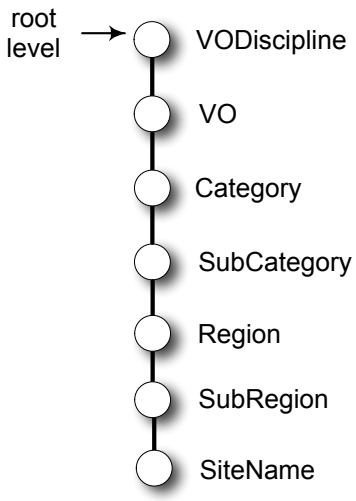


Figure 3.25: *The levels of the concept hierarchy defined for the Grid Information System.*

EGEE sites located in Greece are queried for two different VOs. Moreover, its a common practise in grid infrastructures to include the VO attribute in the queries for the Information Service. According to the information provided by this accounting portal, around 700 sites participate in various categories and projects of the infrastructure. Each site is considered as a different node in the simulation of the proposed system. The size of the inserted dataset is 5789, and only the supported VOs by each site have been considered. For example, the possible values for a site belonging to the *EGEE* project may be:

[High-Energy Physics, atlas, EGEE, Production, SouthEastern, Greece, HG-01-GRNET].

Thus, the sites belonging to other categories may have not been registered with a value for each level of the hierarchy. In this case, the corresponding level has been filled with a value, which is never queried. For example, such a tuple for a site that belongs to *Tier2* has this form:

[Infrastructure, dteam, Tier2,`Tier2Sub`*, UK, UK-London-Tier2, UKI-LT2-HEP].*

The query workload has been created by real queries posed in the EGEE accounting portal during a two month period. Since this application further processes the collected data and aggregates them, it offers the potential that a query may refer to a group of VOs. In the presented results, each query is translated to the equivalent queries and a query for each VO of the group is generated. Each group of queries is submitted to the system at once and a standard rate for the execution of each group is assumed. The same convention has been followed in the simulation for the centralized model and *distributedCP* model. To the extend of our knowledge, it is a common practice to determine the VO when querying the various tools of the Information services. The performance of the proposed architecture is compared to the centralized model, where a central database gathers all the information and answers the queries and the *distributedCP* model, where a central Registry forwards the queries to all the nodes with relative data. The query workload

comprises of around 20k group of queries, which have been translated into 250k of queries using the restriction for a unique VO value for each query. The majority of queries targets ℓ_2, ℓ_3 and ℓ_4 around 85% of the time (19%,36% and 32% respectively).

Experiments have been carried out using all the possible levels as the default insertion level. The precision of the proposed system remains high, with only 3% flooded queries at most. The structure of the dataset and the biased query workload favor the resolution of the queries without flooding. The difference is in the percentage of the exact match queries and the indexed queries. The precision of exact match queries reaches around 65%, when the levels ℓ_2, ℓ_3 and ℓ_4 are selected as the default *pivotlevel*. The selection of these levels as pivot levels appears as good solution for the distribution of the objects amongst the nodes as well.

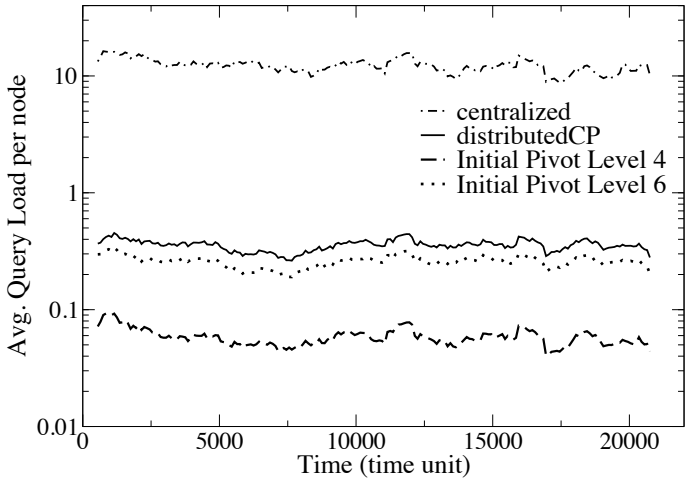


Figure 3.26: *Average load of queries per node*

Figure 3.26 depicts the variations of the average number of resolved queries per node during the simulation time. At the start of a time unit, a group of queries is posed to the system and thus the average load of the centralized solution is equal to the number of queries included in a group. The curve concerning the *distributedCP* model refers only to the average query load of a producer. The load of the Registry node is equal to this of the node of the centralized solution or to a fraction of this load if a Registry node exists per VO. The indexing structure of the proposed system results in a high precision, while each query is resolved by visiting less nodes and thus the query load per node is significantly smaller than in the *distributedCP* model. The average load per node of the proposed Grid Information System reaches the respective load of a node in *distributedCP* model, if l_6 is selected as the default *pivotlevel*. In this case, the indices forward the queries to an almost equal number of nodes as in the *distributedCP* model.

CHAPTER 4

A Distributed System for Interlinking Multidimensional Data

In this chapter, a distributed system for efficient distribution and processing of multidimensional hierarchical data is presented. The system is called LinkedPeers and aims at incorporating two important features: large-scale support for partially structured data and high-performance, distributed query processing. The proposed search strategies mainly focus on the resolution of aggregate operations over multiple dimensions. To enable the efficient resolution of such queries, LinkedPeers utilizes a conceptual chain of DHT rings storing information in a hierarchy preserving manner. Moreover, adaptive mechanisms are implemented adjusting both the granularity of indexing and performing pre-computation of possible future queries according to the incoming query workload. LinkedPeers is evaluated under different data and query workloads proving that the system achieves high precision in answering queries while minimizing communication cost and adapting its indexing to the incoming queries.

The methods introduced in LinkedPeers are adjusted and extended for a real-world use case. The recent Linked Open Data (LOD) initiative integrates large amounts of data from different resources and domains, providing large scale reasoning on data of the Web. The evaluation of queries in such environments mainly occurs along two directions: Centralized and federated systems. While the former are limited over both scalability and size constraints, federation of diverse RDF stores severely hinders performance. Taking into account the structure of linked data and the requirements for efficient management of such data, a P2P Indexing scheme for Linked Data -

PI4LD - is proposed. In this system, the stored entities and classes are organized along hierarchies formed according to the used ontologies, taxonomies and vocabularies. The relationships among entities are maintained in a distributed manner, while data generated by different resources can be integrated to the system. The proposed scheme can either serve as an autonomous solution for hosting linked data or as a mediator over federated resources. PI4LD supports various types of queries, such as simple queries discovering stored entities and at the same time more complex ones. Extensive evaluations using both synthetic and real datasets against a popular centralized RDF store show that PI4LD provides significant reductions in response times.

4.1 Introduction

In a variety of applications, data are described by multiple characteristics (or *dimensions*) such as time, customer, location, etc. The multidimensional data model is an integral component of data warehouses and OLAP tools allowing the representation of data in the form of a *data cube*. Dimensions correspond to perspectives or entities for which the records are maintained and can be further annotated at different levels of granularity through the use of *concept hierarchies* .

Besides the well-documented need for efficient analytics, web-scale data poses extra challenges: While size is the dominating factor, the lack of a centralized or strict schema is another important aspect: Data without rigid structures as those found in traditional database systems are provided by an increasing number of sources, for example data produced among different sources in the Web [Data]. The distribution of data sources renders many centralized solutions useless in performing on-line processing. Consequently, any modern analytics platform is required to be able to perform efficient analytics tasks on distributed, multi-attribute structured data without strict schema.

LinkedPeers is a system that efficiently stores and processes data described with multiple dimensions, while each dimension is organized by a concept hierarchy. A Distributed Hash Table (DHT) substrate is chosen for the organization of *any* number of commodity nodes participating in the system. Data producers can individually insert and update data to the system described by a predefined group of concept hierarchies, while the number of dimensions may vary for each data item. Queries are processed in a fully distributed manner triggering adaptive, query-driven reindexing and materialization mechanisms to minimize communication costs.

The motivation behind the design of *LinkedPeers* is to provide a large-scale distributed infrastructure to accommodate collections of partially-structured data. In contrast to approaches where both data and their relationships are pre-defined by rigid schemas, the specific system intends to support a higher degree of freedom: System objects are described by d dimensions, each of which is further annotated through a corresponding concept hierarchy. *LinkedPeers* does

not require that each inserted fact be described by values for all dimensions. On the contrary, it attempts to fully support it and not restrict the ability to efficiently process it.

LinkedPeers manages to preserve all hierarchy-specific information for each dimension, using a tree-like data structure to store data and interlinking trees among different dimensions. A natural ordering of the dimensions that stems from their importance, query skew, etc, yields to a corresponding organization of the DHT layer: *LinkedPeers* comprises of multiple "virtual" overlays, one for each dimension. In practice, all dimensions can share the same *identifier space* and use a single underlying DHT overlay instead of creating a different one for each dimension. In this case, the separation of rings in each dimension is "virtual". An abstract representation of the virtual rings of *LinkedPeers* for d-dimensions is shown in Figure 4.1. The strategy of constructing as many rings as the number of dimensions results with each object being split into d parts and ending up in nodes of the *primary* and *secondary* rings. Trees at secondary rings maintain information towards related trees of the primary ring.

The purpose of this design is to couple the operational autonomy of the primary ring with a powerful meta-indexing structure integrated at the secondary rings, allowing the system to return fast aggregated results for the queried values by minimizing the communication cost. By allowing adaptive result caching and precomputation of related queries, this efficacy is further enhanced.

The proposed scheme enables the processing of complex aggregate queries for any level of any dimension, such as: "*Which* `Cities` *belong to* `Country` *'Greece' ?*" or "*What is the* `population` *of* `Country` *'Greece' ?*" or "*Which* `Cities` *of* `Country` *'Greece' have* `population` *above 1 million in* `Year` *'2000'?*", considering that the `Location` (see Figure 3.1) and `Time`(see Figure 3.4) hierarchies describe a numerical fact for population. The enforced indexing allows to find the location of any value of any stored hierarchy without requiring any knowledge, while aggregation functions can be calculated on the nodes that a query ends up.

To summarize, *LinkedPeers* offers the following innovative features:

- A complete storage, indexing and query processing system for data described by an arbitrary number of dimensions and annotated according to defined concept hierarchies. *LinkedPeers* is able to perform efficient and online incremental updates and maintain data in a fault-tolerant and fully distributed manner.

- A query-based materialization engine that pro-actively precomputes relevant views of a processed query for future reference.

- Query-based adaptation of the indexing granularity of its indexing according to incoming requests.

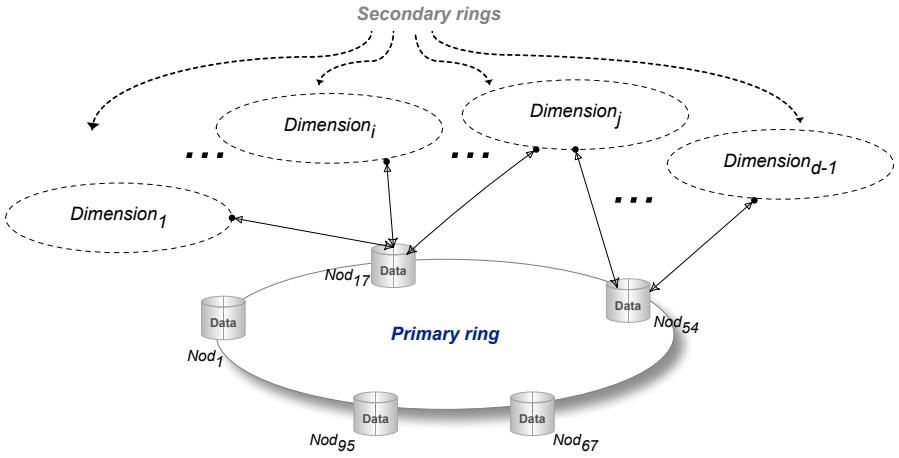


Figure 4.1: *An abstract view of LinkedPeers architecture*

To support the presented analysis, a thorough performance evaluation is exhibited in order to identify the behavior of the scheme under a large range of data and query loads.

4.2 Notation and Definitions

Data items are described by tuples containing values from a data space domain D. These tuples are defined by a set of d dimensions $\{d_0, .., d_{d-1}\}$ and the actual fact(s). Each dimension d_i is associated with a concept hierarchy organized along L_i levels of aggregation ℓ_{ij}, where j ($j \in [0, L_i - 1]$) represents the *j-th* level of the *i-th* dimension. It is defined that ℓ_{ik} lies *higher* (*lower*) than ℓ_{il} and denote it as $\ell_{ik} < \ell_{il}$ ($\ell_{ik} > \ell_{il}$) *iff* $k < l$ ($k > l$), i.e., if ℓ_{ik} corresponds to a less (more) detailed level than ℓ_{il} (e.g., $Month < Day$). Tuples are shown in the form:

$$\langle v_{0,0}, \ldots, v_{0,L_0-1}, \ldots, v_{d-1,0}, \ldots, v_{d-1,L_{d-1}-1}, f_0, ... \rangle$$

where $v_{i,j}$ represents the value of the j-th level of the i-th dimension. Note also that any *value-set* $(v_{i,0}, \ldots, v_{i,L_i-1})$ for the i-th dimension may be absent from a tuple and that *fact* (e.g., f_0) may be of any type (e.g., numerical, text, vector, etc), but one dimension has to be considered as primary. Level ℓ_{i0} is called the *root level* for the *i-th* dimension and its hashed value v_{i0} is called *root key*. The values of the lowest level of a hierarchy $(v_{i,(L_i-1)})$ are also referred to as *leaf values*.

The values of the hierarchy levels in each dimension are organized in tree structures, one per root key. Without loss of generality, it is assumed that each value of ℓ_{ij} has at most one parent in $\ell_{i(j-1)}$. To insert tuples in the multiple rings, one level from each dimension hierarchy is chosen; its hashed value serves as its *key* in the underlying DHT overlay. Any reference to this

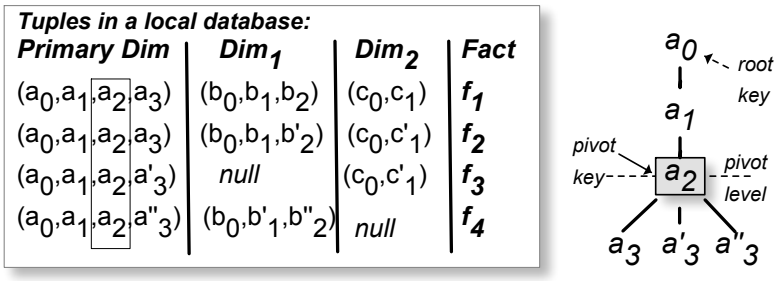


Figure 4.2: *A group of tuples with various value combinations among dimensions and the resulted tree structure for the primary dimension.*

level is noted as *pivot level* and to its hashed value as *pivot key*. The pivot key that corresponds to the primary dimension (or *primary ring*) is called the *primary key*. The highest and lowest pivot levels of each hierarchy for a specific root key are called *MinPivotLevel* and *MaxPivotLevel* respectively.

The value-set of a dimension along the aggregated fact are organized as nodes of a *tree structure*, which contributes to the preservation of semantic relations and search. Figure 4.2 describes the running example. The shown tuples adhere to a 3-dimensional schema. The primary dimension is described by a 4-level hierarchy, while the other two are described by a 3-level and a 2-level hierarchy respectively. Note that the last two tuples do not contain values in d_1 and d_2 respectively. The selected pivot level for the primary dimension is ℓ_{02} and thus all the shown tuples have the same pivot key in the primary dimension. All the value-sets in each dimension are organized in tree-structures with common root keys.

The basic type of query supported in *LinkedPeers* is of the form:

$$q = (q_{0k}, ..., q_{ij}, ..., q_{(d-1)m})$$

over the fact(s) using an appropriate aggregate function. By q_{ij} is denoted the value for the *j-th* hierarchy level of the *i-th* dimension which can also be the special '*' (or *ALL*) value.

4.3 Data Insertion

The proposed system handles both bulk insertions and incremental updates in a unified manner. As the design of the system implies one virtual overlay per dimension, one key (using the SHA1 hash function for instance) for a selected pivot value of each dimension is generated.

During data insertions, the information about the pivot value is vital (only for initial insertions the pivot level can be selected according to the needs of the application). The design of *LinkedPeers* assumes if a value v_{ij} is selected as a pivot key during the insertion of a tuple, every other tuple that contains v_{ij} must also select it as its pivot key for the i-th dimension. To comply

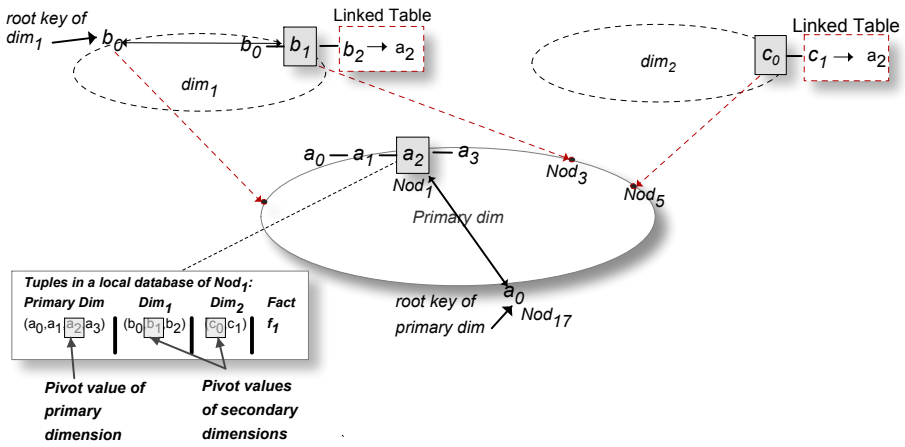


Figure 4.3: *The created data structures after the insertion of the first tuple of Figure 4.2*

with this assumption, a node should be aware of the existing pivot keys during the insertion of a new tuple. Thus, a fully decentralized catalogue storing information about root keys and their respective pivot keys in the network is implemented in *LinkedPeers*. Each root key is stored at the node with ID closest to its value. Every time that a new pivot key corresponding to this root key is inserted in the system, the root key node is informed about it and adds it in a list of known pivot keys. The root key node is also aware of the MaxPivotLevel used during the insertion of its values in the specific dimension.

The procedure for inserting the values of a tuple appropriately in all dimensions constitutes of the following basic steps:

- Inform each root key of every dimension about the corresponding value-set $(v_{i,0}, \ldots, v_{i,L_i-1})$ of the tuple, so as to decide for the appropriate pivot level.
- Insert each value-set $(v_{i,0}, \ldots, v_{i,L_i-1})$ to the corresponding i^{th}-ring.
- Create or update links among the trees of secondary dimensions towards the primary dimension.

Initially, the initiator contacts the root key of the primary dimension's value-set. The root key of the primary dimension is informed about the new tuple and indicates the appropriate pivot level: if the same pivot key already exists, then its pivot level is used, otherwise the MaxPivotLevel. In case that the root key does not already exist, then it is stored in the node responsible for it and the pivot level is chosen either randomly or according to a predefined pivot level for the whole system. Afterwards, the DHT operation for the insertion of the tuple in the primary dimension

starts and the tuple ends up to the node responsible for the decided pivot key. The node responsible for the pivot key of the primary dimension stores its value set in a tree structure and the whole tuple in a store defined as its *local database*. Moreover, it stores the result(s) of the aggregate function(s) over all these tuples that have the same value in each level (i.e., the results for $(v_{0j}, *, \ldots, ^*)$ queries, where $j \in [0, L_i - 1]$). Figure 4.3 demonstrates the insertion of the value-set (a_0, a_1, a_2, a_3) in the primary ring of an overlay consisting of nodes referred to as Nod_i. The root key a_0 does not exist in the overlay and ℓ_{02} is selected randomly as pivot level. The root index is created from a_0 towards a_2 and the tuple is inserted to the node Nod_1, which is responsible for the pivot key of the value a_2 according to the DHT protocol. Nod_1 inserts all the values of the tuple in its local database as well.

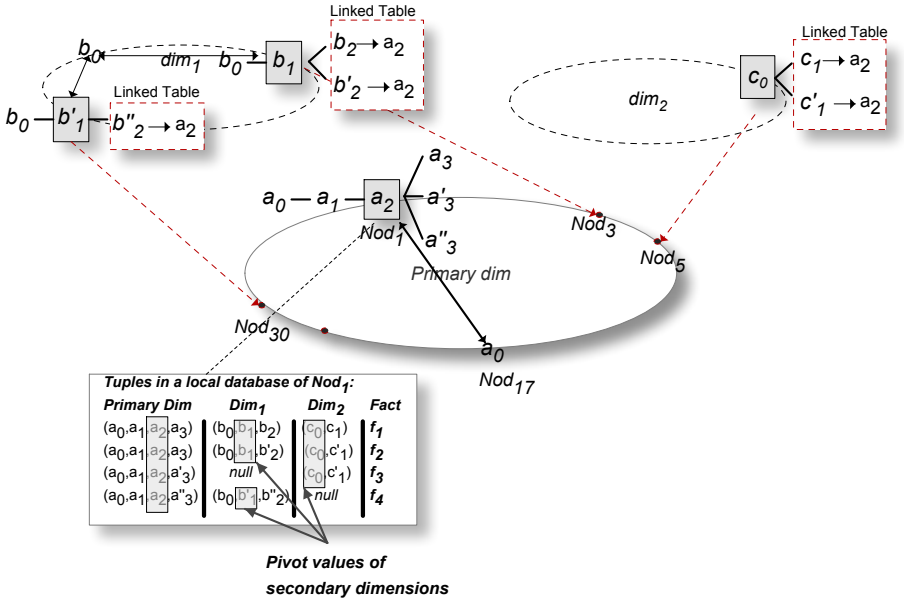


Figure 4.4: *Final placement and indexing of the tuples of Figure 4.2 in LinkedPeers*

The next step is to store the value-sets for the remaining dimensions in the corresponding rings. The node responsible for the primary key contacts each node responsible for the root keys and is informed about the appropriate pivot level in d_i. Since the pivot levels for the secondary dimensions are determined, the value-set of each dimension is stored in the node responsible for its pivot key. Again, the respective aggregates are also maintained in the nodes of the trees. The values of the secondary dimensions are associated to the primary dimension through the primary key. Each leaf value of a secondary tree structure maintains a list of the primary keys that is linked to. The structure storing the mappings among the leaf values and the primary keys is referred to as *Linked Table*. The node holding the primary key also stores the pivot levels of the

value-sets of the secondary dimensions in its local database along with the whole tuple. Another remark is that if the insertion of tuples does not take place during the initial loading of data in the system and the root key already exists, then any existing soft-state indices should be updated as well. The procedure described in Section 3.5.1 can be considered for the value-set that its root key existed. In this case, since soft-state indices may store the aggregated facts for the indexed value, the soft-state indices should not only get informed about the locations of the new trees, but also about the new facts. In case that the tree already existed, the marked values as indexed should also learn about the new tuple.

In Figure 4.3, the tree structures comprising of only one branch for the secondary dimensions are shown as well. During the insertion of value-set (b_0, b_1, b_2), the root index b_0 is created (the pivot level for c_0 is the root level and no further indexing is needed). Figure 4.4 shows the final placement of the values in the tuples of Figure 4.2 among the nodes of the overlay. When the second tuple is inserted in the overlay, the root index for a_0 indicates that the value a_2 exists already as pivot key and thus this tuple needs to be stored in Nod_1. A new branch below the pivot level is inserted in the existing tree. The values of d_1 do not exist in the third tuple, but this fact does not affect the procedure of the insertion. The insertion of the values for d_1 results also in the construction of a new tree for the pivot value b'_1. Since the primary key of all tuples is the same, the local database of Nod_1 contains all the rows shown in Figure 4.2.

4.4 Query Processing

The queries posed to the system are expressed by conjunctions of multiple values. When a query includes a pivot value, then the node responsible for this value can be found with a simple DHT lookup. Otherwise, the native DHT mechanisms are not adequate to search the rest of the stored values. The proposed techniques can be further utilized to enable the search for any stored value.

The idea behind the approach followed for the insertion of tuples in the DHT overlay is the maintenance of the linking among the multiple dimensions, which can be searched either independently from each other or in conjunction with others. When the query does not define a specific value for a dimension (a '*'-value), then any possible value is acceptable for the query. A query is assumed to include up to *d-1* '*' for *d dimensions*.

LinkedPeers allows adaptive change of pivot levels according to the query skew. Therefore, query initiators are not aware if any of the queried values correspond to a pivot value, forcing them to issue consecutive lookups for any value contained in the query according to the dimension priority, until they receive a result. Initially, a *lookup* operation is initiated for the value of the dimension with the highest priority. If the node holding the queried value cannot be located

by the DHT lookup, then a lookup for the next non-'*' value follows. If no results are returned for all the values in the query, then the query is flooded among the nodes of the overlay.

4.4.1 Exact Match Queries

Queries concerning a pivot value of any ring are called `exact match` queries and can be answered by the DHT lookup mechanism. The are two categories of an exact match query:

Category 1: Query is $q = (q_{0pivotlevel}, \ldots)$, where a pivot value of the primary dimension is defined in the query. Any other values may be included for other dimensions as well. The DHT lookup ends up at the node responsible for the pivot key of the primary dimension. If this is the only value asked, the corresponding tree structure is searched for the aggregated fact. Otherwise, the local database is scanned and the results are filtered according to the remaining values locally.

Category 2: Query is $q = (q_{0j}, \ldots, q_{ipivotlevel}, \ldots)$, where q_{0j} does not correspond to a pivot value. In this case, a queried value in one of the secondary dimensions is a pivot value. The strategy followed to resolve this query is that consecutive queries are issued until the node responsible for $q_{ipivotlevel}$ is reached. If the query contains no other values, then the tree structure of this node is adequate to answer it, otherwise the query is forwarded to all the nodes of the primary dimension that store tuples containing $q_{ipivotlevel}$. These nodes query their local databases to retrieve the relative tuples and send back the results to the initiator. If more than one pivot values are present in the query, then the query is resolved by the dimension with the highest priority. In the example of Figure 4.4, a query for value b_1 can be resolved by the aggregated fact stored in Nod_3. On the other hand, a query for the combination of values $(a_3, b_1, *)$ reaches Nod_3, which does not store adequate information to answer it and (using its Linked Table) forwards it to Nod_1, which queries its local database.

4.4.2 Flood Queries

Queries not containing any pivot value cannot be resolved by the native DHT lookup. The only alternative is to circulate the query among all nodes and process it individually. In case that the query contains a single value, then the tree structures of each node are searched. Otherwise, a node searches its local database for the queried values and sends the found results back to the initiator.

To minimize the communication and processing costs, extra steps are taken for the resolution of a flood query. Both the DHT mechanisms and the properties of the data structure are utilized to avoid visiting the same node multiple times and impose an order in the way that the nodes are visited, instead of flooding the query in an uncontrolled manner. The hierarchical structure of data along with the imposed indexing scheme enable a controlled flooding strategy that significantly reduces the communication cost.

Initially, a flood query is forwarded from a node to its closest neighbour in the DHT substrate. Each visited node searches its tree structures for any of the values included in the query. It also searches its local database for any of the queried values and the combination of values included in the query. If nothing is found in the reached node, then the current node registers the ID range(s) under its responsibility in the flood message and forwards the query to its closest neighbour. This strategy is enforced so as to avoid visiting the specific node again during the rest of the procedure for the flood resolution. The reasoning behind this strategy is that if a node has been already queried and does not store any relative tuples to the query, then there is no benefit of searching the same node again, even if it is indicated as a candidate node for holding tuples that answer the query.

If any relative information to the query is found in a reached node, then the query forwarding stops. In case that the queried value is found in a tree structure of the node, then this node becomes the *coordinator* of the flood procedure. If more than one of the queried values are found in the same node, then the query is resolved in the 'virtual' ring of the dimension with the highest priority. The possible cases of a found flooded value are two: either to belong to a level above the pivot level or to a level below the pivot level. The referred node does not become the coordinator, when a value is found in one or more tuples of the local database. Nevertheless, in this case there is enough information in the stored tuple to find out if the found value is located above the pivot level or below the pivot level, so as to forward the flood query either to the root key of this value or its pivot key respectively. Any found tuples answering the specific query (or aggregated facts) are also included to the flood message during the forwarding of the query. Apart from this additional step, the procedure for resolving the query continues as described below without any other changes.

Assuming that the found value is located below the pivot level, then there are no other trees with the specific value. The node either sends the result to the initiator of the query (if the query involves only a single value or the found value belongs to the primary dimension) or forwards the query along its links to the nodes of the primary dimension excluding the already visited ones. The strategy described for the second category of the exact match queries is followed. The nodes with the primary keys respond with the relative tuples or the aggregated fact. These results are collected by the coordinator and sent back to the node that initiated the query.

In case that the found value belongs to a level above the pivot level, there may exist other trees with the same value, even if one has been already found. For example, if a flood message for value a_1 in Figure 4.4 reaches Nod_1, other nodes with the value a_1 and different pivot keys may also exist. Yet, it is certain that this value is not stored at a tree having a different root key. Thus, the flood message is forwarded to the node with the corresponding root key, which becomes the coordinator of the procedure from now on. This node forwards the flood query to the nodes whose pivot keys is aware of, excluding the nodes that have been already visited. If the

found value belongs to the primary dimension or the query does not involve any other values, then nodes respond with the relative tuples or the aggregated fact respectively. Otherwise, each node includes in its response any relative facts that may have found in its local database and a set of candidate links that the pivot key(s) of the found value is(are) linked in the primary dimension. Upon receiving all the results, the coordinator merges the links and excludes from querying the nodes that have been already visited. Finally, the local databases of the remaining nodes are queried and the returned results are merged with the already found ones and returned back to the initiator.

4.4.3 A Query-driven Approach for Partial Materialization

In many high-dimensional storage systems, it is a common practice to pre-compute different views (GROUP-BYs) to improve the response time. For a given data set R described by d (dimensions) annotated by single-level hierarchies, a view is constructed by an aggregation of R along a subset of the given attributes resulting in 2^d different possible views (i.e., exponential time and space complexity). The number of levels in each dimension adds to the exponent of the previous formula. In *LinkedPeers*, a query-based approach is considered to tackle the view selection problem: The selection of which "views" to pre-compute is query-driven, as it is taken advantage of the evaluation process to calculate parts of various views that are expected to be needed in the future and maintain "partial materialized views" in a distributed manner.

Figure 4.5 depicts all the possible combinations of the values of the query (a_1, b_2, c_1), relative to Figure 4.4. The attributes participating correspond to levels $\{\ell_{01}, \ell_{12}, \ell_{31}\}$ respectively. Each combination of values consists of a subset of attribute values in $\{d_0, d_1, d_2\}$ ordered according to the priorities of dimensions in decreasing order. A possible combination of values that can be queried is mapped to a "view identifier" comprising of the respective values. When a view identifier (or combination respectively) is "materialized", then the result for this combination of queried values is computed and stored for future use. For example, the view identifier (a_1, c_1) in Figure 4.5 stores the results of the query $(a_1, *, c_1)$. Moreover, each view identifier in the *i*-th level of the tree structure in Figure 4.5 is deduced by its ancestor view identifier in *(i-1)*-th level by omitting the participation of one dimension each time. When a value of a dimension is omitted in a view identifier, then it is considered that its value is a '*'-value. The identifiers that have already registered on the left-side of this tree are omitted.

Let $S_i \subset S$ be the subset of view identifiers that start with the attribute value defined in dimension d_i. The subset of the specific view identifiers is called $Partition_{d_i}$ and the dimension that participates in all identifiers of the partition as $Root_{d_i}$. In Figure 4.5, $Partition_0$ comprises of all view identifiers that contain a_1, which is the $Root_0$, while a_1 does not appear in any identifier of the remaining partitions.

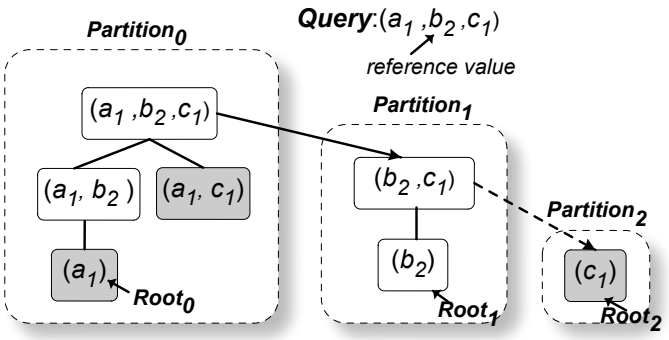


Figure 4.5: *All possible view identifiers for a query combining values in 3 dimensions.*

According to the strategy followed during flooding, all the nodes with trees containing the found value used for the resolution of the query (hence *reference value*) are definitely contacted. Thus, it can be concluded with certainty that there exist no extra nodes with tuples containing the reference value. This assumption is not valid for the rest of the values included in the query. This observation is significant for determining which combinations can be materialized and stored for future queries in a distributed manner.

Let S be the set of all the 2^d identifiers. It can be deduced that only a subset $S_{partial} \subset S$ of the view identifiers can be fully materialized, namely only the identifiers of the combinations including the reference value. In the example of Figure 4.5, let us assume that the flood query for the combination (a_1, b_2, c_1) reaches Nod_3 and the reference value is b_2. The query will be forwarded to Nod_1 and it will be resolved. Nevertheless, it is not ensured that there are no other nodes storing tuples with a_1 or c_1. Thus, $S_{partial}$ comprises of the view identifiers in the *non-grey* boxes, which can be materialized.

In more detail, the calculation of the partial views occurs among the nodes of *LinkedPeers* as follows: Each peer that returns a found aggregated fact in a flood query, it also calculates the available view identifiers in $S_{partial}$ stored in its local database. Due to the flooding strategy, every peer with trees containing the reference value will be definitely contacted. According to this procedure, the following conclusions are made:

- The $S_{partial}$ may comprise only of identifiers belonging to $Partition_{d_0}, Partition_{d_1}, .., Partition_{d_{ref}}$, where the $Root_{d_{ref}}$ of $Partition_{d_{ref}}$ is the reference value used for the resolution of the flood query.

- If the query is flooded in all the nodes of the network, then all the combinations of the queried values can be calculated resulting in $2^d - 1$ combinations ('ALL' is not materialized), if the query does not contain any '*'-value. In case of '*'-values, the number of view identifiers is $2^{d-n} - 1$, where n is the number of '*'-values. If the described strategy for

minimizing the visited nodes during the flood of a query is enforced, then only the combinations that contain the reference value can be calculated. Nevertheless, taking into account the type of the inserted dataset (number of dimensions, number of tuples), the type of the query workload (average number of '*'-values per query) and the specifications of the system (i.e., bandwidth consumption, storage capacity) various policies can be defined to limit the number of calculated aggregated results.

Upon the reception of all the results, the coordinator merges the returned aggregated facts for each view identifier. Afterwards, it calculates the hash value of each $Root_{d_j}$ and inserts each $Partition_{d_j}$ ($j \in [0, d_{ref}]$) to the overlay. The node responsible for the $Root_{d_{ref}}$ also creates *indices* towards the locations of its tree structures to forward any query that cannot be resolved by the stored materialized views. The idea behind the splitting of the partitions is that the stored combinations need to be located with the minimum message cost, namely with the primitive DHT lookup. Since a query is disassembled in its elements and the queries are issued according to the priority of the dimensions, each identifier is stored to the dimension with the highest priority of its values.

Although any approach of existing relational schemas for storing views could be utilized to store the aggregated facts, simple 'linked-listed' structures are maintained in *LinkedPeers* storing the different view identifiers, along with the corresponding facts. As shown in Figure 4.6, the materialized view identifiers of Figure 4.5 are stored to the nodes responsible for the values appearing in the 'dark grey' boxes. All the queries arriving at the node responsible for $Root_0$ (namely a_1) should also include the $Root_{d_{ref}}$, which is b_2. The combination of value(s) that a query should at least include so as to be resolved by one view identifier of such a group is marked with red boxes. It may also include the value(s) contained in the white boxes. For instance, the queries $(a_1, b_2, *)$ and (a_1, b_2, c_1) can be directly answered by the calculated results stored in the node

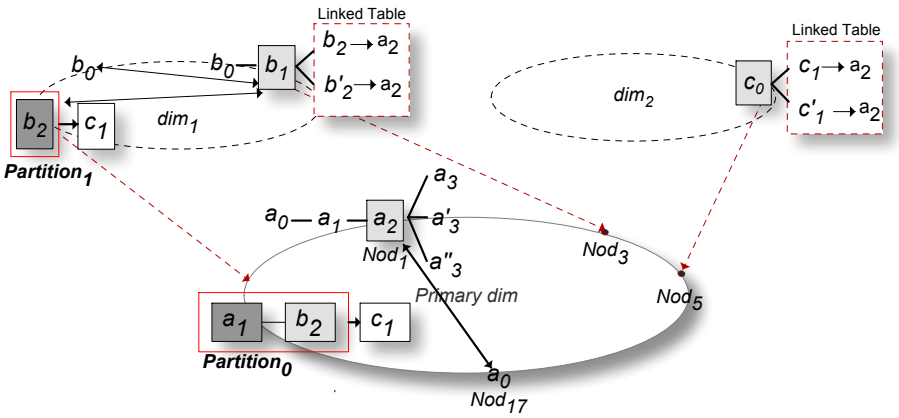


Figure 4.6: *Distribution of materialized view identifiers among the nodes of LinkedPeers*

responsible for the $Partition_0$, as shown in Figure 4.5. The key used by the DHT for assigning these two view identifiers to the appropriate node is the hashed value of a_1. Figures 4.7 and 4.8 depict a different example for the case that the there is data with four dimensions in the system. The posed query is (a_i,b_j,c_k,d_l), while all the values in all four dimensions do not correspond to any pivot value. The possible combinations and the calculated ones (contained in white boxes) are shown in Figure 4.7 after the query is resolved using the value c_k as a reference value. An indicative distribution of the materialized view identifiers is shown in Figure 4.8.

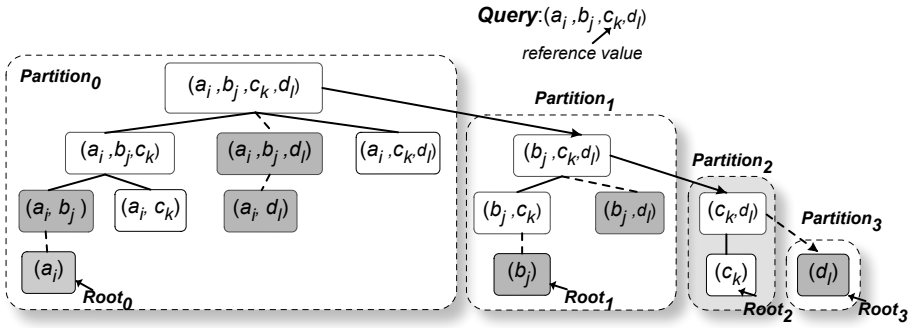


Figure 4.7: *All possible view identifiers for a query combining values in 4 dimensions.*

The created indices and view identifiers are soft-state in order to minimize the redundant information. This means that they expire after a predefined period of time (Time-to-Live or *TTL*). Each time that an existing index is used, its TTL is renewed. This constraint ensures that changes in the system (e.g., data location, node departures, etc) will not result in stale indices, affecting the validity of the lookup mechanism. The indices are bidirectional to ensure data consistency during re-indexing operations. Finally, a limit is posed to the maximum number of indices held by each node. Overall, the system tends to preserve the most "useful" indices towards the most frequently queried data items.

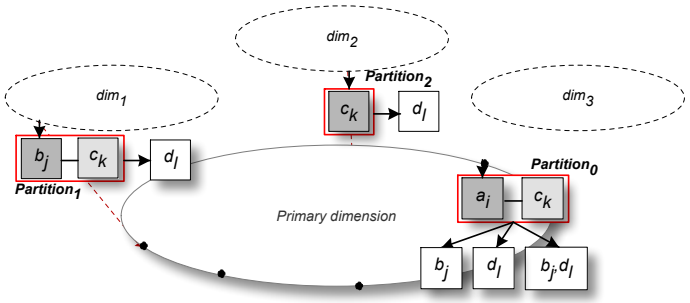


Figure 4.8: *Distribution of materialized view identifiers among the nodes of LinkedPeers for the 4-dimensional example*

During the update procedure, there are different alternatives that can be considered for updating the materialized combinations, since their aggregated facts may change. On one hand, it can be assumed that since the partial materialized views are soft-state, it can be avoided to explicitly update their values and rely on the fact that after their expiration, the new values will be taken into account. On the other hand, if the returned results need to be accurate and there are strict constraints on this, then a strategy can be enforced that lookups about the values of the new tuple(s) and updates any materialized combinations of values appropriately.

4.4.4 Indexed Queries

When a query reaches a node holding an index, then the stored view identifiers (if any) are searched for the combination of values included in the query. If the combination is found, the aggregated value is returned to the initiator. In the case that the combination does not exist, but the index is aware of the nodes with the pivot keys for the specific value, the query is forwarded to the respective pivot keys. If the query is simple or the found value belongs to the primary dimension, then the aggregated facts for the query are returned. Otherwise, the reached nodes return the locations of the primary ring that are correlated with the indexed value. The query is forwarded to these nodes contacting their local databases. After an indexed query which has not been resolved with the use of a stored view identifier, the procedure for materializing all the possible view identifiers described in the Section above is followed.

The nodes with actual tuples of the indexed value need to know the existence of an index. The bidirectionallity of the indices is introduced only to ensure data consistency, despite of them being soft-state. During re-indexing operations, the locations of stored tuples change and indices correlated to these tuples need either to be updated or erased, preventing the existence of stale indices. It has been chosen to erase them, so as to avoid increasing the complexity of the system. Detailed information for an existing index is not essential for the node, where the tuples are stored. A simple mark for each indexed value is adequate in order to erase its index, if needed. In this case, some redundant operations for erasing expired indices may occur. If there are no memory restrictions and local processing is preferable to bandwidth consumption, indexed values can be marked with a time-stamp. Every lookup for an indexed value renews the TTL in both sides of the index and only valid indices are erased during re-indexing operations. The created views holding the data for the calculated combinations are not aware of the locations of the trees and for this reason views are only soft-stated.

4.5 Adaptive Query-driven Re-indexing

A significant feature of the system is that it dynamically adapts its indexing level on a per node basis to incoming queries. To achieve this, two re-indexing operations regarding the selection of pivot level are introduced: *Roll-up* towards more general levels of the hierarchy and *drill-down* to levels lower than the pivot level.

The idea behind the decision procedure is based on the fact that a node is more capable of detecting if the values for a level $\ell_{ij} > pivotlevel$ (where *pivotlevel* denotes the pivot level of a specific hierarchy in the i-th dimension) of a tree are queried at most and thus to proceed to re-indexing operation. On the contrary, it has to cooperate with the rest of the nodes storing a value for a level $\ell_{ij} < pivotlevel$ to obtain a global view and decide if a re-indexing operation towards this level for the involved trees would be beneficial. Therefore, the node has sufficient information to decide if a drill-down will be favorable for the values of this tree. A roll-up towards a level $\ell_{ij} < pivotlevel$ is decided by all the involved nodes storing trees with the specific value in this level. The decision for a possible re-indexing operation is made according to statistics collected by the incoming queries in the trees responsible for the specific value used during the resolution of a query, as discussed in Section 3.5. Since some queries are resolved by the node holding an index and are not further forwarded to the node(s) with the actual tree structure(s), some statistical information is maintained in these nodes. This information is pushed to the referred nodes and merged with their maintained records, as soon as another query needs to be forwarded to these nodes by the index. The queries answered with the use of view identifiers are not counted in the decisions for re-indexing, since they are resolved directly by the node holding them and the query processing is not encumbered. The goal is to increase the number of queries answered as exact matches in each dimension.

Each time that a value of a tree structure is looked up, then the maintained statistical information is updated, as described in Section3.5. The decision process for a possible re-indexing operation can be triggered after an indexed query resolved without using a materialized view or a flood query and only for the reference value. As far as the indexed queries are concerned, a node holding the tree with the specific value checks for a re-indexing operation after a number of indexed queries has been received by the specific node. When a tree structure is searched during the resolution of a flood or indexed query, then the respective statistical information is checked to find out if a re-indexing operation is indicated. If a drill-down is decided or a roll-up seems probable, then this alarm is included to the answer of the node. Since only the tree structures for the reference value are definitely visited, a re-indexing decision is examined only for this value and not all the values contained in the query. The procedure for deciding if a re-indexing operation is advisable is performed according to the algorithms proposed in Section3.5. Nevertheless, major enhancements have been made for the customization of the re-indexing operations in

multiple dimensions due to the requirements arisen from the existing links among the rings. If a re-indexing operation is not needed after a flood query, then no action is taken other than the creation of the soft-state indices and materialization of the view identifiers.

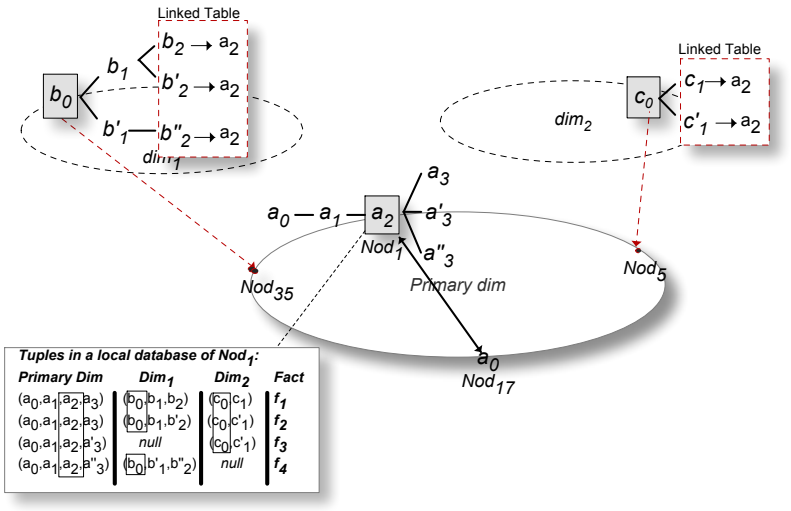


Figure 4.9: *Re-organization of the shown tree structures of dim_1 after a roll-up operation towards ℓ_0*

Roll-up: In general, if a node detects that the demand on a value above the pivot level relatively exceeds the demand for the other levels, it initiates the procedure to decide if a roll-up towards this level would be beneficial (by communicating with the other nodes holding this value). For this reason, it sends a remark to the node collecting the results that a possible roll-up operation towards the queried level needs to be examined. When the coordinator (if a flood operation occurs) or the node that holds the index is informed about the requisite of a possible roll-up towards the queried level, it starts the collection of statistical information from all trees containing the queried value. Afterwards, it decides if a roll-up operation is needed and a positive decision leads to the re-insertion of all trees containing the specific value with the new hash value in the overlay and the trees with the old pivot values are deleted. During a roll-up, one or more nodes re-insert their trees, which end up in one node responsible for the new pivot key. If the roll-up value belongs to a primary dimension, then all the relative tuples in the local database are transferred to the new node. Each node also informs the root key about the pivot key(s) to be erased and the new pivot key to replace it (them) and erases all the soft-state indices towards any value of the re-indexed trees. The root key waits to receive the messages for updating its list of pivot keys from all the nodes participating in the roll-up operation and afterwards replaces the old pivot keys and nodes with the new ones. In the meantime, queries concerning any value of the trees participating in a roll-up operation are answered by the nodes responsible for the old pivot keys. The stored view identifiers containing any of these values in other rings are not affected,

since the relocation of the trees does not influence the stored, aggregated facts. The final step is the update of the links among the primary and the secondary rings, since the links need to be valid for the resolution of future queries. If the roll-up operation is performed in the primary ring, then the entries in the *Linked Tables* containing the old pivot keys need to be updated. Each tuple in the local databases stores also the pivot levels of the secondary dimensions. Thus, the node responsible for the new pivot key finds the pivot keys of the secondary dimensions from its tuples along with their leaf values and informs them about its new pivot key, so as to update their links towards the primary dimension. When the roll-up is performed in a secondary ring, then the pivot levels stored in the tuples containing the new pivot key need to be updated as well. For this reason, the node responsible for the new tree, sends its pivot level along all of its different links towards the primary dimension.

In Figure 4.9, the outcome of a roll-up operation towards ℓ_0 in the secondary ring for dim_1 is shown. The involved tree structures storing the value b_0 before the roll-up are shown in Figure 4.4. It can be assumed that an indexed query for the value b_0 triggered a roll-up operation resulting in the re-insertion of the trees with pivot keys b_1 and b'_1 respectively. The node responsible for the new pivot key b_0 informs also the node responsible for the primary key a_2 that the new pivot level for all the tuples containing the value b_0 in dim_1 is ℓ_0.

Drill-down: The drill-down procedure is less complex, due to the fact that only one node holds the unique tree with values for this level. Thus, the node answering the query can locally decide if the drill-down is needed and proceed to the required actions. Afterwards, it splits the tree to tuples grouped by the new pivot keys and re-inserts them in *LinkedPeers*. In case that the queried value belongs to the primary dimension, the tuples of the local database are transfered to the nodes responsible for the new pivot keys as well. After the re-insertion of the relative trees is completed, the node responsible for the old pivot key informs also the root key about the new pivot keys and the new locations of the trees and all existing indices towards these trees are erased. Finally, the node that decided the drill-down updates the links among itself and the rest of the rings as described for the roll-up procedure.

Figure 4.10 exhibits the placement of the tuples in the primary dimension after a drill-down towards ℓ_3 of the tree with pivot key a_2 shown in Figure 4.4. A flood query for the value a_3 will end up in the node responsible the pivot key a_2 and will trigger the procedure for deciding if a drill-down is needed towards ℓ_3. If this is the case, the node re-inserts the tuples of the specific tree with the new pivot keys a_3, a'_3 and a''_3 respectively. The specific node informs also the root key a_0 about the new pivot keys and the pivot keys of the secondary dimensions that is linked to. For example, the entry $b_2 \rightarrow a_2$ in the Linked Table of the pivot key b_1 is now replaced with the entry $b_2 \rightarrow a_3$.

Group-Drill-down: When a roll-up is examined towards a queried level above the pivot level, it is also examined if a drill-down of the involved trees to a level $l_{ij} \geq MaxPivotLevel$ is needed.

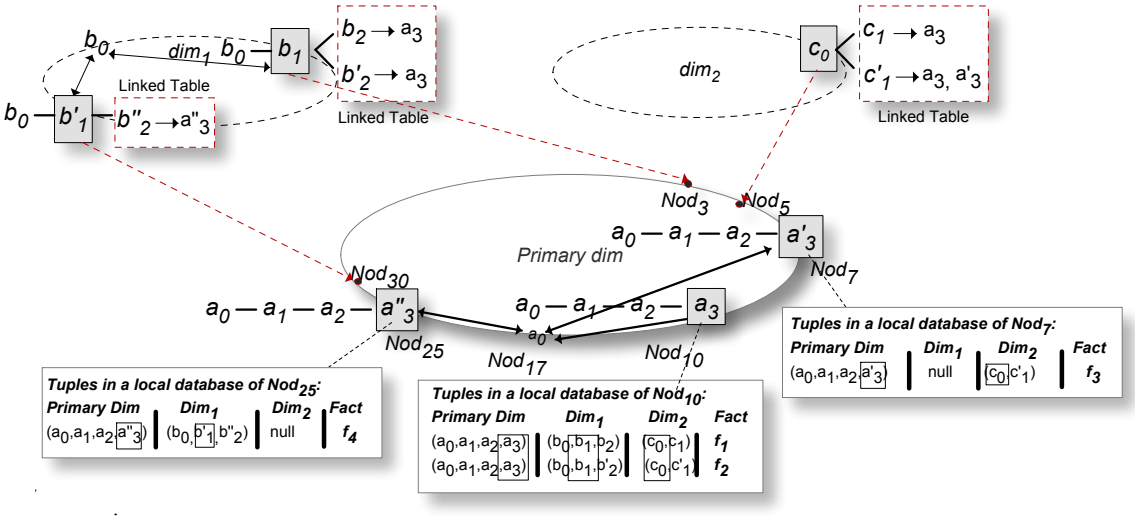


Figure 4.10: *Re-organization of the tree structures and the local databases in the primary dimension after a drill-down operation towards* ℓ_3

It is possible to find a level $\ell_{ij} \geq MaxPivotLevel$ to be the most popular but this tendency not to appear in the partial views of the involved nodes and for this reason the trees have not already performed a drill-down towards this level. In this case, the coordinator informs the involved nodes that a drill-down to this level is needed. This procedure is called `Group-Drill-down`, since more than one nodes participate in the drill-down. All the trees with the queried value drill-down to the new pivot level. If the new pivot level is equal to the MaxPivotLevel, the trees already in the MaxPivotLevel do not perform any action.

4.6 Experimental Results

4.6.1 Simulation Setup

A comprehensive evaluation of *LinkedPeers* is presented. The performance results are based on a heavily modified version of the FreePastry [Fre] using its simulator for the network overlay, although any DHT implementation could be used as a substrate. The network size is 256 nodes, all of which are randomly chosen to initiate queries.

The synthetic data are trees (different per dimension) with each value having a single parent and a constant number of *mul* children. The tuples of the fact table to be stored are created from combinations of the leaf values of each dimension tree plus a randomly generated numerical fact. By default, the used data comprise of 1M tuples, organized in a 4-dimensional, 3-level hierarchy. The number of distinct values of the top level is base $= 100$ with *mul*=10. The level of insertion is, by default, ℓ_1 in all dimensions. For the query workloads, a 3-step approach is followed: At first,

the part of the initial database (i.e., tuple) the query will target (*TupleDist*) is identified. Next, the probability of a dimension *d not* being included (i.e., a '*' in the respective query) is P_{d*}. Finally, for included dimensions, the level is chosen that the query will target according to the *levelDist* distribution. In the presented experiments, a different bias is expressed using the uniform, 80/20 and 90/10 distributions for *TupleDist* and *levelDist*, while P_{d*} increases gradually from 0.1 for the primary dimension to 0.8 for the last utilized dimension. Generated queries arrive at an average rate of $1\frac{query}{time\_unit}$, in a 50k time units total simulation time.

This section is intended to demonstrate the performance of the system for different types of inserted data and query workloads. The experimental results focus on the achieved *precision* (i.e., the percentage of queries which are answered without being flooded) and cost in terms of messages per query.

4.6.2 Performance Under Different Number of Dimensions and Levels

In the first set of experiments, the behavior of the system under data workloads containing tuples with various number of dimensions or tuples with variable number of levels per dimension is identified. The queries target uniformly any tuple of the dataset and any level of the hierarchies in each dimension.

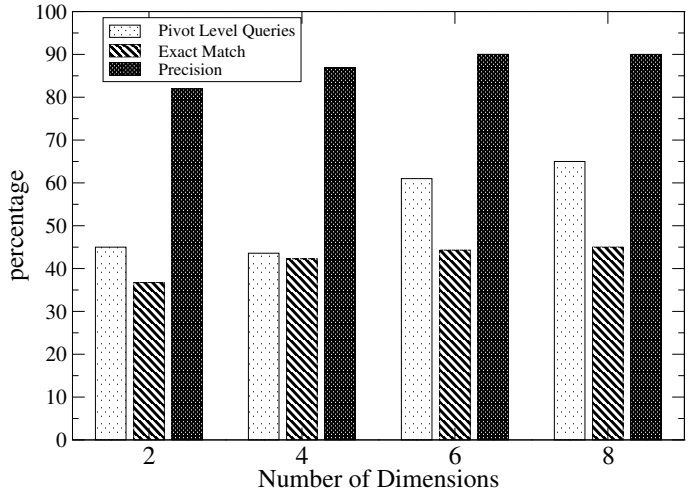


Figure 4.11: *Resolution of queries for data workloads with different number of dimensions, while each dimension is annotated with a* 3-level *hierarchy*

In the first set of the experiments, the number of dimensions varies, while each dimension is further annotated by a 3-level concept hierarchy. Figure 4.11 demonstrates the percentage of the queries in the workload including at least one pivot value (denoted as `Pivot Level Queries`), the percentage of the queries resolved as exact match queries in *LinkedPeers* (denoted as `Exact Match`) and the achieved precision. The precision for non-flooded queries remains above 85% for

all types of datasets, despite the number of dimensions. Queries that are not directed towards the pivot level are answered with the use of an index or a materialized combination assuring that the precision remains high. The difference among the exact matches and the pivot level queries is due to the fact that in the utilized strategy followed during the resolution of queries, it is preferred to use an index of a higher dimension than continue looking up for a pivot value in the dimensions with lower priorities.

In Figure 4.12, the results for *4-dimensional* workloads with varying number of levels in the hierarchies are demonstrated. The decrease in the precision (from 99% to about 70%) is due to the fact that the increase of levels has a negative impact on the probability of a query to include one pivot value for at least one of the dimensions. Thus, the percentage of the `Pivot Level Queries` decreases and consequently the same happens to the percentage of the `Exact Match` queries, as shown in the first pairs of columns in Figure 4.12. The increase of levels also influences the querying of a value that it is already indexed. For this reason, the deviation between the `Pivot Level Queries` and the `Exact Match` is bigger for two levels, where all queries including a value in the primary dimension are resolved with the use of the pivot key or the root index according to the proposed strategy for the query processing.

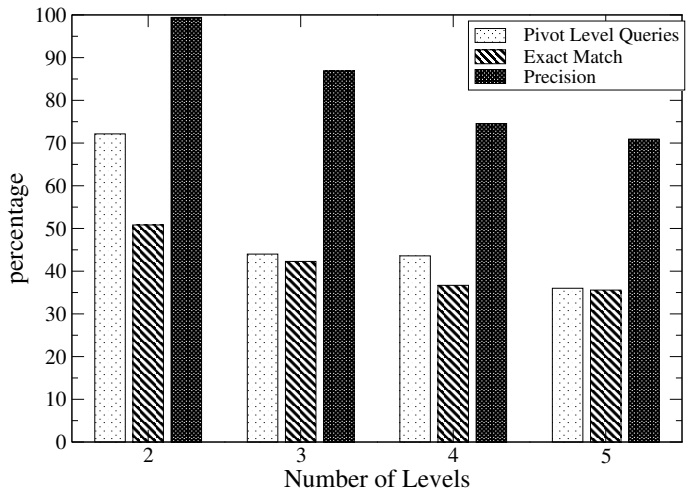


Figure 4.12: *Resolution of queries for data workloads with different number of levels in* 4-dimensional *datasets*

4.6.3 Query Resolution for Different Types of Datasets

In this experiment, the achieved precision of *LinkedPeers* for various types of datasets is demonstrated in Figure 4.13. The number of distinct values in the top level `base` and the number of children `mul` varies resulting in the change of the density for the dataset. `Base` and `mul` influence the connections among primary and secondary rings, the number of distinct values in each level

and in general the dataset density. As shown in Figure 4.13, as `mul` increases, a decrease in the precision is observed. The same trend is also shown for workloads generated with the same value for `mul` parameter, but with different values for the `base`. Nevertheless, *LinkedPeers* achieves to resolve the majority of queries without flooding.

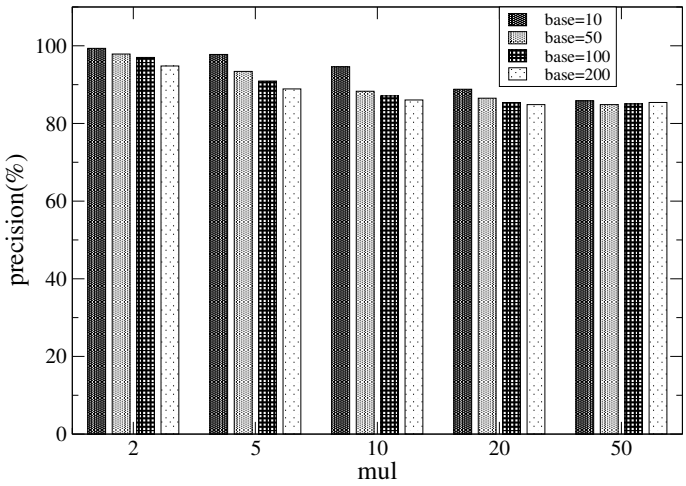


Figure 4.13: *Impact of `mul` and `base` in the achieved precision*

The percentage of exact match queries in the primary dimension (`Exact_PR`) remains stable for all datasets as shown in Figure 4.14, since it depends on the query workload. Nevertheless, the exact matches in the secondary rings (`Exact_SR`) increase as the indexed queries decrease, since the indices of the primary dimension are used less, and more queries are resolved by the secondary rings.

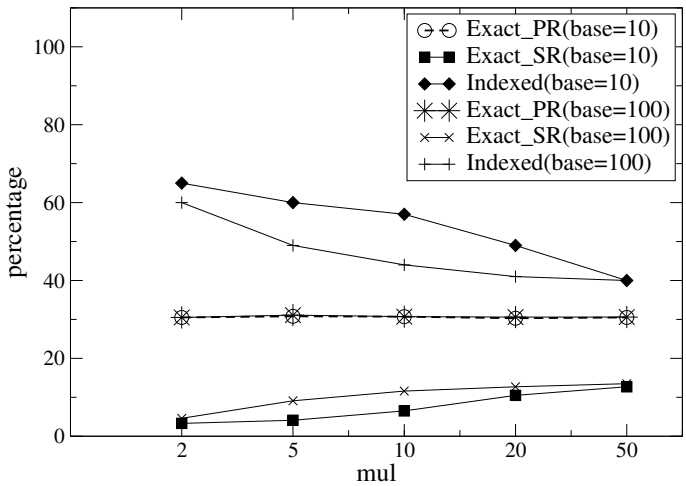


Figure 4.14: *Percentage of each query category for different data workloads*

4.6.4 Precision for Skewed Workloads

The adaptive behavior of *LinkedPeers* is identified in this set of experiments by testing the system under a variety of query loads. In more detail, the first set of experiments is about query loads biased towards the higher levels of the hierarchies, while different values of *TupleDist* are utilized for their generation. The number of queries directed to each level depends on the value of *levelDist*. The results of these experiments are shown in Figure 4.15 focusing on the achieved precision and the percentage of `Exact Match` queries. In this Figure, it is also depicted the percentage of queries including at least one value belonging to ℓ_0 and at least one value belonging to ℓ_1, which are denoted as `Queries_L0` and `Queries_L1` respectively. As shown in the Figure, the more biased the query load towards the higher levels, the higher the precision becomes and remarkably results (close to 100%) are observed. In the biased loads, the percentage of `Queries_L0` is significant bigger that the one of `Queries_L1` and it is easier for the system to decide the required roll-up operations towards ℓ_0 in each dimension. Thus, even if the selection of ℓ_1 is not appropriate for the resolution of queries without flooding, the system manages to adjust the pivot levels of the queried hierarchies and resolve a large portion of the queries as exact matches (denoted as `Exact Match`), proving that the re-indexing mechanisms are highly effective.

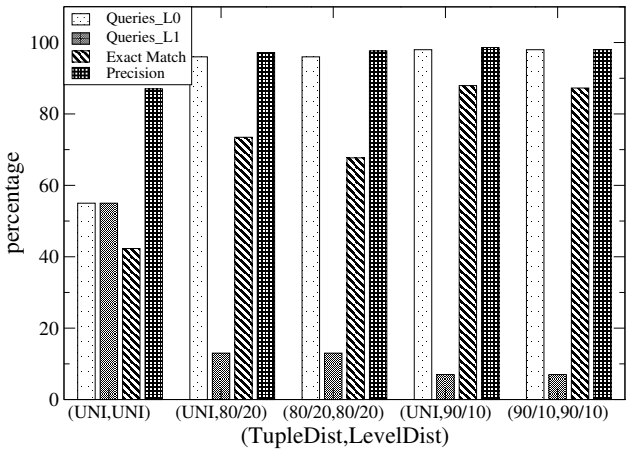


Figure 4.15: *Precision and exact match queries for skew towards higher levels and various* (TupleDist,levelDist) *combinations*

Figure 4.16 depicts the respective measurements, when the query loads favour the lower levels of the hierarchies. A decrease is noticed in the precision of loads, where the *levelDist* becomes more biased for the same *TupleDist*. This is due to the fact that lower levels of the hierarchy have a considerably larger number of values. As the number of queries targeting the lower levels increases, the probability of queries targeting non-indexed values is higher until the re-indexing mechanisms adapt the pivot levels of the popular trees appropriately. Since the percentage of

`Queries_L1` and `Queries_L2` does not indicate clearly which level is a a more appropriate selection as pivot level,the percentage of exact matches is lower in the biased loads compared to the achieved one in the first set of experiments (see Figure 4.15).

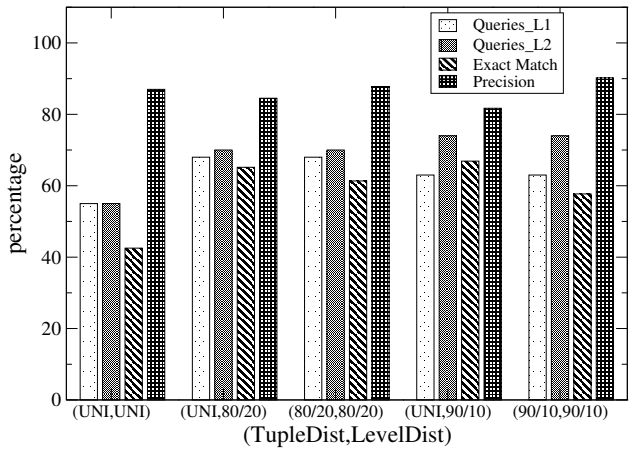


Figure 4.16: *Precision and exact match queries for skew towards lower levels and various* (TupleDist,levelDist) *combinations*

4.6.5 Testing against Partial Materialization

Apart from the re-indexing operations, the materialized combinations can be also utilized to minimize the query cost. In the next experiment, the method is tested against query workloads targeting the dataset either uniformly or biased (90/10) (*TupleDist*) with uniform and biased (90/10) skew (*levelDist*) towards the higher levels (denoted as `UP`) and towards the lower levels (`DOWN`).

As shown in Figure 4.17, the percentage of queries resolved with the utilization of a precomputed combination (`ViewQ`) increase in the query loads with 90/10 as *TupleDist* compared to the percentage of the corresponding loads with UNI *TupleDist* for the same values of *levelDist*. This happens due to the following fact: if a part of the dataset is queried at most, then the probability of asking a calculated combination of values increases as well. Thus, more queries are resolved with the use of combinations. Moreover, the percentage of retrieving an answer from a stored combination is higher for uniform (`UNI`) *levelDist*. In this case, the re-indexing mechanisms cannot adjust the pivot levels to all the incoming queries and the indexed queries are more often resulting in the utilization of more identifiers.

Figure 4.18 depicts how queries are resolved during the simulation of the query load that targets mostly a specific part of the dataset and uniformly all the levels of the hierarchy in Figure 4.17. The total number of queries has increased to 100k and the percentage of queries answered with the use of a materialized combination is not included in the percentage of the indexed queries. It

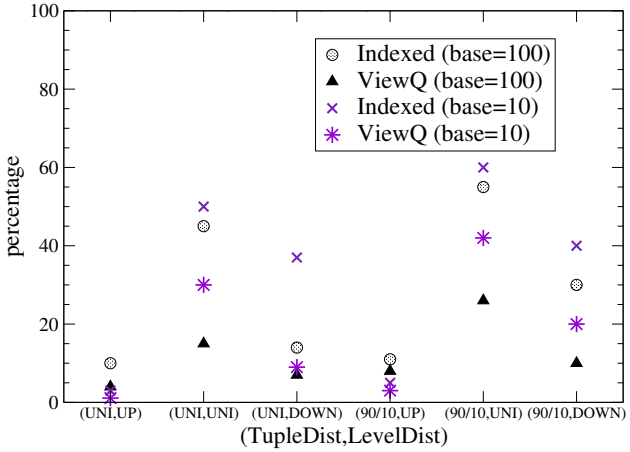


Figure 4.17: *Utilization of materialized combinations compared to queries resolved as indexed*

can be observed that the utilization of view identifiers increases over time and less queries needs to be forwarded across the indices.

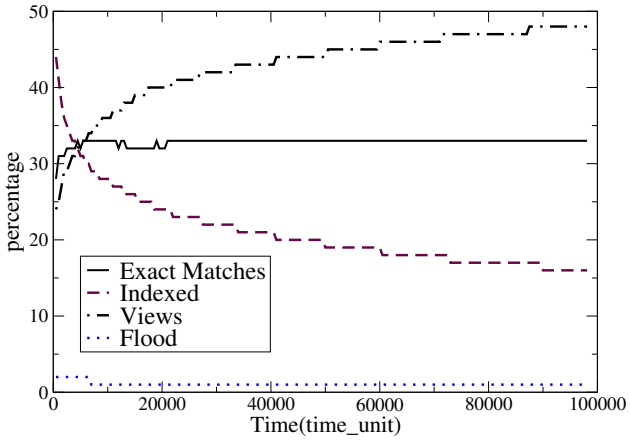


Figure 4.18: *Resolution of queries emphasizing on queries resolved with the utilization of materialized combinations compared to forwarded indexed queries over time*

4.6.6 Cost of the Various Types of Query Resolution

The cost of a query is considered as the messages that need to be issued for its resolution. A query resolved as exact match in the primary dimension utilizes only the DHT lookup mechanism. Figure 4.19 depicts the average number of messages only for exact queries resolved by secondary dimensions (`Exact_SR`), which number is significantly smaller (less than 20% of all queries in all cases) and indexed queries (`Indexed`). The average number of messages for `Exact_SR`

depends on the type of dataset, namely the number of links among secondary pivot keys and primary pivot keys. When the query workload is skewed towards the higher levels (UP), then the messages decrease due to the fact that popular trees roll-up towards ℓ_0. Thus, the secondary keys are connected to a smaller number of primary keys. The opposite observation is valid for the (DOWN) query workloads.

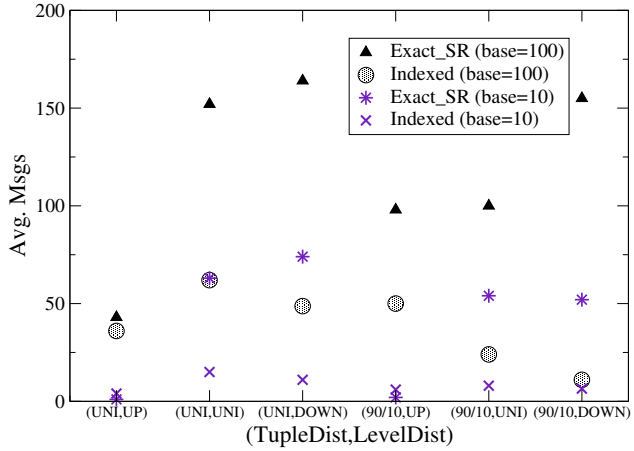


Figure 4.19: *Average number of messages for exact matches in secondary rings and indexed queries*

4.6.7 Performance for Dataset of the APB benchmark

The adaptiveness of the system is also tested using some realistic data. For this reason, we generated query sets with the APB-1 benchmark [apb]. APB-1 creates a database structure with multiple dimensions and generates a set of business operations reflecting basic functionality of OLAP applications. The generated data are described by 4-dimensions. The customer dimension (C) is 100 times the number of members in the channel dimension and comprises of 2 levels. The channel dimension (Ch) has one level and 10 members. The product (P) dimension is a steep hierarchy with 6 levels and 10.000 members. Finally, the time dimension (T) is described by a 3-level dimension and made up of two years. The dataset is sparse (0.1 density) and comprises of 1.3M tuples.

Figure 4.20 shows the percentage of exact match queries resolved in primary and secondary rings compared to all exact match queries of a 25K query workload and for different combinations of ordering of dimensions. For all combinations, the precision of non-flooded queries is over 98%. The selection of the primary dimension influences the number of exact match queries in the primary ring.

Figure 4.21 presents the average number of messages for exact matches resolved by a secondary ring and indexed queries, since only a DHT lookup is performed for exact match queries in the primary ring. The average number of messages is small for both exact and indexed queries,

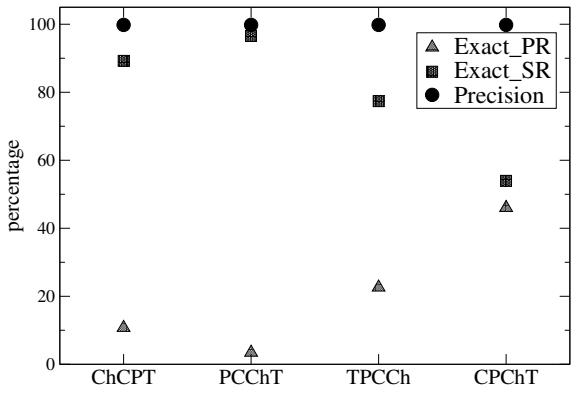


Figure 4.20: *Precision for APB query workload in LinkedPeers*

apart from the case that the customer dimension has been selected as a primary dimension. In the rest of the cases, the resolution of the queries occurs with a very low cost in terms of additional nodes to visit, even though the majority of the exact queries is resolved by a secondary dimension, as shown in Figure 4.20. The increase of messages for the CPChT dataset is due to the large number of distinct values used as pivot keys and thus each node responsible for a pivot key stores smaller portion of the total dataset in its local database. For all combinations of datasets, the overhead of the additional indexing structures needed by *LinkedPeers* such as tree structures, root indices, links and indices and statistical information is up to 1%. Thus, *LinkedPeers* can be considered as a lightweight solution for indexing multidimensional hierarchical data.

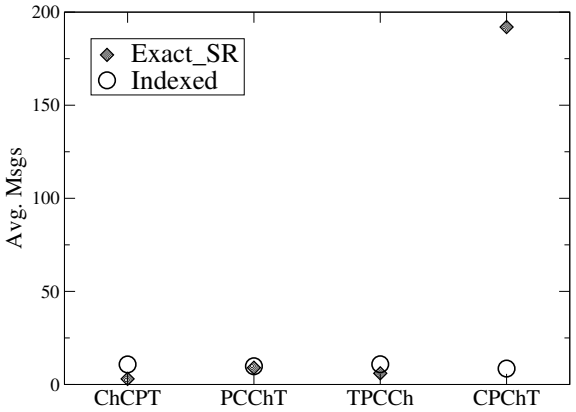


Figure 4.21: *Average number of messages for exact match and indexed queries*

4.7 The Linked Data Paradigm

The advent of the Semantic Web [Sem] has brought forth to the notion of the "Web of Data", i.e., the integration and interlinking of diverse data and their descriptions in order to enable people to create data stores on the Web, build vocabularies, and write rules for handling them. Web-scale data management imposes some interesting challenges, which are mainly arisen from the big size of published data and the lack of a well-defined schema followed by all resources. The latest factor is an obstacle for the re-usability of data, due to the fact that regular and well-defined structures can be handled and reused with greater success from created tools. To overcome the lack of a general structure, different APIs are developed for each source that publishes data. Nevertheless, the complexity of this solution prevents Web applications of integrating datasets coming from different resources. Yet, it is the efficient *linking* and processing of data published by different sources that usually provides with interesting insights.

The above assumptions can be met in the recent Linked Data initiative [Data], [BHBL09], where data from different resources and domains are interlinked providing large scale integration of, and reasoning on, data of the Web [Datb]. The term *Linked Data* refers to a set of best practices for publishing and interlinking structured data on the Web. The "Linked Data principles" [BHBL09] define the following basic rules of publishing data on the Web so as to become part of a single global data space:

- Use URIs as the names for things
- Use HTTP URIs, so that people can look up those names.
- When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL).
- Include links to other URIs, so that they can discover more things.

The linking of data published by various resources across the web presumes the existence of mechanisms that follow the above principles, but also indicates the existence and significance of connections between items described by this data. The Resource Description Framework (RDF) [RDF] has been widely introduced for the representation and exchange of such information, since it provides a flexible way to describe things in the world (e.g. people, locations or abstract concepts) and their relationships to other things. RDF data is, in essence, a collection of statements represented as triples: $\langle$*subject, property, object*$\rangle$. Each triple states the relation between the subject and the object. The current trend for resources publishing such data is to make them available through SPARQL endpoints [SPA].

The evaluation of queries in such environments mainly occurs along two directions: Centralized systems (e.g., data warehouses or materialization-based approaches) and federation of

resources. The typical approach in existing systems is to collect large dumps of data (possible through crawling), to preprocess and to load these dumps in a centralized storage point so as to allow querying of the merged data locally. For example, the DBpedia project [BLK+09] is an effort towards this direction trying to extract structured information from Wikipedia, link this information to other existing resources and make it available on the Web. Although centralized solutions are advantageous during query processing by providing access to the whole dataset, these solutions are vulnerable to the growth of the data size ([HMZ10], [HHK+10]). Moreover, the pre-processing phase may be time-consuming preventing the leverage of the benefits of Linked Data. Another problem is the synchronization of local copies in the centralized repositories, especially when the data sources change frequently or the RDF instances are created on-the-fly.

An alternative approach is the federation of existing endpoints and the implementation of mechanisms for distributed query processing over a federation of RDF resources [GS11]. During distributed processing over federated resources, the query is split into subqueries, the resources are queried to determine the ones with relative data, the query is evaluated against the found sources directly and finally the retrieved results are merged in the federator. DARQ [QL08] provides an engine for SPARQL queries among different SPARQL endpoints. It uses service descriptions for the provided data to acquire information to be used for the optimization of the selection of the sources during the querying planning. In [HHK+10], the authors aims at providing a solution that combines features from completely centralized and distributed approaches. At that end, they build a Q-tree based indexing structure to store descriptions of the content of the data sources and use this index to determine relevant sources with high probability. FedX [SHH+11] provides a framework for executing SPARQL queries over different endpoints. Trying to minimize the forwarding of queries to sources with no relevant content, it uses SPARQL ASK queries returning *yes* if a query pattern has at least one solution. Also, special care is given for determining the join order to minimize the size of intermediate results.

The federation of different endpoints results in additional overhead since the query mechanisms do not have direct access to the whole repository and additional information (i.e. statistics) to optimize query planning. The lack of common schemas among the different endpoints also adds more complexity during the query processing, especially when the number of web resources increases ([HMZ10], [HHK+10]). The distributed indexing scheme proposed in this thesis can provide a high performance substrate for implementing advanced techniques for query planning similar to the ones proposed in [QL08], [SHH+11]. As a result, it can speed up the procedure before contacting the actual sources of the data and tackle performance and integration issues of Linked Data.

4.8 PI4LD: A P2P Indexing Scheme 'FOR' Linked Data

In this thesis, a system for *Peer-to-peer Indexing FOR Linked Data* -called *PI4LD-* is designed for the efficient distribution of data published according to the principles of the Linked Data, namely data which are self-describing and contain semantic information. Since RDF provides a generic, abstract data model for the description of resources, then taxonomies, vocabularies and ontologies can be used to express more domain-specific information. Apart from the assumed RDF Data Model, PI4LD supports a more sophisticated data model combining schema languages as well, such as the RDF Vocabulary Definition Language (RDFS) and the Web Ontology Language (OWL), allowing the representation of tightly structured or semi-structured data.

To overcome the limitations met in centralized and federated approaches, a decentralized approach to index linked data for efficient processing of queries is presented. The described system can be utilized either as a fully autonomous infrastructure to load, store and query large volumes of RDF dumps or as a federation overlay for indexing the linked data on top of existing resources and their endpoints. To achieve this goal, a DHT overlay is utilized to maintain summaries of the entities and their relationships in a hierarchy preserving manner, while the stored data are distributed across the participating nodes. The methods developed in the context of *LinkedPeers* are customized and adapted to provide the appropriate functionalities to serve the current requirements. The overlay in effect comprises of two 'virtual' DHT rings, one for *subject* and one for the *object* value, with *property* being encoded in the indexing scheme. The indexing scheme of PI4LD ensures that the least number of relevant nodes will be promptly identified, so as to reduce the join operations.

In brief, the PI4LD system accomplishes to meet the following requirements:

- A complete solution for distributing, storing and indexing linked data among commodity nodes. The system is also able to perform online incremental updates and maintain data in a fault-tolerant and decentralized manner. There are no requirements for the data to follow a predefined, strict schema and for intensive processing during the creation and update of indices.
- The presented approach offers a transparent layer to the end-user, who can query the stored data in a unified manner. Queries at different level of concepts and entities can be performed effectively due to the organization of data in hierarchical structures. The adaptation of the indexing granularity among different levels of entities and classes is performed according to the incoming queries.
- A thorough performance evaluation proves that the system is appropriate for accommodating different types of data published by different resources across the Web and can be considered as a viable solution compared to other centralized ones.

4.9 Data Model in PI4LD

From the advent of the LOD paradigm, most of the data providers have exploited the RDF model for publishing their data. An important feature introduced in these efforts for publishing structured information over the Web is the adoption of models based on RDFS and the Web Ontology Language OWL to represent the semantic knowledge, i.e., to express entities, facts, relations between facts and properties of relations. For example, the DBpedia community uses a cross-domain, manually created ontology extracted from the infoboxes within Wikipedia. All objects (e.g. countries, cities, organizations, people) can be considered as entities, while each entity is an instance of at least one class defined in the ontology [SKW07]. Classes are also considered as entities. The property `rdf:type` is used to state that a a resource (all things described by RDF are called *resources*) is an instance (or individual) of a class.

The use of ontologies implies a hierarchical organization of the data, since an ontology represents knowledge as a set of concepts within a domain and these concepts can be usually arranged in a hierarchical manner. The arrangement of concepts of an ontology in a taxonomy (or category hierarchy) contributes to their use and reasoning [SKW07], [JHS+10]. In the studied cases, the classes of the ontology are arranged in hierarchies and the `rdfs:subClassOf` relation is used to state that one class is a subclass of another. In the proposed system, the hierarchical structure is exploited, which is extracted from such data, and it is used to distribute them among the nodes of the system in a way that preserves the hierarchy semantics and enables more efficient query resolution. Thus, the data model introduced in this system organizes entities in trees according to the `rdfs:subClassOf` relationships among the classes that the entities are instances of. All references to instances are denoted as *entities* from now on and it assumed that an entity can appear either as a subject or an object of the triple.

Figure 4.22 shows some triples describing different instances and the relationships that link them, as they were extracted from the DBpedia dataset. These triples are organized in different tree structures according to their common values. On the logical level, the grey nodes representing the *schema* layer are distinguished from the actual instance layer depicted by the white nodes. The solid arrows accrued from the `rdf:type` property and the relationship defined in the used ontology (the labels are omitted for these edges for readability reasons) while the coloured, dotted edges visualize relationships among entities and these properties are denoted by the label of each edge. All edges are directed from the subject of the triple towards the object. For example, the types of the entity about `Athens` and its relationship to the entity `NTUA` are described by the following triples[1]:

[1]The URIs are omitted for readability reasons

```
Athens rdf:type Place
Athens rdf:type Populated Place
Athens rdf:type Administrative Region
NTUA dbpedia:campus Athens
```

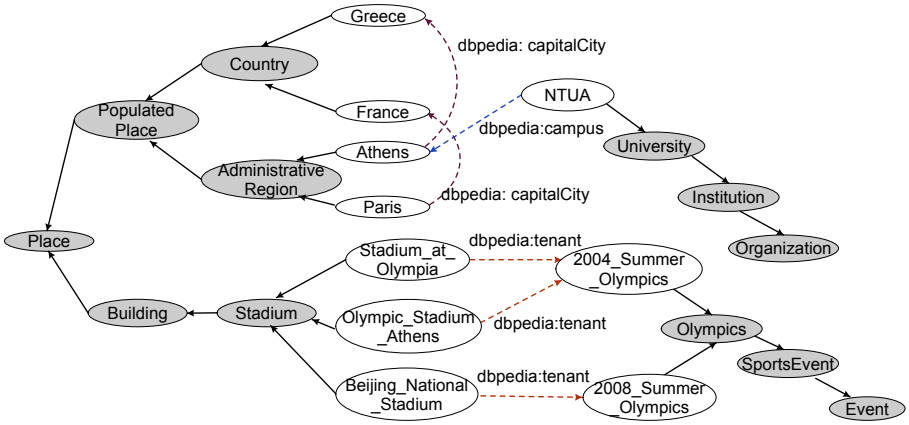


Figure 4.22: *A representation of some DBpedia instances*

4.9.1 Notation and Definitions

In the proposed system, the data items are inserted in the form of triples as described above. A requirement posed by the architecture of the system is that a new entity is inserted in PI4LD along with all the triples defining its types so as the "*full path*" of the hierarchy of classes describing the specific entity to be defined. For example, the insertion of the triple ⟨*NTUA, campus, Athens*⟩ requires that all triples from the most generic concept to the most detailed one are given for both entities of the triple. This approach has been followed to provide an extensible and flexible system, without requiring the existing ontologies to be populated among all the nodes of the system.

The entities are organized along hierarchies with L_i levels of aggregation ℓ_{ij}, where i denotes the *i-th* ontology and j ($j \in [0, L_i - 1]$) represents the *j-th* level of the *i-th* ontology. Level ℓ_{i0} is called the *root level* for the *i-th* ontology and its hashed value is called *root key* and corresponds to the most basic concept in a domain. Although every individual in the OWL world is a member of the class `owl:Thing` and thus all entities should have `Thing` as a root value, this level is omitted in the hierarchies and the entities of the next level are considered as the root values.

It is defined that ℓ_{ik} lies *higher* (*lower*) than ℓ_{il} and denote it as $\ell_{ik} < \ell_{il}$ ($\ell_{ik} > \ell_{il}$) *iff* $k < l$ ($k > l$), i.e., if ℓ_{ik} corresponds to a less (more) detailed level than ℓ_{il} (e.g., *PopulatedPlace* < *Country*).

The values of the hierarchy levels are organized in tree structures, one per root key. Without loss of generality, it is assumed that each value of ℓ_{ij} has at most one parent in $\ell_{i(j-1)}$. Note also that the classes of the ontology form a subsumption hierarchy, where the number of levels in each branch of the hierarchy is not constant. In the proposed scheme, it is a prerequisite that all the branches of the same root value have the same number of levels so as the re-indexing operations to be enabled in the same context described in Section 3.5. Since in a real-world ontology, there may be a different number of levels below each node, the missing levels are filled above the leaf value with "pseudo" values.

4.9.2 Organization of Data

The proposed architecture for a fully distributed system that accommodates linked data is based on the fundamental primitives of DHTs. In PI4LD, scalable mechanisms are developed for the distribution, indexing and querying of linked data in a fully decentralized manner.

The distribution of data among the nodes of the overlay occurs in a manner that preserves the ontology-specific information of each entity, while it also interlinks different entities according to their properties. The goal in the design of the architecture is that each entity is stored in a fully-descriptive manner. Thus, the node responsible for an entity stores also all the classes (or concepts) that are related to the specific entity. According to the DHT protocol, a *key* needs to be mapped to each entity inserted in the system, so as the entity to end up to the node responsible for its key. It is chosen to generate this key by hashing the value of a level of the hierarchy. This value is referred as *pivot value*, the generated hashed value as *pivot key* and its level as *pivot level*. The highest and lowest pivot levels of each hierarchy for a specific root key are called *MinPivotLevel* and *MaxPivotLevel* respectively. Since multiple entities can be described by the same order of classes (or concepts) defined in the followed ontology, each node organizes all the relative entities in tree structures characterized by a common classes (or concepts) (or entity) in the root level and the same pivot value.

The resolution of queries searching for linked entities requires that this information is also stored in the overlay. For example, the query to retrieve the names of the Stadiums that hosted Olympic Games is:

```
SELECT ?Stadium ?Event
WHERE { ?Stadium dbpedia:tenant ?Event.
?Stadium rdf:type  Stadium. ?Event rdf:type  Olympics.}
```

To achieve the resolution of this query, it is not only needed to find which entities are members of the classes `Stadium` and `Olympics`, but also which of these entities are linked with the property `dbpedia:tenant`. For this purpose, it has been decided to store a *tuple* containing both semantic and property related information in a local database of the node responsible for the entity appearing as a subject. In the above example, such a tuple about two of the existing entities of Figure 4.22 answering the above query would look like:

```
⟨Place, Building, Stadium, Stadium_of_Olympia, Event, SportsEvent, Olympics,
2004_Summer_Olympics,dbpedia:tenant⟩
```

where the last value is the property connecting the two entities. The idea is to maintain a *primary virtual ring* storing all the information about the triples and a *secondary virtual ring* indexing the entities appearing as objects. The pivot keys corresponding to the primary ring are called *primary keys* and the tuples are stored in a local database in the primary ring. Due to the fact that most of the values appearing above the lowest level are common, this property can be exploited for the design of more efficient schemas than using a simple table with multiple columns. Another factor to take into account is that the semantic information describing the entity of the subject can be also found in the tree structure maintained by the node. Nevertheless, the organization of the local database is beyond the scope of the described architecture and it is not further analyzed.

Another remark to be made is that virtual overlays index only the subjects and the objects. Thus, the queries containing only properties can be resolved only if these queries are forwarded and evaluated among all the nodes of the system. The creation of indices for the properties is omitted due to the fact that properties usually do not adhere to hierarchical relationships. If it was decided to index them, then their values would be simply hashed and inserted in the DHT according to the procedures followed for the insertions in a secondary dimension. It has been also observed that it can be a common occurrence in linked data queries, that at least the type of the subject or the object is defined [Hal09]. If this is the case, the query resolution can start from this value and follow the links maintained in the system until it is resolved. Nevertheless, if the system is about to serve mostly queries only about properties, then an additional secondary ring can be added to the system as the one for indexing the objects without requiring further modifications.

During data insertions, the information about the pivot value is vital (only for initial insertions the pivot level can be selected according to the needs of the application). The design of the system requires that if a value has been selected as a pivot key during the insertion of an entity, this pivot key should also be selected as pivot key for every other entity even in the case that it is related to it with the property `rdf:type`. To comply with this assumption, a node should be aware of the existing pivot keys during the insertion of a new tuple. Thus, a fully decentralized catalogue

storing information about root keys and their respective pivot keys in the network is implemented in PI4LD. Each root key is stored at the node with ID closest to its value. Every time that a new pivot key corresponding to this root key is inserted in the system, the root key node is informed about it and adds it in a list of known pivot keys. The root key node is also aware of the MaxPivotLevel used during the insertion of its values in the specific dimension.

Initially, the node performing the insertion of a tuple contacts the node responsible for the root key of the subtuple in the primary dimension. The root key of the primary dimension is informed about the new entity and indicates the appropriate pivot level (if the same pivot key already exists, then its pivot level is used, otherwise the MaxPivotLevel). Afterwards, the DHT operation for the insertion of the entity in the primary dimension starts and the tuple ends up to the node responsible for the decided pivot key. The node responsible for the pivot key of the primary dimension stores the entity in the respective tree structure and the whole tuple in its *local database*.

The next step is to store the entity appearing as an object in the new triple. The node responsible for the primary key contacts the node responsible for the root key of this entity and is informed about the appropriate pivot level. After deciding of the pivot value for the specific entity, the entity is stored in the tree structure of the responsible node for its pivot key. The object is associated to the primary dimension through the primary key. Each entity of a secondary tree structure maintains a list of the primary keys that is linked to. The structure storing the mappings among the leaf values and the primary keys is referred to as *Linked Table*. All entities inserted in the overlay with the same pivot key end up in the same node determined by their pivot key. The only thing that differentiates a subject from an object is that in the case of the subject the whole triple can be found in the local database while only a link towards the primary pivot key exists in the opposite case. Given the importance of `owl:sameAs` property for querying linked data from different resources, the links towards the primary ring characterized by this property can be marked. During updates, the same procedure is followed for the insertion of the new triples. The additional requirement is to update any indices that may exist for the instances of the classes that the new entity adheres and lay above the pivot level, if its pivot key does not already exist in the overlay. Thus a lookup is issued and any related information is updated (see Section 3.5.1).

In the example of Figure 4.23, the selected pivot level is ℓ_1 and all the triples of Figure 4.22 related directly or not to the property `dbpedia:tenant` are shown. The root keys of the primary and secondary dimension for these tuples are `Place` and `Event` respectively. The tree structure in Nod_1 stores all the entities appearing as a subject in the triples with the property `dbpedia:tenant`, while all the objects have formed the tree structure shown in Nod_{10}. The entities of the lowest level also have links towards the pivot key of the primary dimension. If a new entity of type `Populated Place` is inserted in the overlay, then a new tree structure while be formed in another node of the overlay and the root key for `Place` will show to this node as well. In case that a new

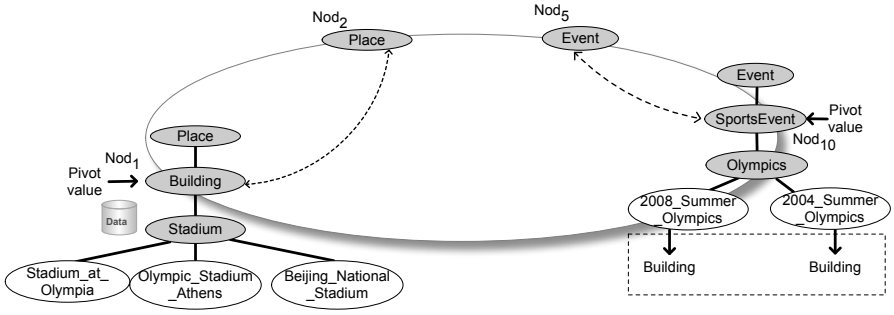


Figure 4.23: *Distribution of linked data in PI4LD*

triple arrives containing the entity `2012_Summer_Olympics` as a subject, then it will also end up in Nod_{10} and added in the existing tree (if it does not already exist). The only difference is that a link towards the primary key will not be kept, but the respective tuple will be added in the local database.

4.10 Querying Linked Data

The proposed system extends the DHT lookup functionality to provide a distributed mechanism able to allocate any value of the stored triples. Since the system accommodates RDF triples, it is expected that SPARQL will be used for querying the stored data. The basic type of SPARQL queries consists of two parts the `SELECT` clause identifying the variables to appear in the query results and the `WHERE` clause that provides the basic graph pattern to match against the data graph. The basic SPARQL query unit refers to triple patterns <*subject, property, object*> where variables might be used instead of specific values for each of the three components. Apart from the `SELECT` queries, the SPARQL standard [SPA] also defines other types of queries, such as `ASK`, `DESCRIBE` and `CONSTRUCT` queries. Since the presented architecture mainly focuses on aspects regarding the distribution of the data among multiple nodes and the efficient indexing to enable the resolution of queries in a fully distributed environment, `SELECT` statements are studied.

The most basic type of queries that can be posed to the system may contain either a single value or a combination of values to be searched. The basic type of query that is natively supported by the proposed system is of the following form:

$$q = (?s, ?o, ?p, restrictions, measures)$$

The values of the variables s, p and o can either have a predefined value or acquire the value specified during the resolution of the query. The query may also include restrictions about the queried values and also measures can be specified for retrieval. Measures are referred to any other relative information and "attribute" like or datatype properties of the entities, which is not

indexed in the DHT but it is stored in the local database and can be retrieved. For example, if an entity is related to an abstract, then this abstract is only stored in the local database of the node storing its entity. For example, a query about the campus located in "Athens"

```
SELECT ?s
WHERE { ?s dbpedia:campus Athens.}
```

is a basic query, which can be expressed with the following internal form in the system:

```
(?s,"Athens","dbpedia:campus")
```

In general, the lookup primitive in a DHT overlay allows the retrieval of an item if its hashed value has been used as a *key* during its insertion. If an entity inserted as a pivot value is queried in the system, then the node responsible for this value can be found with a simple DHT lookup. Otherwise, the native DHT mechanisms are not adequate to locate this entity. Moreover, the proposed system allows adaptive changing of the pivot levels according to the incoming queries. Therefore, a node initiating a query is not aware if any of the queried values correspond to a pivot value. The strategy followed in the presented approach is that the node issues consecutive lookups for any defined value contained in the query starting from the subject value. If this lookup operation does not return any result, then a lookup for the object value starts. The idea behind this strategy is that the lookup operations are not expensive due to their logarithmic routing cost in the DHT overlays. Thus, issuing an additional lookup is preferable to querying all the nodes of the overlay. Nevertheless, DHT lookups are not adequate for the resolution of queries that do not contain any pivot value. For that reason, more advanced search functionalities have been developed which can be further utilized to enable the search of any stored entity. The basic techniques for searching entities in PI4LD are the following ones:

Exact Match queries: The queries concerning any pivot key which may appear either as a subject or an object belong to the category of `exact match` queries. If the queried entity corresponds to a pivot key, then a DHT lookup up ends in the node responsible for this pivot key and the corresponding tree structure is searched for retrieving relative triples. The tree structures are only used for resolving queries about a single entity and how it is related with the other entities in the hierarchy, e.g., a query about all stored buildings `(?s, Building,rdf:type)` is resolved directly with the information included in the corresponding tree structure. In all other cases, the query is evaluated with the use of the information stored in the local database. In case that the local database needs to be queried and the pivot key corresponds to the subject entity, then the local database of the reached node contains all relative information. Otherwise, the query needs to be forwarded across the links towards the related pivot keys found in the Linked Table, so that their local databases to be queried for the final resolution of the query.

Flood queries: Queries that do not contain any pivot keys cannot be resolved by the native DHT lookup. The only alternative is to circulate the query among all nodes and process it individually and return the found results back to the node that initiated the query. A simpler flood procedure is enforced compared to the one enforced in *LinkedPeers* and the query is forwarded from one node to its closest neighbour until all nodes of the overlay are visited. Each node receiving a flood query processes it locally by searching its tree structures and its local database. In more detail, if only a single entity is included in the query, then the information stored in the tree structures is adequate to answer it and the searching of the local database is avoided. In case that more than one values are queried, then the query is answered directly by the local database. The tree structures are only searched to find if any of the queried entities are contained so as to update the respective statistic records maintained for the specific tree.

Indexed queries: During the flood procedure, all the nodes of the overlay are contacted and thus the acquired knowledge can be further utilized. Towards the exploitation of this knowledge, the creation of soft-state indices is introduced. When a node answers a flood query, it checks if a re-indexing operation is needed (see Section 4.11). If the system decides that a re-indexing operation is not indicated, then the query initiator starts the procedure of creating soft-state indices for all the values contained in the query. Since a flood query is circulated among all nodes, the nodes responsible for the tree structures containing the queried entities can be found. The query initiator starts the procedure of creating indices by hashing the requested values and inserting them in the DHT along with nodes responsible for them. The next time that a query is issued for an indexed entity, the node holding the index is located and the query is directly forwarded to the node(s) responsible for the pivot key(s) of this value. If the indexed entity appears as an object and the local database needs to be queried, then the reached nodes return the locations of the primary ring that are correlated with the indexed entity. The query is forwarded to these nodes which contacts their local database for its final resolution.

As in all previous approaches, the nodes storing an indexed entity in their local database need to know the existence of an index. The bidirectionallity of the indices is introduced only to ensure data consistency, despite of them being soft-state. During re-indexing operations, the locations of stored tuples change and indices correlated to these tuples need either to be updated or erased, preventing the existence of stale indices. It is chosen to erase them, so as to avoid increasing the complexity of the system. Every lookup for an indexed value renews a TTL value in both sides of the index and only valid indices are erased during re-indexing operations.

According to the distribution of triples in Figure 4.23, the query (“Building”, “SportsEvent”,?p) is an exact match query to be resolved in the primary ring, when the node responsible for the pivot key `Building` is reached. On the other hand, the query (“Stadium”,“SportsEvent”,?p) is an exact-match query resolved by the secondary ring, which needs to be forwarded in the node responsible for `Building` so as to be evaluated in the local database. During the first time that the

query (“Stadium”,?p,?o) is posed to the system, it is flooded in the network. Nevertheless, the rest of the times this query is resolved as an indexed query as long as the soft-state index for `Stadium` is valid. Finally, it should be noticed that a query of this form

```
SELECT ?Stadium ?Event
WHERE { ?Stadium dbpedia:tenant ?Event.
?Stadium rdf:type  Building.
?Event rdf:type  Olympics.}
```

is represented and resolved in the system as the query(“Building”,“Olympics”,“dbpedia:tenant”), but it is also specified in the query that all the entities of these types need to be returned in the answer.

The queries posed to the system are translated into the internal representation described above. Although, any advanced techniques for join optimization has not been implemented yet, such as using statistics, histograms,etc, some features of the described indexing scheme can be exploited during the resolution of queries. According to the intrinsic characteristics of RDF and SPARQL,the resolution of two general classes of queries is analysed. One major class is queries containing star-shaped subqueries combining several properties of the same entity. In this case, consecutive lookups are issued for the searched entities until one of the defined entities is found either as a pivot key or as an indexed entity. When an entity is found, then the whole query is evaluated in the local database either directly (the found entity appears in the subject of the query) or after forwarding. Another major category is queries with path expressions, namely “*chains*” of triple patterns, where the object of the first pattern is the subject of the next pattern, again with given properties. If a triple includes the property `rdf:type`, this is equivalent to merge the specific triple with the following one, having to perform less joins in the retrieved results in the described scheme. The strategy followed during the evaluation is that the resolution starts with the subquery containing specific entities and retrieves the smallest number of results.

4.11 Query-driven Re-indexing

PI4LD maintains the ability to perceive the trends of queries described in *LinkedPeers* and the mechanisms for roll-up and drill-down operations are implemented. The procedure for deciding on the need of a roll-up or drill-down operation can be triggered after a flood or indexed query. In more detail, when a query is flooded, then all nodes of the overlay are contacted and search their tree structures to find out if a possible re-index operation to the level of any queried entity is indicated according to the information maintained by them. If more than one entities are included in a query, then it is possible multiple re-indexing operations to take place in different rings. In case that an indexed query is resolved by a node and the specific node has received

a predefined number of queries using an appropriate sliding window, then possible re-indexing for the queried level of the indexed entity is considered.

If a drill-down towards a level below the pivot level is required, then the node that detected this demand proceeds to the required actions, as described in Section 4.5. When the queried value belongs to a level above the pivot level, then each node that receives the flood query checks if a roll-up will be favourable according to its incoming queries and if this it the case, then includes this fact to its answer to the initiator of the query. When the initiator collects all the answers and discovers that at least one node asks for a roll-up, then it gathers information from all the nodes and decides if a roll-up or group-drill-down is needed, as it has been described in Section 3.5 and Section 4.5. A remark to be made is that in the case of PI4LD, the same pivot key may appear as a subject in some triples and as an object in some others. Thus, the update of the links among the primary and the secondary dimensions occurs in both directions, namely both the pivot levels for the secondary dimension in the tuples of the database and the records of the Linked Tables are updated.

4.12 Experimental Evaluation

4.12.1 General Setup

The proposed system is implemented on top of a heavily modified version of FreePastry [Fre], although any DHT implementation could be used as a substrate. The nodes are hosted by different physical machines that have a Xeon processor at 2GHz with 8GB of memory running a 64-bit Debian Linux kernel. Nodes are connected with Gigabit Ethernet and communicate through the FreePastry sockets. By default, experiments were conducted using a 16-node overlay.

The local databases for each node are setup with SQLite [SQL]. The schema used is a single table; a more sophisticated schema may also be used to improve the retrieval performance. Yet, the purpose is to showcase the advantages of a unified distributed indexing scheme for multiple RDF repositories. Further boosts due to improved performance of the local databases is outside the scope of the evaluation of the system.

The PI4LD approach is compared to a setup consisting of a central repository (denoted as `VirtStore` in the Figures) built with the use of *Virtuoso Open-Source Edition version 6.1* [Vir], which is a centralized triple store for RDF data. The specific store is a popular open-source solution for this purpose, while the Virtuoso software is also used to provide a back-end database engine for the public SPARQL endpoint of DBpedia [DBpa]. The default configuration provided during the Debian repository installation is utilized. Queries to the store are posed from a client hosted on a different machine. Queries are executed using the Virtuoso Jena provider [Jen], a fully operational Native Graph Model Storage Provider enabling Semantic Web applications to

directly query the Virtuoso RDF store. Before the execution of a set of queries, the system is forced to drop all filesystem caches; the database is also rebooted to clear any internal caches. During the execution of a queryset, the database creates and uses its internal caches.

To compare the performance of the two systems, two RDF datasets (Section 4.12.2) are utilized and the achieved response times for different types of queries are measured, as described below (Section 4.12.3). The times presented are the averages over 1000 executions, unless stated otherwise.

4.12.2 Datasets

LUBM Dataset

The basic comparison dataset in the presented experiments is the Lehigh University benchmark [GPH04] (hence LUBM). The specific benchmark generates synthetic datasets of any size that commit to a single realistic ontology and models information encountered in the academic domain. The LUBM ontology, while not a large one, has the distinctive property of having a small number of different values in the most generic level while being extremely wide in the level of the leaves (i.e., actual entities). The benchmark also includes a set of different query categories with various levels of complexity and selectivity. A significant characteristic of these queries is that some of the categories assume the `rdfs:subClassOf` relationship among the concepts. This property is handled transparently by the proposed system. For the experimental evaluation, a dataset featuring 100 universities with 18 different predicates has been generated resulting in a final dataset of 13.5M triples. By default, ℓ_2 is chosen as a pivot level.

DBpedia Dataset

For the presented evaluation, the real dataset from the DBpedia project [BLK+09] has been utilized. The specific dataset contains extracted information from Wikipedia and major effort has been made to classify this information. The specific ontology [DBpb] is a shallow, cross-domain ontology with the majority of its branches having at most 3 levels. The dataset used in the experiments consists of triples defining the type of the entities (i.e., triples of the form $< \$object\ rdf:type\ \$class >$ from the ontology-based extraction) and triples containing predicates of the ontology namespace. Triples that describe specific properties of an entity have not been included, such as titles, abstracts, images, geographic coordinates, etc., since these are only stored and retrieved from the local database of the nodes. The dataset consists of 6.7M triples for the entity types and 8M triples for the actual predicates.

4.12.3 Query Sets

LUBM queries

A set of nine meaningful queries is included in order to provide direct comparison between the proposed scheme and the centralized store. These queries do not particularly favor any specific storage scheme and require no complex execution plan[2]. Each category of query is now described in detail:

Query 1 (LQ1): This query aims to find any person whose type is `GraduateStudent` and is related to a specific course (e.g., `GraduateCourse0`) according to the relationship `takesCourse`. The resolution of this query starts with the lookup of the value `GraduateStudent`. If no results are found, then the value of the course is looked up and if it is not pivot key or indexed too, then the query is flooded.

Query 2 (LQ2): This query refers to all publications related through the `publicationAuthor` property to a specific professor (e.g., `AssociateProfessor0`). The difference with LQ1 is that the class `Publication` has a wide hierarchy.

Query 3 (LQ3): This query targets the retrieval of all the information related to a professor that `worksFor` for a specific university's department (i.e. `Department0`). It also queries about multiple properties of a single class and most of these properties do not link the subject with other entities: for this reason they are only stored locally in the node of the specific professor (i.e., the property about the `name` and the `emailAddress`). This query also assumes the `rdf:subClass` relationship between `Professor` and its subclasses. Its resolution starts with looking up any of the values of the subquery that contains the `worksFor`.

Query 4 (LQ4): This category searches for all `Professors` that work for any department of a specific university (e.g., `University0`) and selects someone only if she is `Chair` of the department. Let us note here that, in the LUBM ontology, `Department` and `University` lay on the same level. The approach followed for the resolution of the specific query is to find all the departments of the specific university at first and then search for the `Chairs` of each department.

Query 5 (LQ5): This query has to find all the persons that are `memberOf` of a specific department of a university (e.g., `Department0`). The execution of this query can be done in parallel if it is splitted into two queries corresponding to the subClasses of `Person`. In fact, it has been eliminated the `Person` from the root level. The majority of entities included in the LUBM dataset inherit the `Person` class and thus the purpose was to avoid the creation of a very steep hierarchy resulting in gathering all the triples in a single node after a possible roll-up operation. The results for each query are returned back to the node that initiated the query and its union is presented to the user.

[2]Query planning techniques can be implemented on top of the proposed indexing scheme. This work does not deal with queries that require special such strategies

Query 6 (LQ6): This category of queries is about finding all the entities of a specific class or subclass, whether they appear as a subject or an object. For example, we would like to find all the members of the class `Student` or the subclass `UndergraduateStudent`. The corresponding query plan is the simplest one, since only the nodes responsible for the tree structures containing this value need to be found and return all entities of the lowest level.

Query 7 (LQ7): The purpose of this query is to find all `Students` that are related through `takesCourse` to the courses taught by a specific professor (e.g., `AssociateProfessor0`). In this query, `Course` always appears as an object in the subqueries. The query is answered by resolving the subquery that finds all courses of the specific proffessor at first. Afterwards, the students for each found course are retrieved.

Query 8 (LQ8): This query resembles the previous category with increased complexity. It searches for all the `Students` that are members of all the `Departments`, which in their turn are `subOrganization` of a specific university. In this query, the department appears both as a subject and an object. To resolve the query, a subquery is first issued to find all the departments of the university contained in the query. Afterwards, a subquery is issued in order to retrieve any student of these departments. Since `Student` is a root key of the primary key, this subquery is always resolved by the primary dimension. The existing root index (if it is not the pivot key by itself) forwards this subquery to the nodes responsible for the triples containing `Student` as a subject; these nodes search their local databases to retrieve the students of the found courses.

Query 9 (LQ9): This query searches for all the students that `takesCourse` a specific course (e.g., `GraduateCourse0`). This query is either resolved as an indexed query using the existing root index for `Student` or as an exact match in the primary ring, in the case that the initial insertion level is the root level or in the case that the trees containing the `Student` have performed a roll-up towards the root level.

DBpedia queries

Since there is no specific benchmark for the DBpedia set, the queries of the FedBench benchmark suite [SGH+11] regarding DBpedia have been adapted, which focus on analyzing the efficiency of federated query processing strategies over semantic data. Taking into account the FedBench scenarios, a set of six meaningful benchmark queries with different output sizes has been designed:

Query 1 (DBQ1): This is a simple query that searches for all the entities that adhere to the class `President`, which belongs in level ℓ_2 of the DBpedia ontology.

Query 2 (DBQ2): The aim of this query is to find every `President` with a specific nationality (e.g., `United States`) and the political parties they belong to.

Query 3 (DBQ3): This query finds all `Politician` entities and groups them by their `Country` of residence.

Query 4 (DBQ4): This query searches for all `Song` entities as well as the artists of that song.

Query 5 (DBQ5): The purpose of this query is to find all the entities of type `AmericanFootballPlayer` along with their teams and the cities where the teams are based. This is a query, which is split into two subqueries to be executed in parallel. The first query retrieves all the players and the teams, while the second query retrieves all the teams and their respective cities. When the results of both subqueries have been retrieved, they are merged in the node that initiated the query.

Query 6 (DBQ6): This query searches only for the teams of a specific city (e.g., `Baltimore`), and can be considered as subquery of the previous category.

4.12.4 Query Performance over Query Categories

The query response times are presented in Figures 4.24, 4.25 and 4.27 for all categories described in Section 4.12.3. In the generated queries, various values of the constants are queried (e.g., GraduateCourse0, AssociateProfessor0, etc) choosing uniformly among the children that adhere to the type defined by the query and average the response times over 1000 iterations. A single client poses queries to both systems, waiting until the answer(s) are received before posing a new query. For the centralized approach, the maximum response time per query is also registered: this corresponds to the time required for the first answer, since the system performs some main memory result caching to respond faster to the consequent queries.

Query Cat.	LQ1	LQ2	LQ3	LQ4
Avg. rows	4	14	29	20

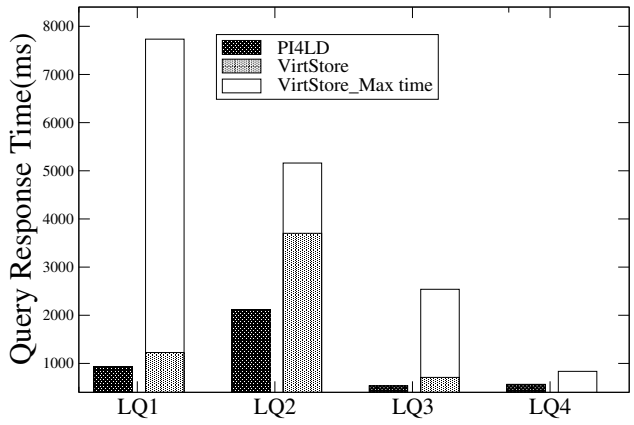


Figure 4.24: *Performance comparison of the response time(ms) for query categories 1–4 of the LUBM benchmark*

Query Cat.	LQ5	LQ6	LQ7	LQ8	LQ9
Avg. rows	625	956711	44	11532	20

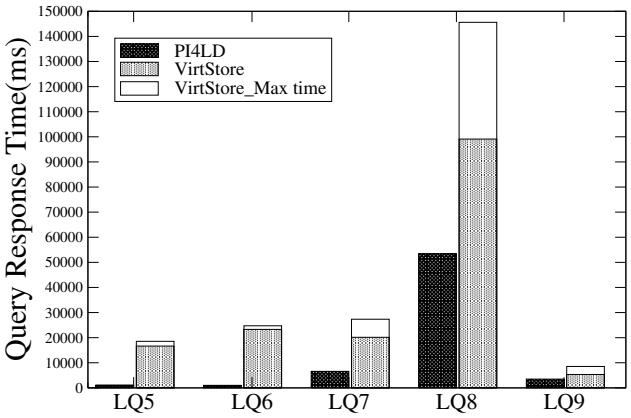


Figure 4.25: *Performance comparison of the average response time(ms) for query categories 5–9 of the LUBM benchmark*

Figure 4.24 depicts the response time for the categories of queries that are mainly of the *(?s, o, p)* format, where the subject is required to be of a specific type. Moreover, these categories of queries have small input and high selectivity. In Figure 4.25, the categories of queries become more complex and return a large number of rows. The average number of returned triples per query is shown in the respective tables above the figures. The results show that PI4LD achieves better response times for all categories of queries.

Even as the complexity of the queries increases, PI4LD manages to resolve the queries efficiently as shown for queries LQ7 and LQ8, which are path queries and involve more joins. Moreover, each node holds a smaller portion of the whole dataset and this feature contributes to its faster processing. Moreover, due to the fact that PI4LD incorporates the whole hierarchy information, less lookups are performed to discover if a found entity is of the requested type. These queries are resolved 2.7 and 1.8 times faster compared to the VirtStore solution. LQ5 is another category which is strongly favoured by the distributed approach, since the query is split into two different subqueries executed in parallel, resulting in a faster response time of the order of 15 times. A significant speedup is achieved for LQ6 as well, which is the simplest query that can be posed to the system: It can be directly resolved by the maintained tree structures.

The selection of the initial insertion level influences the performance of the system for the resolution of a query only until the required re-indexing operations take place. Figure 4.26 shows the time required for answering LQ4 and LQ7 as a function of the sequence of queries, while ℓ_1 was selected as an initial pivot level. The system has been configured to allow roll-up and drill-down operations to take place after 30 queries have been posed. As it is demonstrated,

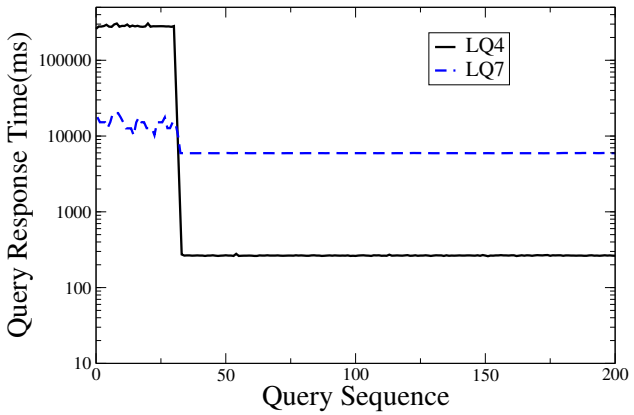


Figure 4.26: *Effect of the adaptive re-indexing operations in the query response time of LQ4 and LQ7, when ℓ_1 is chosen as initial insertion level*

the performed roll-up operation in the queried trees leads to a major speed-up of the query resolution for both categories of the queries. After the re-indexing operation, the performance of the system remains stable for the rest of the followed queries.

Query Cat.	DBQ1	DBQ2	DBQ3	DBQ4	DBQ5	DBQ6
Avg. rows	2239	13	428	1279	7041	303

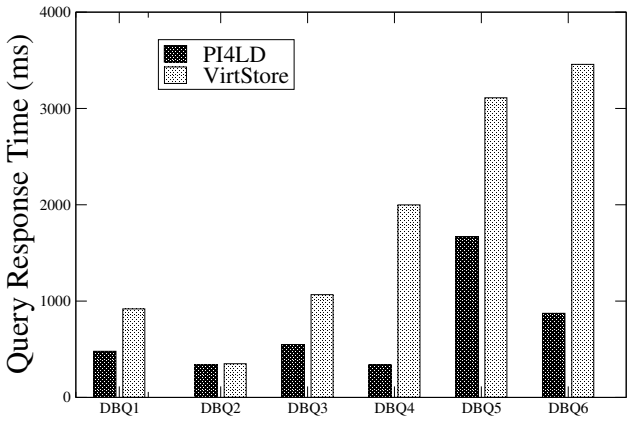


Figure 4.27: *Performance comparison of the average response time(ms) for each category of queries of the DBpedia dataset*

In Figure 4.27, the response times for the categories of queries referring to the DBPedia dataset are demonstrated. The system is allowed to perform re-indexing operations after 30 queries have targeted the same tree. Since in the VirtStore setup, the database caches the results of a resolved query, a restart the database takes place before the execution of each query. PI4LD manages to respond faster than the centralized approach for all categories. The response times achieved in PI4LD depend on the type of the query and the size of the retrieved data. The

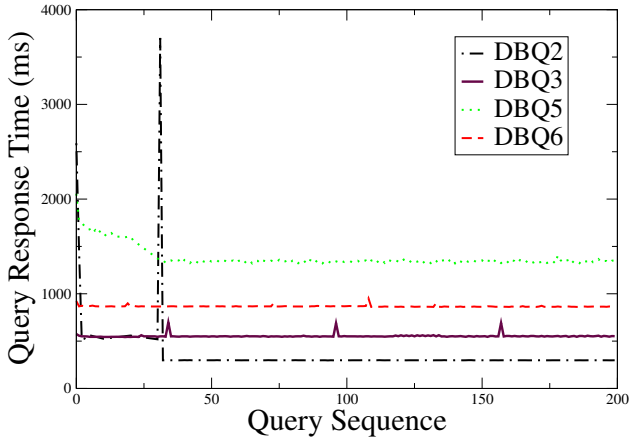


Figure 4.28: *Response times for some selected categories of the DBpedia query workload*

PI4LD mechanism used for the resolution of the query also plays a role in the measured response times. The influence of this factor is shown in more detail in Figure 4.28, where all queries involve two entities. The queried entities in DBQ2 are not indexed during the initial insertion of the dataset and this fact results in the flooding of the query during the first time. Afterwards the entity of the subject is indexed until a drill-down operation takes place (when the spike appears) and the query is answered as an exact match query in the primary ring from then on. On the other hand, the entity `Politician` in DBQ3 is used as a pivot key during initial insertions and thus it has constant response times during the whole query workload. DBQ5 and DBQ6 can be considered as similar queries as far as the query about the same entity appearing as a subject and related entities in the object is concerned. In DBQ5, the object `SportsTeam` is used as a pivot key during the initial insertions and for this reason DBQ5 is always resolved with the procedure regarding exact match query in the secondary dimension. In contrast, `SportsTeam` appears as a subject in DBQ6 and thus all queries are answered as exact matches in the primary dimension.

4.12.5 Concurrent Queries

The performance of both systems is tested under increased load posed from concurrent users, as is the case in most web applications hosting linked data. To achieve this goal, a query workload of 2500 randomly chosen queries from all LUBM categories is generated. First, the queries are sent to the systems using a single client. The average response times are shown in Figure 4.29. The proposed system outperforms the centralized approach by answering the queries almost 5 times faster on average, which is a consistent result to the achieved response times of each category. In Figure 4.30, the average response times are registered when the same workload is applied by 5 concurrent clients. PI4LD manages to maintain its faster response rates at almost the same levels observed in Figure 4.29. The distribution of data among multiple nodes in PI4LD contributes to

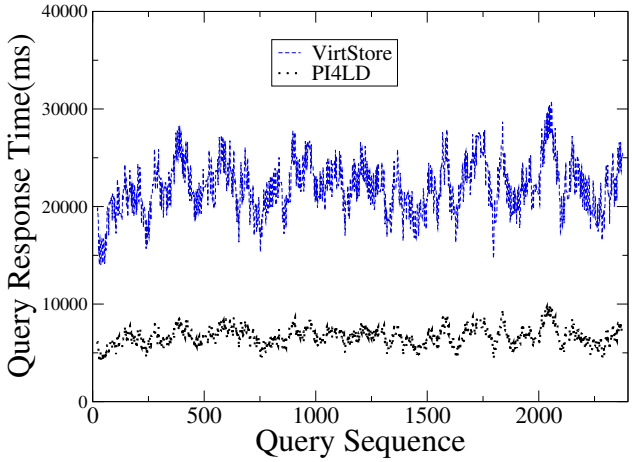


Figure 4.29: *Response times for each query of a workload including randomly chosen queries of the LUBM categories*

this fact due to the parallel processing of multiple queries from different nodes concurrently (even though in the utilized implementation each node processes a single query per time). Moreover, each node holds a smaller portion of data which can be processed needing smaller amounts of time.

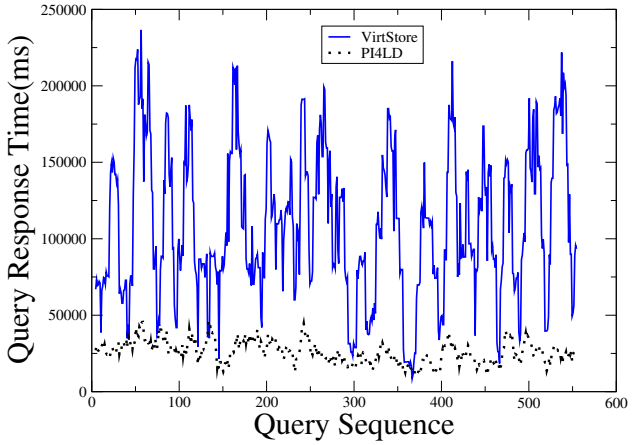


Figure 4.30: *Response times for a workload including randomly chosen queries of the LUBM categories and with 5 clients*

4.12.6 Scalability of the System

The scalability of the system is tested, when the size of the dataset becomes larger. Figure 4.31 depicts the load times for inserting four different LUBM datasets generated for 10, 50, 100 and 200 universities respectively. The datasets are sorted and chunks of triples with the same

pivot key in the primary dimension are inserted in PI4LD. A desirable feature observed in this experiment is that the load time increases linearly to the size of the dataset.

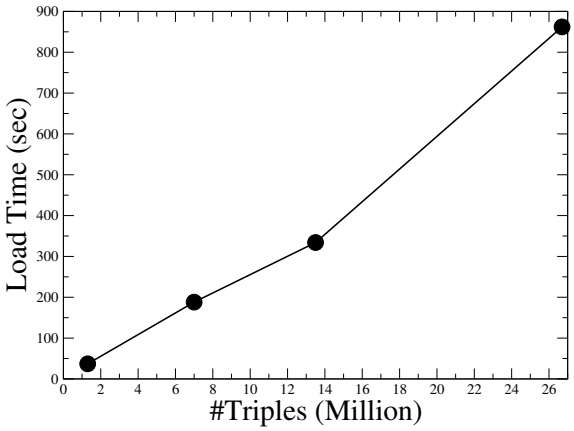


Figure 4.31: *Load time for different sizes of LUBM datasets (university 10, 50, 100 and 200 respectively)*

The impact of the dataset size is also studied for three representative queries of the LUBM benchmark. The categories of the shown queries in Figure 4.32 are selected because they are more complex and assume the ontological information stored in the hierarchical structures of PI4LD. Thus, these queries cannot be resolved by simple DHT lookups for the hashed values either of the subject or the object. In the same time, different features of the system can be exploited during the resolution of each category: LQ5 can be split into subqueries to be executed in parallel and both LQ7 and LQ8 are path queries requiring multiple joins. The increase in the size of the dataset causes an increase in the response time for LQ7 and LQ8, but the slope is small. More increased response times are observed for LQ8, since the query retrieves a significantly larger number of triples compared to LQ7 and so it is more affected from the increase in the data size.

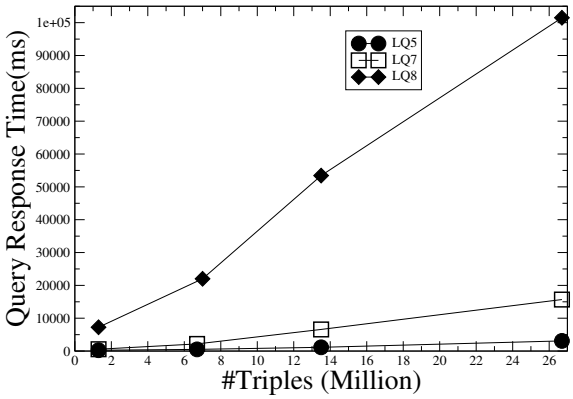


Figure 4.32: *Avg. response times of LUBM queries for different sizes of datasets*

CHAPTER 5

Conclusions and Future Extensions

> The world is a fine place, and worth fighting for.
>
> *Ernest Hemingway*

Several challenges remain open on data management in distributed environments. The evolution of computing systems makes the development of applications that could not even exist some years ago possible. New perspectives are also created for "traditional" applications with limited functionality due to the lack of computing resources, such as processing power and storage space. Distributed data management is one of the main current trends mandated by the advent of web applications, the market globalization, the business process automation and the increasing use of sensors.

The adoption of DHT-based overlays contributes to the design of scalable, fault-tolerant and robust platforms able to cope with data-intensive applications in a fully decentralized manner. Nevertheless, DHTs pose some limitations to the search facilities of such infrastructures and additional methods for indexing and searching data items are needed to overcome these limitations.

In this thesis, providing seamless access to information published by heterogeneous resources and improving the quality of search methods are considered as a significant problem requiring further investigation. The organization, indexing and searching of data is studied aiming at the development of techniques to speed up the query processing. For this reason, the properties of the data structure are taken into account for the shaping of strategies that place and organize

data among the participating nodes in the overlay. The described algorithms and techniques focus on concept hierarchies and data annotated by them. The multidimensional data model is also explored in the proposed methods, but the support of partially structured data items differentiates strongly the specific approach from other existing ones. The fundamental type of queries supported in the proposed approaches is about locating any stored value regarding any level of the stored hierarchies. Emphasis is also given on supporting simple aggregate queries and and multidimensional aggregate queries on various levels of granularity.

Another major contribution of this work is also the adaptive behaviour of the indexing techniques to the query trends of the users on a per node basis. As the size of data is growing rapidly, the exhaustive exploration of all values cannot be completed during a tolerable time period and the indexing schemes lose their effectiveness. If a system has the ability to adapt to the requirements of the users dynamically, then the benefits in the quality and speed of searches can be significant. Moreover, studies on web-based applications show that skewed query loads (e.g., following the zipfian distribution) are a common case, thus a flexible system able to organize its structures to the shifts of the query load presents advantages.

One of the outcomes of this thesis is highly adaptive, scalable, online techniques for organizing and indexing data annotated by concept hierarchies. The system that implements the proposed techniques aims at distributing large amounts of data over a DHT overlay in a way that the hierarchy semantics are maintained. The distinctive characteristic of the demonstrated method is its adaptiveness and the lack of global coordination: The stored hierarchies are indexed over variable levels according to the granularity of the incoming queries. Using only limited knowledge, peers decide on the indexing level of their stored tuples so that flooding is minimized. Moreover, there is no need for off-line updates as the system consistently updates the dataset online at low cost. The experimental evaluation over multiple dynamic and challenging query workloads confirms the properties of the introduced methods: The resultant scheme manages to efficiently answer the large majority of queries using very few messages. It is especially effective in skewed workloads, adapts to sudden shifts in skew and updates datasets in a fast, reliable and cost-efficient manner.

Moreover, an interesting application has been described, which can be deployed on top of a system implementing the proposed methods for the management of hierarchical data. The goal to achieve is the design of a fully distributed Grid Information System. As a result, a system has been built that presents advantages over existing approaches. The re-indexing functionality can be exploited in this use case to enable automatic aggregation of older data and more detailed views of recent ones.

Another principal focus of this thesis has been the design and the implementation of *LinkedPeers*: a system that exploits the above mentioned methods to accommodate data described by multiple dimensions, which are further annotated by concept hierarchies. Each data item can

be described by an arbitrary number of dimensions, while the insertion of values for all dimensions is not required. *LinkedPeers* comprises of multiple 'virtual' rings (or real ones with different identifier spaces), one for each dimension. The idea behind the design of *LinkedPeers* is the creation of an autonomous, "primary" ring for the storage of the inserted data, while a powerful meta-indexing scheme is provided by the secondary rings. A query-driven materialization engine pro-actively computes relevant, partial views of a posed query for future reference, contributing even more to the adaptive nature of the system. Moreover, the indexing granularity is dynamically adapted to the incoming queries, resulting in the reduction of communication cost during distributed query processing. The ability of the system to adapt the organization of its data becomes even more important in the case of multidimensional data, as it is not required that all inserted values and possible combinations of values among dimensions to be indexed leading to exponential growth of storage space as the number of dimensions increases. An extended experimental study shows that the system behaves in an efficient manner during the resolution of the majority of queries, it can be used as lightweight indexing scheme for such data and it adapts to sudden shifts in the skew by deciding correctly to perform re-indexing operations.

The established methods in *LinkedPeers* are further investigated and utilized in *PI4LD*, a system that integrates, stores, indexes and queries linked data. The proposed architecture targets towards two directions: either serving as a system that fully hosts such data or as an intermediate layer for indexing. The adaptiveness of the system to the incoming queries is maintained as one of its basic features and contributes to the speed up of calculating the results.

Finally, the resolution of range queries in a more generic framework based on DHT overlays can be achieved by taking advantage of the properties observed in Space Filling Curves. Without changing the topology imposed of the used DHT protocol, an SFC-based function can be used to replace the hash function of the DHT protocol and combine multiple attributes in a single key used during the insertion of data items in the overlay. A technique based on the ordering provided by SFCs for the resolution of range and point queries has been described and applied for the design of a distributed metadata catalogue, which has been used for the search of annotated multimedia content. It is intended that this technique will be further extended to support range queries in hierarchical data.

5.1 Future Destinations

Possible research extensions may target various aspects of the proposed systems. Some of the most interesting issues are described in this Section.

LinkedPeers can be further customized to enable more efficient processing of other types of queries, especially categories demanding the combination of information dispersed among different nodes, e.g., top-k queries. Therefore, a possible direction of the presented work in this

thesis is the development of more sophisticated techniques for query planning regarding to categories of queries that have not been already addressed. Moreover, some aggregated functions can be inherently implemented in *LinkedPeers*, for example *SUM* or *AVG*, which are widely used functions in analytic applications. The goal is to facilitate the calculation of a specific function automatically during the insertion and update of data and to limit the maintenance of information and indices that are not needed. Furthermore, the already maintained information in *LinkedPeers* can be consulted in such a manner that useful information will be extracted, e.g. the knowledge about the number of links may be used as a metric to classify nodes, so as nodes contributing a small number of results to be avoided during the query forwarding. In general, the establishment of methodologies based on ranking of the participating resources could be helpful to estimate which resources are highly probable to return results for a given query. This estimation can lead to the exclusion of nodes that do not hold relative items and the definition of the order that nodes are visited in order to achieve better response times and less communication overhead.

The study of *scoring functions* for the evaluation of the returned objects after the processing of a query seems promising. Using the results of these functions may be beneficial during the routing of queries and for the quality of the results. The implementation of ranking methods can be utilized for efficient top-k processing, which gains increasing interest in many applications. As the volume of information increases, the search methods need to take into account the preferences of the users and "personalize" the returned results. On one hand, results of high importance to a user may be irrelevant to another user, who posed the same query. On the other hand, the presentation of a large number of results that are of no interest for the user may conceal the really meaningful ones. The search methods in *LinkedPeers* and PI4LD can make use of information provided by the user to narrow down the search space and select the most relevant results to the user's preferences. Such information could be the domain that interests the user, the time period that the data are published, the resources that published the data, etc.

LinkedPeers has the potential to host other types of semi-structured data as well. Data described with the use of XML is considered as a strong candidate. *LinkedPeers* is appropriate for the management of semi-structured data and it can be adapted to the requirements of XML documents. Nevertheless, there are aspects that need to be studied related to the XML applications. For instance, the distribution of documents among different nodes is a problem to be investigated. Moreover, the complexity of the structure of the XML data, XPath and XQuery raise many performance challenges. One direction towards improving the performance of XML query evaluation is based on the utilization of materialized views storing precomputed query results. These views increase the speed of answering queries compared to the times of using the original documents only. The materialization of views is an issue addressed in *LinkedPeers* and needs to further explored for the case of XML data.

Bearing in mind that the proposed systems target also web applications, the heterogeneity of the resources publishing data is highly probable. This heterogeneity can be further addressed in both *LinkedPeers* and PI4LD. Both of the systems can become more flexible and encounter issues such as supporting schema matching and mapping. The links among dimensions introduced in *LinkedPeers* could be further manipulated to extract semantic information and correlations among values that are not directly related.

Another aspect to be explored is how the ability of SFCs to resolve range queries can be incorporated in *LinkedPeers*. Although SFCs require a stable number of dimensions and do not favor queries that do not include a specific value for each dimension, they can be studied so as range queries to become more efficient in *LinkedPeers*.

As far as PI4LD is concerned, the integration of data coming from heterogeneous resources is an ongoing work. Special emphasis is given to the query planning mechanisms and techniques applied for federated resources are studied. The proposed system maintains information which can be further exploited during the resolution of a query. Also, the integration of data coming from different sources can be further explored.

Προσαρμοστική Διαχείριση και Αναζήτηση Δεδομένων Ευρείας Κλίμακας σε Κατανεμημένα Συστήματα

ΚΕΦΑΛΑΙΟ 1

Εισαγωγή

Intelligence is the ability to adapt to change.

Stephen Hawking

Η διαχείριση δεδομένων είναι ένας σημαντικός τομέας της Επιστήμης των Υπολογιστών, ειδικά εάν ληφθεί υπόψη το γεγονός ότι η έκρηξη των δεδομένων δημιουργεί μία καινούργια πραγματικότητα στη ψηφιακή εποχή. Η αλματώδης αύξηση του όγκου του ψηφιακού περιεχομένου είναι ένα παγκόσμιο φαινόμενο που επηρεάζει μια πληθώρα εφαρμογών και το καθιστά μία από τις σημαντικότερες προκλήσεις των Τεχνολογιών Πληροφορικής (Information Technologies). Τα δεδομένα εμφανίζονται παντού και μεγάλοι όγκοι πληροφορίας παράγονται από διάφορες εφαρμογές της καθημερινής μας ζωής και μεταφέρονται μέσω του δικτύου σε μακρινές αποστάσεις. Ποικίλες πηγές, όπως υπολογιστές, κάμερες, αισθητήρες,κτλ., πλημμυρίζουν τον κόσμο μας με περισσότερα δεδομένα την κάθε στιγμή. Η επικρατούσα αυτή κατάσταση απαιτεί νέες λύσεις για τη διαχείριση των δεδομένων και νέες προσεγγίσεις για την ανάπτυξη των υποκείμενων υπολογιστικών υποδομών.

Πολλές μελέτες έχουν διεξαχθεί για την αύξηση του όγκου των δεδομένων σε πολλαπλούς τομείς, συμπεριλαμβανομένου των επιστημονικών και επιχειρησιακών κλάδων. Όμως ακόμα και στην καθημερινή μας ζωή, παράγουμε και καταναλώνουμε μεγάλα ποσά ψηφιακών δεδομένων

κατά την αλληλεπίδραση μας με διάφορες ηλεκτρονικές συσκευές. Δραστηριότητες όπως η χρήση του Διαδικτύου, οι ηλεκτρονικές αγορές, ο διαμοιρασμός αρχείων και η αναζήτηση πληροφορίας συνεισφέρουν στην αύξηση του όγκου των δεδομένων. Εκτός από τη χρήση των προσωπικών υπολογιστών, άλλες δραστηριότητες όπως η παρακολούθηση τηλεόρασης ή η επικοινωνία μέσω τηλεφώνου έχουν προχωρήσει στην ψηφιακή εποχή και αποτελούν σημαντικό παράγοντα στην αύξηση της ψηφιακής πληροφορίας που αποθηκεύεται, μεταφέρεται και αντιγράφεται. Καθημερινές συνήθειες έχουν οδηγήσει στην δημιουργία μίας νέας ψηφιακής πραγματικότητας, όπου για παράδειγμα μόνο τα νοικοκυριά των ΗΠΑ έχουν σπάσει το *"φράγμα του zettabyte"* [GRC+07].

Το ίδιο φαινόμενο παρατηρείται και σε πολλούς επιστημονικούς κλάδους, όπου ποικίλες εφαρμογές με απαιτήσεις για αποτελεσματική διαχείριση δεδομένων ευρείας κλίμακας έχουν καταγραφεί σε τομείς όπως η γεωλογία, η αστρονομία, η αεροναυπηγική, η μετεωρολογία, η πρόβλεψη του κλίματος, η ιατρική απεικόνιση, η χαρτογράφηση γενετικών ακολουθιών. Ένα τέτοιο παράδειγμα είναι η γνωστή περίπτωση της μαζικής παραγωγής δεδομένων που σχετίζεται με το Worldwide LHC Computing Grid (WLCG) [WLC], το οποίο καταγράφει και επεξεργάζεται γεγονότα που παράγονται από τους ανιχνευτές του πειράματος LHC στο CERN. Οι ψηφιακές περιλήψεις των δεδομένων που καταγράφονται από ένα μεγάλο αριθμό αισθητήρων παράγονται με τον εκπληκτικό ρυθμό της τάξης των terabytes ανά δευτερόλεπτο. Η διαχείριση τέτοιου ρυθμού παραγωγής δεδομένων δε μπορεί να γίνει σε μία κεντρικοποιημένη λύση, αν και πολλαπλά στάδια φιλτραρίσματος λαμβάνουν χώρα για τον περιορισμό του στην τάξη των megabytes, ώστε να είναι εφικτή η αποθήκευση και επεξεργασία των δεδομένων σε μία γεωγραφικά κατανεμημένη υποδομή ευρείας κλίμακας. Η ερεύνα στους τομείς της κλιματολογίας αποτελεί έναν ακόμα απαιτητικό τομέα, όπου τα πειράματα για την εξομοίωση μοντέλων, οι απομακρυσμένοι αισθητήρες και οι πλατφόρμες παρατήρησης παράγουν εκατοντάδες από petabytes δεδομένων που πρέπει να ενσωματωθούν αποτελεσματικά σε μία "έξυπνη υποδομή δεδομένων" [AHL+11]. Σε γενικές γραμμές, το petascale computing έχει ήδη επιφέρει τη μετάβαση σε ένα νέο τρόπο διαχείρισης των δεδομένων από τις περισσότερες εφαρμογές και έχει αλλάξει τις αντιλήψεις για τις ανάγκες των εφαρμογών. Οι επιστήμονες έρχονται αντιμέτωποι με τεράστιους όγκους δεδομένων που παράγονται από εξελιγμένα επιστημονικά όργανα και εξομοιώσεις. Εν τω μεταξύ, η δυνατότητα αποθήκευσης petabytes από δεδομένα με οικονομικό τρόπο και η έλευση του διαδικτύου και των υπολογιστών υψηλών επιδόσεων έχει σαν αποτέλεσμα την ακόμη μεγαλύτερη αύξηση των δεδομένων που παράγονται και αντιγράφονται [BGS06]. Το όραμα του "exascale computing" θα επιφέρει ακόμα μεγαλύτερες προκλήσεις, καθώς θα παρέχει τα μέσα για τη δημιουργία ακόμα πιο πολύπλοκων μοντέλων και την εκτέλεση πιο αναλυτικών εξομοιώσεων. Αυτή η νέα πραγματικότητα όσον αφορά τις δυνατότητες της υπολογιστικής επεξεργασίας θα δημιουργήσει νέες απαιτήσεις για την διαχείριση, μεταφορά, ανάλυση και οπτικοποίηση μεγάλων και πιθανώς κατανεμημένων συλλογών από δεδομένα [GL09].

Οι καινοτόμες τεχνικές για τη διαχείριση δεδομένων μπορούν να διαδραματίσουν σημαντικό ρόλο και στον τρόπο λειτουργίας των επιχειρήσεων και των οργανισμών. Η συνηθισμένη αντιμετώπιση που συναντάται στη βιομηχανία και τις επιχειρήσεις είναι η διατήρηση των δεδομένων τους σε τοπικά συστήματα αποθήκευσης, που συνήθως χρησιμοποιούν τεχνολογίες βάσεων δεδομένων. Όμως, γίνεται όλο και πιο συνηθισμένο το εξής φαινόμενο: πετυχημένες και αποδοτικές λύσεις κεντρικοποιημένων βάσεων δεδομένων δε μπορούν να αντεπεξέλθουν στο μεγάλο όγκο της πληροφορίας που πρέπει να διαχειριστούν. Σε πολλές περιπτώσεις, η αποθήκευση της συνολικής πληροφορίας που παράγεται από κάποια εφαρμογή είναι απλά απαγορευτική, όπως σε ιατρικές εφαρμογές [MCB+11]. Για παράδειγμα, δεν είναι δυνατό να αποθηκευτεί πάνω από 90% των δεδομένων που παράγεται από ιατρικές εφαρμογές (π.χ. όλα σχεδόν τα δεδομένα που παράγονται από την καταγραφή των εγχειρήσεων). Έτσι λοιπόν, η νέα τάση είναι η δημιουργία των κατάλληλων λογισμικών εργαλείων που μπορούν να αποθηκεύουν, να διαχειρίζονται και να αναλύουν τα επονομαζόμενα "big data". Ένας από τους στόχους που μπορεί να επιτευχθεί με τέτοιου είδους προσεγγίσεις είναι η εξαγωγή των τάσεων που επικρατούν, ώστε με την εκμετάλλευση της γνώσης αυτής να επιτευχθεί η αύξηση της παραγωγικότητας και της ανταγωνιστικότητας. Ένα τέτοιο παράδειγμα αφορά την περίπτωση της ανάλυσης στοιχείων για την πρόβλεψη των τάσεων στις προτιμήσεις των πελατών. Η εξαγωγή γνώσης που εμπεριέχεται στην αποθηκευμένη πληροφορία ενός οργανισμού με τη χρήση πλατφορμών αναλυτικής επεξεργασίας μπορεί να ευνοήσει τις δραστηριότητες ενός οργανισμού, αλλά ταυτόχρονα καταδεικνύει την ανάγκη για ανάπτυξη νέων εργαλείων και υποδομών για την αντιμετώπιση των προβλημάτων που περιγράφηκαν.

Ωστόσο, το μέσο που άλλαξε τον τρόπο που γίνεται η πρόσβαση στην πληροφορία και τον τρόπο με τον οποίο μπορούμε να εκμεταλλευτούμε τη "δύναμη των δεδομένων" είναι ο Παγκόσμιος Ιστός. Στις μέρες μας, τεράστιοι όγκοι δεδομένων μεταφέρονται μέσω του δικτύου συνεχώς και όλοι μπορούν να εκτελέσουν αναζητήσεις στην "αστείρευτη" πληροφορία που διατίθεται στο Διαδίκτυο. Η πληροφορία αυτή δεικτοδοτείται συνεχώς από τις μηχανές αναζήτησης, ώστε να αυξηθεί η ταχύτητα ανάκτησης των αποτελεσμάτων αλλά και η ποιότητα τους. Χαρακτηριστικό παράδειγμα αποτελεί το γεγονός ότι το 2008 η εταιρεία Google [Goo] ανακοίνωσε ότι κατάφερε να δεικτοδοτήσει ένα τρισεκατομμύριο ιστοσελίδες ταυτόχρονα, ενώ ο αντίστοιχος αριθμός το έτος 2000 ήταν ένα δισεκατομμύριο και η πρώτη προσπάθεια που έγινε το 1998 αναφέρονταν σε είκοσι έξι εκατομμύρια[1]. Η κατανεμημένη υποδομή που χρησιμοποιείται από το Google επιτρέπει στις εφαρμογές της την εύκολη διάσχιση ενός γράφου με πολλά τρισεκατομμύρια από συνδέσεις και τη γρήγορη ταξινόμηση των δεδομένων με όγκο που ανέρχεται σε petabytes. Όλα αυτά διαδραματίζονται μόνο και μόνο για να απαντηθεί μία απλή αναζήτηση [1]. Όμως η συγκεκριμένη μηχανή αναζήτησης δεν είναι η μόνη που αντιμετωπίζει την πρόκληση της διαχείρισης

[1]http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html

μεγάλου όγκου δεδομένων. Τα δίκτυα κοινωνικής δικτύωσης είναι μία άλλη μορφή πηγών που προσθέτει νέο περιεχόμενο με αυξανόμενο ρυθμό. Στο Facebook [fac], περισσότεροι από 800 εκατομμύρια χρηστών ανεβάζουν περισσότερα από 250 εκατομμύρια φωτογραφιών ανά ημέρα[2]. Στο Twitter [Twi], ο μέσος αριθμός των "tweets" που στέλνουν οι χρήστες του σε μία μόνο ημέρα ξεπερνάει τα 140 εκατομμύρια [3]. Τέλος, κάθε ένα από τα 20 εκατομμύρια άρθρα του Wikipedia τροποποιείται πάνω από 30 φορές ανά μήνα και πάνω από 8K νέων άρθρων προστίθενται στη συνολική συλλογή ανά ημέρα [4]. Ένα ακόμα εντυπωσιακό γεγονός είναι ότι αν και μερικά δισεκατομμύρια σελίδων είναι ήδη δεικτοδοτημένα στο διαδίκτυο, υπάρχει ακόμα ένα μεγάλο μέρος του περιεχομένου του που δεν είναι προσβάσιμο από τις μηχανές αναζήτησης και είναι γνωστό ως το *"Deep Web"*. Λαμβάνοντας όλα αυτά υπόψη προκύπτει ότι η συσχέτιση της πληροφορίας από διαφορετικές πηγές δημιουργεί την ανάγκη για νέες μορφές μηχανών αναζήτησης [CBB11]. Οι μηχανές αυτές πρέπει να είναι ικανές να υποστηρίζουν πιο πολύπλοκους τύπους ερωτημάτων που προϋποθέτουν την ενοποίηση και το συνδυασμό των δεδομένων.

Ποικίλα σενάρια χρήσης από διάφορους τομείς περιγράφηκαν ως βασικές πηγές δεδομένων. Ένα συχνό φαινόμενο που παρατηρείται είναι ότι η αποθήκευση όλων των δεδομένων που παράγονται από κάποια εφαρμογή δεν είναι πάντα απαραίτητη. Σε μοντέρνες εφαρμογές, υπάρχουν τύποι δεδομένων που παράγονται για μικρά χρονικά διαστήματα, χρησιμεύουν για ένα συγκεκριμένο σκοπό (π.χ. για την επικοινωνία μεταξύ ψηφιακών συσκευών) και στη συνέχεια καταστρέφονται. Χωρίς αυτό να υποβιβάζει τη χρησιμότητα τέτοιων δεδομένων, το συγκεκριμένο παράδειγμα τονίζει ακόμα περισσότερο πόσο σημαντικό είναι να επιτυγχάνεται η εξαγωγή χρήσιμης πληροφορίας από μεγάλο όγκο δεδομένων πέραν από την απλή αποθήκευση του. Μεγάλη προσπάθεια για την ανάπτυξη νέων μεθόδων και εργαλείων στοχεύει προς αυτήν την κατεύθυνση. Όπως αναφέρεται στο [GR11], ο ρυθμός δημιουργίας μεταδεδομένων με αυτοματοποιημένο τρόπο διπλασιάζεται σε σύγκριση με την αύξηση του όγκου των πραγματικών δεδομένων. Ένα σύνηθες παράδειγμα είναι ο συνεχώς αυξανόμενος αριθμός των φωτογραφιών στα μέσα κοινωνικής δικτύωσης. Για να επιτευχθεί η αναζήτηση τους αναπτύσσονται ειδικά εργαλεία για το σχολιασμό τους με χρήση τεχνολογιών αναγνώρισης προσώπων; διαφορετικά δεν είναι δυνατόν να διαμοιραστούν και να αναζητηθούν χιλιάδες φωτογραφίες.

Μία ευρέως διαδεδομένη λύση για διαχείριση δεδομένων είναι η δημιουργία κεντρικών αποθηκών λόγω των διάφορων πλεονεκτημάτων που προσφέρουν. Η αποθήκευση όλου του περιεχομένου σε ένα κεντρικό σημείο επιτρέπει τη διατήρηση μίας γενικής εικόνας του συνόλου των αποθηκευμένων δεδομένων και καθιστά δυνατή την αποτελεσματικότερη επεξεργασία τους.

[2] http://www.facebook.com/press/info.php?statistics
[3] http://blog.twitter.com/2011/03/numbers.html
[4] http://stats.wikimedia.org/EN/TablesWikipediaZZ.htm

Επίσης, όλη η απαραίτητη πληροφορία για την επίλυση μίας ερώτησης είναι προσβάσιμη άμεσα στο σημείο που γίνεται η επεξεργασία της και δε χρειάζεται η μεταφορά δεδομένων ανάμεσα σε διαφορετικούς κόμβους. Όμως, ο μεγάλος όγκος των δεδομένων επηρεάζει αρνητικά την αποδοτικότητα των κεντρικών λύσεων και διάφορες "προβληματικές" καταστάσεις μπορούν να προκύψουν, όπως η αύξηση του χρόνου για τη φόρτωση και επεξεργασία των δεδομένων και η μη διαθεσιμότητα των παρεχόμενων υπηρεσιών όσο αυξάνεται η ζήτηση τους.

Οι "μοντέρνες" εφαρμογές υιοθετούν πιο κατανεμημένες προσεγγίσεις για την αντιμετώπιση τέτοιου είδους ζητημάτων. Τα κατανεμημένα συστήματα πληροφορικής είναι ένας διαρκώς αναπτυσσόμενος κλάδος και για αυτό το λόγο είναι δύσκολο να δοθεί ένας ακριβής ορισμός για αυτά. Κατανεμημένοι υπολογισμοί λαμβάνουν χώρα ακόμα και σε έναν προσωπικό υπολογιστή, όπου διαφορετικά μέρη εκτέλεσης μίας εφαρμογής ανατίθενται σε διαφορετικούς πόρους του συστήματος. Η συγκεκριμένη διατριβή αναφέρεται στα κατανεμημένα συστήματα στα πλαίσια του ορισμού που μπορεί να βρεθεί στις πηγές [TS06] και [OV11] της βιβλιογραφίας: Ένα κατανεμημένο υπολογιστικό σύστημα αποτελείται από ένα αριθμό αυτόνομων στοιχείων που είναι είτε ομοιογενή είτε ετερογενή, διασυνδέονται μέσω δικτύου και μπορούν να συνεργάζονται για την επίλυση των εργασιών που τους ανατίθενται. Η έννοια της κατανομής μπορεί να εφαρμοστεί σε διάφορα επίπεδα θεώρησης, αλλά στη συγκεκριμένη διατριβή η κατανομή των υπολογισμών και η κατανομή των δεδομένων μελετώνται. Η κατανομή των υπολογισμών περιλαμβάνει την ανάθεση διαφορετικών τμημάτων μίας εργασίας σε διαφορετικούς υπολογιστικούς κόμβους, οι οποίοι συνεργάζονται για τη διεκπεραίωση της. Παρόλα αυτά, η κατανομή των δεδομένων σε διαφορετικούς υπολογιστικούς πόρους είναι αυτή που συγκεντρώνει την κύρια προσοχή σε αυτήν τη διατριβή.

Ένα σημαντικό χαρακτηριστικό των κατανεμημένων συστημάτων είναι η αποφυγή μεμονωμένων σημείων που μπορούν να αποτελέσουν σημεία αποτυχίας ή βλάβης του συστήματος. Τα τελευταία χρόνια έχουν παρατηρηθεί διάφορα συμβάντα όπου κόμβοι που παρέχουν σημαντικές υπηρεσίες να θέτονται εκτός λειτουργίας λόγω κάποιου σφάλματος ή μεγάλου φόρτου εργασίας του κόμβου [SC11]. Επιπλέον, η κατανομή της επεξεργασίας και ανάκτησης σε πολλαπλούς κόμβους έχει σαν αποτέλεσμα την επίτευξη καλύτερης απόδοσης, ενώ μπορούν να αποφευχθούν βαρέως φορτωμένοι ή ανενεργοί κόμβοι.

Ένας άλλος βασικός στόχος που επιτυγχάνεται σε ένα κατανεμημένο σύστημα είναι η κατάλληλη τοποθέτηση των δεδομένων σε κόμβους, ειδικά στις περιπτώσεις όπου οι εφαρμογές εκτελούνται με κατανεμημένο τρόπο και αυτό επιβάλλεται από την ίδια τη φύση της υλοποίησης τους, όπως στις διαδικτυακές εφαρμογές ή σε επιστημονικές εφαρμογές. Η κατανεμημένη διαχείριση των δεδομένων επιτρέπει περισσότερη ευελιξία στην τοποθέτηση των δεδομένων ανάλογα με την εφαρμογή και αυτό έχει τεθεί ως απαίτηση και στη μεγάλη πρόκληση για την επίτευξη του "exascale" computing [GL09]. Επίσης, η δημιουργία πολλαπλών αντιγράφων του ίδιου περιεχομένου μπορεί να συνεισφέρει σημαντικά στη διαθεσιμότητα τους. Ένα άλλο βασικό

στοιχείο αναφοράς είναι ότι σε ένα κατανεμημένο σύστημα συμμετέχουν πολλοί διαφορετικοί κόμβοι. Συνεπώς το ίδιο αρχείο μπορεί να αντιγραφεί σε διαφορετικούς κόμβους, ώστε να αυξηθεί η πιθανότητα να παραμείνει κάποιο αντίγραφο διαθέσιμο ακόμα και αν κάποιοι κόμβοι τεθούν εκτός λειτουργίας ή υπάρχουν προβλήματα συνδεσιμότητας ή περιορισμένου εύρους ζώνης.

Τέλος, αξίζει να αναφερθεί ότι οι προσεγγίσεις που δε χρησιμοποιούν κεντρικές δομές μπορούν να διαχειριστούν με μεγαλύτερη αποτελεσματικότητα την αύξηση των δεδομένων τους και των αιτημάτων προς αυτές. Επίσης, η προσθήκη νέων κόμβων στο σύστημα γίνεται χωρίς να χρειάζεται η ολοκληρωτική αναδόμηση του. Συμπερασματικά, η επεκτασιμότητα είναι ένα από τα βασικά πλεονεκτήματα των κατανεμημένων προσεγγίσεων. Στα πλαίσια της συγκεκριμένης διατριβής, ο παράγοντας της επεκτασιμότητας μελετάται ως προς τον όγκο των δεδομένων, το μέγεθος του δικτύου και των αριθμό των πελατών. Αυτό που θεωρείται σημαντικό είναι η ανάπτυξη ενός συνεκτικού συστήματος που διαχειρίζεται τα δεδομένα αποδοτικά και παρέχει τις υπηρεσίες του με ενιαίο τρόπο ανεξάρτητα από τον αριθμό των κόμβων που συμμετέχουν. Οι χρήστες και οι εφαρμογές δε χρειάζεται να γνωρίζουν την ακριβή τοποθεσία που είναι αποθηκευμένα τα δεδομένα και απευθύνουν τα ερωτήματα στο σύστημα με τον ίδιο τρόπο σαν να βρίσκονταν τοπικά.

Στη βιβλιογραφία υπάρχουν πολλές κατηγορίες κατανεμημένων αρχιτεκτονικών εκτός από το βασικό μοντέλο πελάτη - εξυπηρετητή. Τα δίκτυα ομότιμων κόμβων και τα Υπολογιστικά Πλέγματα μπορούν να θεωρηθούν ως κάποια από τα πιο διαδεδομένα μοντέλα. Κάποιες από τις θεμελιώδεις έννοιες εμφανίζονται στην πλειοψηφία των κατανεμημένων μοντέλων, ενώ βάσει κάποιων άλλες διαφοροποιούνται τα διάφορα συστήματα. Στη συνέχεια περιγράφονται τα μοντέλα που συναντώνται στα δίκτυα ομότιμων κόμβων και στα Υπολογιστικά Πλέγματα δεδομένου ότι είναι αυτά που μελετώνται και χρησιμοποιούνται και στη συγκεκριμένη διατριβή.

Δίκτυα Ομότιμων Κόμβων (Peer-to-Peer computing): Οι τεχνολογίες ομότιμων κόμβων χρησιμοποιούνται για την ανάπτυξη επεκτάσιμων δικτυακών επικαλύψεων. Οι κόμβοι συμμετέχουν στην επικάλυψη με έναν αυτόνομο τρόπο χωρίς επίβλεψη και συμπεριφέρονται είτε σαν εξυπηρετητές είτε σαν πελάτες. Η επικάλυψη που προκύπτει μπορεί να προσαρμόζεται δυναμικά στις αναχωρήσεις και στις αφίξεις κόμβων, ενώ το δίκτυο αυτό-οργανώνεται με εσωτερικές του διαδικασίες. Τα συστήματα ομότιμων κόμβων μπορούν να αποτελέσουν κατάλληλες λύσεις για τη δημιουργία δικτύων διαμοιρασμού αρχείων και περιεχομένου ευρείας κλίμακας. Επιπλέον, η χρήση δικτύων ομότιμων κόμβων είναι επικερδής για εφαρμογές που εξαρτώνται σημαντικά από τον τρόπο που γίνεται η διαχείριση των δεδομένων, επειδή προσφέρουν αποτελεσματική δρομολόγηση των μηνυμάτων, ανεκτικότητα σε σφάλματα, επεκτασιμότητα και εξισορρόπηση φόρτου σε περιβάλλοντα ευρείας κλίμακας. Δεδομένης της κατανεμημένης φύσης των αρχιτεκτονικών ομότιμων κόμβων, ένα

σημαντικό ζήτημα είναι η οργάνωση των δεδομένων στους κόμβους ώστε να εξασφαλιστεί η διαθεσιμότητα τους κατά τη διάρκεια δυναμικών αλλαγών της τοπολογίας του δικτύου και υψηλού φόρτου εργασίας, χωρίς να επηρεάζεται η αποτελεσματικότητα των μεθόδων αναζήτησης. Οι υπάρχουσες προσεγγίσεις προσανατολίζονται προς δύο κατευθύνσεις: τα δεδομένα τοποθετούνται στους κόμβους είτε τυχαία είτε με ντετερμινιστικό τρόπο που καθορίζεται από το χρησιμοποιούμενο πρωτόκολλο. Σημαντική κατηγορία δικτύων ομότιμων κόμβων είναι οι δομημένες επικαλύψεις που υλοποιούν ένα Κατανεμημένο Πίνακα Κατακερματισμού. Σε αυτήν την περίπτωση, το πρωτόκολλο που χρησιμοποιείται ελέγχει με αυστηρό τρόπο την ανάθεση των δεδομένων σε κόμβους αναθέτοντας τους αναγνωριστικά, τα οποία με τη σειρά τους αντιστοιχίζονται στα αναγνωριστικά των κόμβων με τη χρήση μίας μετρικής απόστασης [BKK+03]. Η κατανεμημένη δεικτοδότηση αυτών των επικαλύψεων παρέχει αποτελεσματική αναζήτηση των αποθηκευμένων αντικειμένων και τα καθιστά ιδιαίτερα δημοφιλή. Δίκτυα ομότιμων κόμβων έχουν χρησιμοποιηθεί για διάφορες διαδικτυακές εφαρμογές ευρείας κλίμακας, όπως για τη διανομή περιεχομένου (Bittorent [Bit], eMule [eMu]) και σε δίκτυα επικοινωνίας (Skype [Sky]).

Υπολογιστικό Πλέγμα (Grid computing): Τα Υπολογιστικά Πλέγματα προορίζονται κυρίως για υπολογιστικές εφαρμογές υψηλών επιδόσεων. Μία πλατφόρμα τύπου Πλέγματος μπορεί να θεωρηθεί ως ένα σύνολο ετερογενών υπολογιστικών και αποθηκευτικών πόρων που διαμοιράζονται από διαφορετικές ομάδες χρηστών. Τα βασικά συστατικά μέρη τέτοιων υπαρχόντων υποδομών είναι υπολογιστικές συστοιχίες που διασυνδέονται με δίκτυα υψηλών ταχυτήτων και ανήκουν σε διαφορετικούς διαχειριστικούς τομείς. Οι υπολογιστικοί και αποθηκευτικοί πόροι διαμοιράζονται βάσει πολιτικών που έχουν θεσπιστεί μεταξύ της κάθε ομάδας των χρηστών και των παρόχων. Η ανάθεση των εργασιών στους κατάλληλους πόρους επιτυγχάνεται με την ύπαρξη κεντρικών σημείων ελέγχου, τα οποία διατηρούν πληροφορία (π.χ. αριθμός των εργασιών που εκτελούνται, αριθμός των εργασιών που αναμένουν την εκτέλεση τους) για το σύνολο της υποδομής, ώστε να μπορούν να προγραμματίζουν κατάλληλα τις εργασίες.

1.1 Περιγραφή του Προβλήματος

Η κατανεμημένη διαχείριση δεδομένων προσφέρει επεκτασιμότητα, ανεκτικότητα σε σφάλματα και άλλες ενδιαφέρουσες ιδιότητες σε εφαρμογές που διαχειρίζονται δεδομένα μεγάλου όγκου. Παρόλα αυτά, προκύπτουν διάφορα ζητήματα τα οποία πρέπει να αντιμετωπιστούν κατά τη σχεδίαση μίας κατανεμημένης πλατφόρμας, ενώ το είδος της εφαρμογής και ο τύπος των δεδομένων παίζουν καθοριστικό ρόλο για την αρχιτεκτονική που θα προκύψει.

Το πρόβλημα που μελετάται σε αυτήν τη διατριβή είναι η αποτελεσματική διαχείριση μεγάλου όγκου ιεραρχικών, πολυδιάστατων δεδομένων που μπορεί να είναι και μερικώς δομημένα. Οι τεχνικές που αναπτύσσονται αφορούν υπολογιστικά περιβάλλοντα που απαρτίζονται από δικτυακά διασυνδεδεμένους κόμβους που ανήκουν είτε σε γεωγραφικά διάσπαρτους πόρους είτε σε υπολογιστικά κέντρα. Τα ζητήματα που αντιμετωπίζονται είναι η αποδοτική ενοποίηση, οργάνωση, δεικτοδότηση, επεξεργασία και ενημέρωση δεδομένων που περιγράφονται με εννοιολογικές ιεραρχίες. Επιπρόσθετα, τα ζητήματα αυτά αντιμετωπίζονται και για δεδομένα πολλαπλών διαστάσεων που μπορεί να είναι μερικώς δομημένα.

Οι εννοιολογικές ιεραρχίες συναντώνται συχνά σε εφαρμογές "εξόρυξης δεδομένων" και αναλυτικής επεξεργασίας δεδομένων, οι οποίες αναλύουν πολλά δεδομένα για την εξαγωγή μοτίβων και τάσεων [HK00]. Τα αποτελέσματα τέτοιου τύπου επεξεργασίας προορίζονται τις περισσότερες φορές για συστήματα λήψης αποφάσεων, εργαλεία για "business intelligence" και πλατφόρμες για αναλυτική εξόρυξη δεδομένων. Αυτό έχει σαν αποτέλεσμα να δημιουργείται η απαίτηση για επεξεργασία ερωτημάτων με πολύπλοκες μορφές που περιλαμβάνουν πολλαπλά επίπεδα σύνοψης. Η χρήση των ιεραρχιών καθιστά δυνατή τη διάταξη των ιδιοτήτων (attributes) που υπάρχουν σε μία βάση δεδομένων και βοηθά σημαντικά στην οργάνωση και επαναχρησιμοποίηση της πληροφορίας. Μία *εννοιολογική ιεραρχία* ή *ταξινομία* ορίζει μία αλληλουχία αντιστοιχίσεων από πιο γενικές σε πιο ειδικές έννοιες. Οι εννοιολογικές ιεραρχίες είναι σημαντικές γιατί επιτρέπουν τη κατηγοριοποίηση της πληροφορίας, επομένως ευκολότερες αναζητήσεις και δημιουργίες συνόψεων. Η επιλογή και η διάταξη των εννοιών επιτελείται συνήθως από τις ίδιες τις εφαρμογές και από τους ειδικούς του κάθε κλάδου. Για παράδειγμα, μία εννοιολογική ιεραρχία για την ιδιότητα *τοποθεσία* μπορεί να περιλαμβάνει τα ακόλουθα επίπεδα: `Χώρα - Πόλη - Οδός - Ταχυδρομικός Κώδικας`, όπου το επίπεδο της `Χώρας` αντιστοιχεί στο πιο γενικό επίπεδο. Όσο ο όγκος των δεδομένων αυξάνεται και η εξαγωγή χρήσιμης πληροφορίας γίνεται δυσκολότερη, οι εννοιολογικές ιεραρχίες μπορούν να συνεισφέρουν στη οργάνωση των δεδομένων και να προσφέρουν γρήγορες επισκοπήσεις των δεδομένων λόγω της ιδιότητας για διατήρηση συνόψεων που περιέχεται στη δομή τους. Εκτός από τη χρήση των ιεραρχικών δομών στις βάσεις δεδομένων και στις αποθήκες δεδομένων (data warehouses), μπορούν να θεωρηθούν διάφορες άλλες προσεγγίσεις, ειδικά όταν ο διαμοιρασμός της πληροφορίας εκτελείται μεταξύ εφαρμογών και υπηρεσιών. Για παράδειγμα, οι ιδιότητες των μεταδεδομένων θα μπορούσαν να δομηθούν ιεραρχικά. Ένα γενικότερο παράδειγμα είναι οι ταξινομίες που αποτελούν ένα θεμελιώδες μέρος του Σημασιολογικού Ιστού (Semantic Web).

Το πολυδιάστατο μοντέλο δεδομένων είναι ένα δομικό συστατικό μέρος της αναλυτικής επεξεργασίας δεδομένων. Σε αυτό το μοντέλο, πολλαπλές ιδιότητες αντιστοιχίζονται σε διαστάσεις που περιγράφουν αριθμητικές τιμές (μία ή περισσότερες) που ονομάζονται *facts*. Το πρόβλημα που προκύπτει με τα πολυδιάστατα δεδομένα είναι ότι η αύξηση του αριθμού των διαστάσεων

έχει ως αποτέλεσμα την εκθετική αύξηση του αριθμού των όψεων που μπορούν να υπολογιστούν. Η πολυπλοκότητα των πολυδιάστατων δεδομένων αυξάνει ακόμα περισσότερο όταν η κάθε διάσταση χαρακτηρίζεται από μία διαφορετική εννοιολογική ιεραρχία. Εκτός από την ευρεία χρήση του πολυδιάστατου μοντέλου δεδομένων σε εφαρμογές τύπου OLAP για τη δημιουργία *data cubes*, αυτό το μοντέλο μπορεί να χρησιμοποιηθεί και σε πολλές άλλες περιπτώσεις. Για παράδειγμα, μπορεί να θεωρηθεί ότι οι ιδιότητες για την περιγραφή φωτογραφιών (όπως τοποθεσία, όνομα, ημερομηνία, θέμα, τύπος φωτογραφίας, κτλ.) σχηματίζουν ένα πολυδιάστατο χώρο, όπου κάθε σημείο αντιστοιχεί στη συνολική περιγραφή μίας φωτογραφίας. Ο τύπος των ερωτημάτων που αναμένεται σε μία τέτοια εφαρμογή για την αναζήτηση φωτογραφιών βάσει της περιγραφής τους είναι η εύρεση των φωτογραφιών που έχουν τραβηχτεί σε μία συγκεκριμένη τοποθεσία ή που έχουν ένα συγκεκριμένο θέμα ή που συνδυάζουν και τα δύο αυτά χαρακτηριστικά. Επίσης, το παράδειγμα αυτό δείχνει ότι οι τιμές όλων των διαστάσεων μπορεί να μην είναι πάντα γνωστές, π.χ. να μη υπάρχει η τοποθεσία στην περιγραφή όλων των φωτογραφιών. Ημιδομημένα δεδομένα (semi-structured data) [Bun97] που είτε περιέχουν το σχήμα τους είτε ακολουθούν κάποιο μη αυστηρώς ορισμένο σχήμα μελετώνται σε διάφορες εφαρμογές, ειδικά σε αυτές που σχετίζονται με το Διαδίκτυο. Η διαχείριση αυτής της κατηγορίας δεδομένων περιγράφεται ως μία από τις επερχόμενες προκλήσεις που θα αντιμετωπίσει η κοινότητα των βάσεων δεδομένων [AAB+08], ενώ ταυτόχρονα υπογραμμίζεται ότι η εποχή των βάσεων δεδομένων που περιγράφονται από αυστηρώς ορισμένα σχήματα δίνει τη θέση της στη διαχείριση ετερογενών δεδομένων (δηλαδή δομημένων, ημι-δομημένων και αδόμητων) από διαφορετικές αποθήκες.

Ένα άλλο αντιπροσωπευτικό παράδειγμα που αφορά δεδομένα με τις παραπάνω ιδιότητες συναντάται στο Διαδίκτυο Δεδομένων (Web of Data) [BHBL09], που παρέχει τη δυνατότητα για την αξιοποίηση διαφορετικών συνόλων δεδομένων μέσω της διασύνδεσης τους. Με την υιοθέτηση πρακτικών σαν αυτές που υιοθετούνται στα Διασυνδεδεμένα Δεδομένα (*Linked Data*) [Data] δίνεται μία μεγάλη ευκαιρία για τη διασύνδεση πληροφορίας που εμφανίζεται διάσπαρτη στο Διαδίκτυο και συνεπώς για τη διασύνδεση διαφορετικών πηγών σε ένα παγκόσμιο ενιαίο χώρο δεδομένων, που έχει σαν αποτέλεσμα την αλλαγή του τρόπου με τον οποίο εκμεταλλευόμαστε τα δεδομένα στο Διαδίκτυο. Η επίτευξη διασυνδεδεμένων πόρων της κλίμακας του Διαδικτύου συνεπάγεται ότι τα δεδομένα τους θα χαρακτηρίζονται από υψηλούς βαθμούς ανομοιογένειας και θα σχετίζονται με διάφορους τομείς όπως με πρόσωπα, εταιρείες, ταινίες, μουσική, τοποθεσίες, κ.α.. Το συγκεκριμένο παράδειγμα τονίζει την ανάγκη για την υποστήριξη και ενοποίηση δομημένου και ημι-δομημένου περιεχομένου το οποίο ταξινομείται σε διαφορετικές κατηγορίες και παράγεται από διαφορετικές πηγές. Ως εκ τούτου, νέες προσεγγίσεις προκύπτουν για την αντιμετώπιση των προκλήσεων της εξαγωγής δομής από τέτοιου είδους δεδομένα και της ανάπτυξης μεθόδων για την αποτελεσματική επερώτηση του αποτελέσματος της ενοποίησης τους.

Η συγκεκριμένη διατριβή στοχεύει εφαρμογές που διαχειρίζονται δεδομένα των μορφών που περιγράφηκαν και οι ακόλουθες βασικές κατηγορίες διακρίνονται από την οπτική που αφορά τις απαιτήσεις τους ως προς την υποκείμενη αρχιτεκτονική που τις φιλοξενεί.

- Εφαρμογές που παραδοσιακά εκτελούνται σε κεντρικοποιημένες αρχιτεκτονικές και απαιτείται να επεκταθούν και να προσαρμοστούν, ώστε να φιλοξενηθούν σε κατανεμημένες αρχιτεκτονικές.

- Εφαρμογές που είναι κατανεμημένες εκ φύσεως, όπως για παράδειγμα διάφορες διαδικτυακές και επιστημονικές εφαρμογές.

Κατά το σχεδιασμό ενός κατανεμημένου συστήματος για τη διαχείριση δεδομένων, διάφορα ζητήματα πρέπει να αντιμετωπιστούν, τα οποία συνήθως προκύπτουν από τη φύση της ίδιας της εφαρμογής. Δύο σημαντικοί παράγοντες που εξετάζονται σε αυτήν τη διατριβή είναι η δομή των δεδομένων και οι τύποι των ερωτημάτων που επιλύονται από το προκύπτον σύστημα. Η δομή των δεδομένων σχετίζεται με την οργάνωση τους στους συμμετέχοντες κόμβους και τον τρόπο που αποθηκεύονται σε αυτούς. Ο τύπος των ερωτημάτων υποδεικνύει τις στρατηγικές επεξεργασίας τους και την επιπρόσθετη πληροφορία που χρειάζεται να διατηρηθεί. Για παράδειγμα, η επίλυση "top-k" ερωτημάτων απαιτεί επιπρόσθετη πληροφορία για την κατάταξη των πιθανών απαντήσεων, ώστε να υπολογιστεί το τελικό αποτέλεσμα.

Μερικές ουσιώδεις ερωτήσεις προς διερεύνηση σχετικά με τα παραπάνω ζητήματα κατά το σχεδιασμό ενός κατανεμημένου συστήματος, που επίσης εξετάζονται στις προσεγγίσεις που παρουσιάζονται σε αυτή τη διατριβή είναι οι ακόλουθες: *Ποια διάταξη είναι πιο αποδοτική για την οργάνωση των διαθέσιμων κόμβων και τη μεταξύ τους επικοινωνία; Πως κατανέμονται τα δεδομένα και διανέμονται στους συμμετέχοντες κόμβους; Πως θα γίνεται η πρόσβαση στα δεδομένα ώστε να επιτυγχάνεται η εύκολη αναζήτηση και επαναχρησιμοποίηση τους; Πως μπορούν να ενοποιηθούν δεδομένα από διαφορετικές πηγές; Πως μπορούν τα δεδομένα να δεικτοδοτηθούν αποδοτικά λαμβάνοντας υπόψη τις ιδιότητες των δομών τους, τις μεταξύ τους σχέσεις και τις ερωτήσεις που στοχεύει το σύστημα; Ποιοι τύποι ερωτημάτων επιλύονται από το σύστημα προς ανάπτυξη; Ποια στρατηγική είναι πιο κατάλληλη για αποδοτική επεξεργασία ερωτημάτων; Πόσο συχνά ενημερώνονται τα δεδομένα και πόση επεξεργασία είναι αποδεκτή κατά τη διάρκεια των ενημερώσεων; Υπάρχουν θέματα συγχρονισμού που πρέπει να ληφθούν υπόψη κατά τη διάρκεια των ενημερώσεων;*

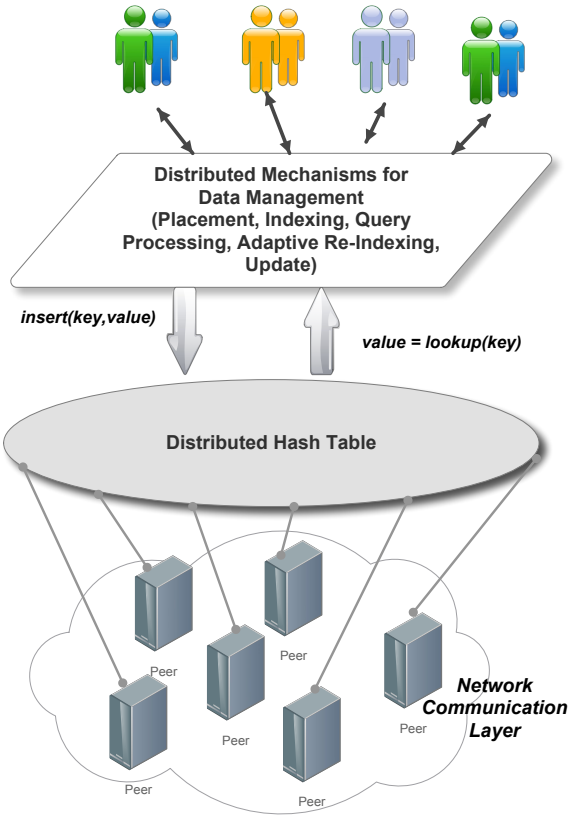
Το πλαίσιο επικοινωνίας των συστημάτων που προτείνονται στη παρούσα διατριβή βασίζεται στη χρήση DHT επικαλύψεων. Πολλές ερευνητικές δουλειές τονίζουν το γεγονός ότι οι P2P τεχνολογίες μπορούν να προσφέρουν ανταγωνιστικές λύσεις για το σχεδιασμό κατανεμημένων συστημάτων για απαιτητικές εφαρμογές ως προς τη διαχείριση των δεδομένων ([LCP+05], [ATS04],κλπ.). Οι υλοποιήσεις των συστημάτων που παρουσιάζονται σε αυτή τη διατριβή δεν

εξαρτώνται από ένα συγκεκριμένο DHT πρωτόκολλο, καθώς οποιαδήποτε υλοποίηση που παρέχει τις βασικές λειτουργίες *εισαγωγής (insert)* και *εντοπισμού (lookup)* αντικειμένων μπορεί να χρησιμοποιηθεί. Η επικάλυψη αυτή χρησιμοποιείται ως το βασικό επίπεδο για την οργάνωση των κόμβων που συμμετέχουν στο σύστημα με ένα συνεπή τρόπο και για τη δρομολόγηση των μηνυμάτων. Οι DHT επικαλύψεις επιλέχθηκαν λόγω της ικανότητας τους να οργανώνουν τους κόμβους με συνεπή τρόπο που προσδιορίζεται από το πρωτόκολλο που χρησιμοποιείται καθώς και της ιδιότητας τους να επιτυγχάνουν λογαριθμικό κόστος αναζήτησης ως προς τον αριθμό των κόμβων. Για αυτό το λόγο, οι αναζητήσεις σε DHTs επικαλύψεις δεν έχουν ως συνέπεια μεγάλη κατανάλωση εύρους ζώνης και οι DHTs επικαλύψεις δεν αντιμετωπίζουν εσωτερικούς περιορισμούς από την έλλειψη δομής. Το Σχήμα 1.1 παρουσιάζει μία γενική αφαιρετική επισκόπηση των επιπέδων των αρχιτεκτονικών που παρουσιάζονται στη διατριβή μου. Οι κόμβοι οργανώνονται σε μία DHT επικάλυψη, πάνω από την οποία υλοποιούνται οι λειτουργίες που υποστηρίζονται για τη διαχείριση των αποθηκευμένων δεδομένων που εξετάζονται.

Η κατασκευή μίας DHT επικάλυψης συνεπάγεται ότι τα αντικείμενα καταμερίζονται στους κόμβους βάσει μίας συγκεκριμένης λογικής που καθορίζεται από το αντίστοιχο πρωτόκολλο. Σε κάθε αντικείμενο (data item) αντιστοιχίζεται ένα αναγνωριστικό που ονομάζεται *κλειδί* (*key*), το οποίο στη συνέχεια χρησιμοποιείται για τον καθορισμό του κόμβου που είναι υπεύθυνος για αυτό το αντικείμενο. Εάν το κλειδί του αντικειμένου είναι γνωστό κατά τη προσπάθεια εντοπισμού του, τότε το αντικείμενο μπορεί να εντοπιστεί εύκολα στην επικάλυψη. Παρόλα αυτά, η συγκεκριμένη ιδιότητα των DHT επικαλύψεων εισάγει και κάποιους περιορισμούς. Σε πιο περίπλοκους τύπους ερωτημάτων, τα κλειδιά που αναζητούνται μπορεί να μην είναι γνωστά εκ των προτέρων και να ανακαλύπτονται κατή τη διάρκεια της αναζήτησης. Όσο πιο πολύπλοκα γίνονται τα ερωτήματα, επιπρόσθετοι μηχανισμοί που επεκτείνουν τις βασικές λειτουργίες των DHT απαιτούνται.

Οι DHT επικαλύψεις βασίζονται στην τυχαία ανάθεση των αντικειμένων ώστε να μην παρατηρείται το φαινόμενο κόμβων με υψηλό φόρτο. Ωστόσο η τυχαία ανάθεση των δεδομένων μπορεί να επηρεάσει αρνητικά την απόδοση και το κόστος επίλυσης κάποιων τύπων ερωτημάτων. Για παράδειγμα, τα ερωτήματα εύρους τιμών (range queries) απαιτούν μία διάταξη των αντικειμένων, ώστε να αποφευχθεί η διαδοχική και ξεχωριστή επερώτηση κάθε τιμής μέσα στο εύρος, γεγονός που θα έχει σαν αποτέλεσμα μεγάλη κατανάλωση εύρους ζώνης. Η ίδια παρατήρηση ισχύει για την περίπτωση των ερωτημάτων συνάθροισης (aggregate queries), όπου η συνάρτηση συνάθροισης υπολογίζεται επί ενός συνόλου τιμών που έχουν ομαδοποιηθεί από μία συγκεκριμένη τιμή μίας ιδιότητας ή μία ιδιότητα. Σε αυτήν την περίπτωση, η δομή των δεδομένων μπορεί να βοηθήσει σε πιο αποδοτική τοποθέτηση και δεικτοδότηση τους, δηλαδή λαμβάνοντας υπόψη κάποιο χαρακτηριστικό της δομής τους να ομαδοποιηθούν και να ανατεθούν στον ίδιο κόμβο.

Η αντιμετώπιση αυτή μπορεί να συνεισφέρει σημαντικά στην περίπτωση των ερωτημάτων συνάθροισης (ή σύνοψης) και εύρους τιμών, οι οποίες είναι οι κύριες κατηγορίες ερωτήσεων που μελετώνται στη συγκεκριμένη διατριβή εκτός από της ερωτήσεις για κάποια συγκεκριμένη τιμή.



Σχήμα 1.1: *Μία γενική επισκόπηση των επιπέδων των αρχιτεκτονικών που παρουσιάζονται στη συγκεκριμένη διατριβή*

Σε γενικές γραμμές, η χρήση των συναρτήσεων συνάθροισης σε ερωτήματα αποτελεί μία δημοφιλή και αποτελεσματική τακτική, ειδικά για τη λήψη αποφάσεων όπου απαιτείται η πρόσβαση, ο συνδυασμός και η σύνοψη των δεδομένων [WJOT09]. Η χρησιμότητα των συναρτήσεων συνάθροισης μπορεί να επηρεαστεί από το γεγονός ότι απαιτείται αρκετή επεξεργασία για τον υπολογισμό τους. Η επίλυση ενός τέτοιου ερωτήματος απαιτεί πολλές φορές να ανακτηθούν πολλά αντικείμενα από μία βάση δεδομένων για τον υπολογισμό της συνάρτησης συνάθροισης. Η αύξηση του όγκου των αποθηκευμένων δεδομένων έχει ως συνέπεια την επιμήκυνση το χρόνου και την αύξηση της πολυπλοκότητας επεξεργασίας των ερωτημάτων και για αυτό το λόγο δε χρησιμοποιούνται κεντρικοποιημένες λύσεις αλλά κυρίως παράλληλες και κατανεμημένες.

Ο μεγάλος όγκος των δεδομένων καθιστά τη δεικτοδότηση όλων των αποθηκευμένων τιμών και των μεταξύ τους συνδυασμών μία “δαπανηρή” διαδικασία. Ειδικά όσον αφορά τα πολυδιάστατα δεδομένα, ο αριθμός όλων των πιθανών συνδυασμών αυξάνει εκθετικά με τον αριθμό των

διαστάσεων. Η ιεραρχική δομή των δεδομένων υποστηρίζει εκ της κατασκευή της τη σύνοψη των τιμών και η συγκεκριμένη ιδιότητα αξιοποιείται για την τοποθέτηση των αντικειμένων που περιγράφονται από την ίδια τιμή στον ίδιο κόμβο. Η επιλογή ενός επιπέδου για να δεικτοδοτηθεί μπορεί να αποδειχθεί πολύ αποτελεσματική, ειδικά αν το επίπεδο που δεικτοδοτείται προσαρμόζεται στις απαιτήσεις των χρηστών. Η λύση αυτή υιοθετείται πλήρως σε αυτήν τη διατριβή: η δεικτοδότηση είναι ιδιαίτερα προσαρμοστική στις τάσεις που παρατηρούνται στα ερωτήματα, ενώ λαμβάνει χώρα online χωρίς να απαιτούνται πολύπλοκες διαδικασίες επεξεργασίας ή η επαναεισαγωγή όλων των δεδομένων από την αρχή της λειτουργίας του συστήματος.

Ως συνέπεια της επιλογής ενός μόνο επιπέδου της ιεραρχίας, ο εντοπισμός τιμών της DHT επικάλυψης μπορεί να εκτελεστεί μόνο για τις τιμές που αντιστοιχούν σε αυτό το επίπεδο, διαφορετικά το ερώτημα πρέπει να προωθηθεί σε όλους τους κόμβους του δικτύου. Για αυτό το λόγο, επιπρόσθετες δομές δεικτοδότησης απαιτούνται για την αποφυγή της "πλημμύρας" και ταυτόχρονα να επιβαρύνουν το σύστημα με λογικό κόστος για τη διατήρηση και την ενημέρωση τους. Λαμβάνοντας υπόψη ότι υπάρχουν επαναλαμβανόμενα μοτίβα στα ερωτήματα των εφαρμογών που θεωρούνται (όπως διαδικτυακές και OLAP εφαρμογές), η προσαρμοστικότητα της δεικτοδότησης μπορεί να ωφελήσει την αποτελεσματικότητα του συστήματος. Για παράδειγμα, σε μία διαδικτυακή εφαρμογή όταν ένα αντικείμενο γίνεται δημοφιλές, τότε η πλειοψηφία των ερωτημάτων στοχεύει τις τιμές του επιπέδου αυτού. Σε αυτήν την περίπτωση, η δεικτοδότηση των δημοφιλών αντικειμένων είναι επαρκής για να επιταχύνει τον υπολογισμό των απαντήσεων και να μειώσει το κόστος επικοινωνίας. Ως εκ τούτου, έμφαση δίνεται στην ανάπτυξη των μηχανισμών για την προσαρμογή της δεικτοδότησης, εκτός από τη δημιουργία απλών δεικτών. Επίσης, αποτελεί στόχο η διατήρηση των ιδιοτήτων του DHT, ώστε να μη δημιουργηθούν μη αναμενόμενες παρενέργειες όπως η υπερφόρτωση κάποιων κόμβων ή επιστροφή μη ακριβών αποτελεσμάτων.

1.2 Συμβολή της Διατριβής

Σε αυτήν τη διατριβή προτείνονται καινοτόμες μεθοδολογίες που επιτρέπουν την υλοποίηση επεκτάσιμων μηχανισμών για την αποδοτική ενοποίηση, οργάνωση, δεικτοδότηση, αναζήτηση και ενημέρωση δεδομένων σε κατανεμημένα συστήματα ευρείας κλίμακας. Οι μηχανισμοί αυτοί συνδυάζονται και ενσωματώνονται στο σχεδιασμό πλήρως λειτουργικών συστημάτων που προορίζονται για τη διαχείριση δεδομένων που χαρακτηρίζονται από ιεραρχίες, πολλαπλές διαστάσεις και σημασιολογική πληροφορία που τα περιγράφει. Τα κίνητρα για την ύπαρξη των προτεινόμενων συστημάτων καθορίζονται από πραγματικές εφαρμογές που θεωρήθηκαν και για τις οποίες τα συστήματα αυτά προσαρμόστηκαν και αξιολογήθηκαν.

Οι γενικές αρχές που ενοποιούν τα συστήματα που προτείνονται στη διατριβή είναι:

- Οι μηχανισμοί τοποθέτησης και δεικτοδότησης επικεντρώνονται σε ιεραρχικά και πολυδιάστατα δεδομένα, ενώ μπορούν να υποστηριχτούν δομημένα και ημι-δομημένα δεδομένα με αποδοτικό τρόπο.

- Οι τύποι των ερωτημάτων που μελετώνται, όπως aggregate ερωτήματα και ερωτήματα εύρους τιμών απαιτούν πιο πολύπλοκη επεξεργασία και για αυτό το λόγο δημιουργούνται πιο σύνθετες λειτουργίες από αυτές για τον απλό εντοπισμό τιμών.

- Κατανεμημένοι και αποτελεσματικοί μηχανισμοί δεικτοδότησης αναπτύσσονται σύμφωνα με τη δομή των δεδομένων και τους τύπους των ερωτημάτων που στοχεύουν.

- Προτείνονται ευέλικτοι μηχανισμοί που επιτυγχάνουν την προσαρμογή της δεικτοδότησης σε κατανεμημένες υποδομές.

- Η επεξεργασία των δεδομένων για τη δημιουργία και την ενημέρωση των δομών δεικτοδότησης δεν απαιτεί διεργασίες που πρέπει να γίνουν αφού διακοπεί η λειτουργία του συστήματος.

- Επιδιώκεται η επεκτασιμότητα των προτεινόμενων συστημάτων όσο αυξάνεται ο όγκος των δεδομένων και ο αριθμός των πόρων.

Οι κύριες συνεισφορές της διατριβής μου απαριθμούνται ως εξής:

- Αποδοτικές μέθοδοι προτείνονται για την οργάνωση, δεικτοδότηση, αναζήτηση και ενημέρωση δεδομένων που ακολουθούν εννοιολογικές ιεραρχίες. Η τοποθέτηση των δεδομένων σε κόμβους γίνεται με τέτοιο τρόπο ώστε να διατηρείται η σημασιολογική πληροφορία που περιλαμβάνεται στις εννοιολογικές ιεραρχίες. Η προσέγγιση για τη δεικτοδότηση που εφαρμόζεται αυτό-προσαρμόζεται στα εισερχόμενα ερωτήματα και ευέλικτοι μηχανισμοί προτείνονται για τη πραγματοποίηση μηχανισμών επαναδεικτοδότησης που ενεργοποιούνται ανάλογα με τα ερωτήματα. Το σύστημα που προκύπτει είναι ικανό να αντιλαμβάνεται τις τάσεις στα ερωτήματα που θέτονται και παρουσιάζει ιδιαίτερα προσαρμοστική συμπεριφορά μεταβάλοντας τη δεικτοδότηση με αυτόματο τρόπο, ώστε να αυξηθεί η απόδοση και να μειωθεί η καθυστέρηση κατά την απάντηση των ερωτημάτων σε διάφορα επίπεδα σύνοψης. Οι κόμβοι που συμμετέχουν στο σύστημα αποφασίζουν μεμονωμένα για το επίπεδο της δεικτοδότησης ανάλογα με τα εισερχόμενα ερωτήματα και εκτελούν τις απαιτούμενες διαδικασίες προσαρμογής της δεικτοδότησης μόνο για τα τμήματα του συνόλου των δεδομένων, στα οποία κρίνεται ότι θα έχουν θετικά αποτελέσματα. Επιπλέον, μία μέθοδος για online ενημερώσεις των ιεραρχικών δεδομένων προτείνεται. Η δυσκολία που πρέπει να αντιμετωπιστεί είναι ότι η κατανομή των αντικειμένων σε πολλούς κόμβους

και η έλλειψη κεντρικοποιημένων δομών απαιτούν επιπρόσθετη προσπάθεια για να αποφασιστεί σε ποιο κόμβο πρέπει να καταλήξει ένα αντικείμενο και ποια πληροφορία που διατηρείται πρέπει να ενημερωθεί. Για αυτό το λόγο εισάγεται ένας κατανεμημένος κατάλογος για την αποθήκευση αυτού του είδους της πληροφορίας. Όλες αυτές οι μέθοδοι έχουν συνδυαστεί και υλοποιηθεί σε ένα πλήρως λειτουργικό σύστημα που χρησιμοποιεί ως βάση του μία δομημένη P2P επικάλυψη, ώστε να αξιολογηθεί η αποτελεσματικότητα του και η απόδοση που επιτυγχάνεται.

- Η χρησιμότητα αυτών των μεθόδων παρακινείται από την προσαρμογή τους σε ένα σύστημα που μπορεί να εξυπηρετήσει ως ένα κατανεμημένο Πληροφοριακό Σύστημα (Information System) υψηλών επιδόσεων. Το περιγραφόμενο σύστημα δημιουργεί, επερωτά και ενημερώνει εγγραφές που περιέχουν πληροφορία για την κατάσταση μίας υποδομής Πλέγματος (όπως για παράδειγμα τον αριθμό διαθέσιμων υπολογιστικών μονάδων, τον αποθηκευτικό χώρο,κτλ.) και μπορεί να αποτελέσει μία βιώσιμη λύση σε σχέση με τα παραδοσιακά πληροφοριακά συστήματα για τέτοιου είδους υποδομές. Πειραματικά αποτελέσματα για μία πραγματική εφαρμογή accounting που χρησιμοποιεί το Grid Information Service δείχνουν ότι η προτεινόμενη κατανεμημένη αρχιτεκτονική μπορεί να είναι αποδοτικότερη, διότι δεν απαιτεί offline επεξεργασία και εξαλείφει σημεία που μπορούν να προκαλέσουν προβλήματα στην απόδοση.

- Ένα πλήρως λειτουργικό σύστημα προτείνεται για την αποτελεσματική κατανομή και επεξεργασία δεδομένων που περιγράφονται από πολλαπλές ιδιότητες, δηλαδή πολλαπλές διαστάσεις, οι οποίες δομούνται με τη σειρά τους από εννοιολογικές ιεραρχίες. Ένα σημαντικό χαρακτηριστικό είναι ότι οι μηχανισμοί που περιγράφονται δεν εξαρτώνται από προκαθορισμένα, αυστηρά σχήματα και υποστηρίζουν μερικώς δομημένα δεδομένα χωρίς να απαιτούν κόμβους - "μεσολαβητές" για τη μετάφραση των σχημάτων. Για την αποδοτική επίλυση των ερωτημάτων, το περιγραφόμενο σύστημα χρησιμοποιεί μία "εννοιολογική" αλυσίδα δαχτυλιδιών που αποθηκεύει την πληροφορία. Η προσαρμοστικότητα του συστήματος διατηρείται ως ένα από τα καινοτόμα χαρακτηριστικά του. Εκτός από τους μηχανισμούς για την προσαρμογή της δεικτοδότησης, μία άλλη ιδιότητα που διαφοροποιεί το σύστημα αυτό είναι η προτεινόμενη στρατηγική για τον προϋπολογισμό συνδυασμών από ερωτηθέντες τιμές για τη μερική υλοποίηση μερικών όψεων για μελλοντική χρήση.

- Οι τεχνικές που προτείνονται για τη διαχείριση των πολυδιάστατων, ιεραρχικών δεδομένων εφαρμόζονται για το σχεδιασμό ενός συστήματος για την αποθήκευση και αναζήτηση σημασιολογικών δεδομένων. Μελετάται το σενάριο χρήσης των ημιδομημένων δεδομένων που χρησιμοποιούνται για τη δημοσίευση ενός μεγάλου αριθμού συνόλων δεδομένων. στο Διαδίκτυο, δηλαδή μελετώνται τα *Linked Data*. Το σύστημα που προκύπτει είναι μία

κατανεμημένη πλατφόρμα που υπηρετεί τις ανάγκες για αποθήκευση, δεικτοδότηση και επερώτηση δεδομένων που ακολουθούν τη μορφή των Linked Data.

1.3 Οργάνωση της Διατριβής

Το υπόλοιπο της διατριβής είναι οργανωμένο ως εξής:

Στο *Κεφάλαιο 2* παρουσιάζεται μία επισκόπηση των υπαρχόντων μεθόδων δεικτοδότησης και αναζήτησης σε κατανεμημένα συστήματα. Επίσης, περιγράφεται μία τεχνική δεικτοδότησης που βασίζεται σε Καμπύλες Πλήρωσης του Χώρου για πολυδιάστατα δεδομένα, η οποία χρησιμοποιείται στην υλοποίηση μίας κατανεμημένης και επεκτάσιμης υπηρεσίας για διαχείριση μεταδεδομένων σε υποδομές Πλέγματος.

Το *Κεφάλαιο 3* προτείνει προσαρμοστικές μεθόδους για την κατανομή και αναζήτηση δεδομένων που περιγράφονται από εννοιολογικές ιεραρχίες. Οι τύποι των ιεραρχικών δεδομένων μελετώνται και παρουσιάζονται μεθοδολογίες για την αποθήκευση, δεικτοδότηση, επερώτηση και ενημέρωση τέτοιου είδους δεδομένων σε ένα σύστημα που χρησιμοποιεί μία DHT επικάλυψη. Ιδιαίτερη προσοχή δίνεται στους αλγορίθμους για τη λήψη των αποφάσεων σχετικά με τις λειτουργίες προσαρμογής της δεικτοδότησης ώστε να προσαρμοστεί το επίπεδο λεπτομέρειας που δεικτοδοτείται κατάλληλα. Επίσης παρουσιάζεται η υλοποίηση των προτεινόμενων μεθοδολογιών και περιγράφεται ο τρόπος με τον οποίο αλληλεπιδρούν σε ένα σύστημα που στοχεύει στη διαχείριση ιεραρχικών δεδομένων. Επιπλέον, διερευνάται η περίπτωση χρήσης των μεθόδων αυτών σε ένα κατανεμημένο Πληροφοριακό Σύστημα για υποδομές Πλέγματος και αναλύεται πως αυτή η εφαρμογή μπορεί να υλοποιηθεί με τη χρήση ενός συστήματος με τις προηγούμενες ιδιότητες. Το Κεφάλαιο αυτό περιέχει μία αναλυτική πειραματική αξιολόγηση της απόδοσης ενός τέτοιου συστήματος τόσο για συνθετικά δεδομένα και όσο και για πραγματικά δεδομένα, τα οποία προέρχονται από ένα πραγματικό Πληροφοριακό Σύστημα μίας υπάρχουσας υποδομής Πλέγματος.

Το πρόβλημα των μερικώς δομημένων, πολυδιάστατων δεδομένων εξερευνάται στο *Κεφάλαιο 4*. Η θεώρηση ότι τα δεδομένα προς αποθήκευση μπορούν να οργανωθούν με ιεραρχικό τρόπο εξακολουθεί να ισχύει. Ένα αρχικό ζήτημα που αντιμετωπίζεται είναι η κατανομή και τοποθέτηση τέτοιων δεδομένων στους διαθέσιμους κόμβους μίας επικάλυψης, ώστε να είναι δυνατή η επίλυση ερωτημάτων που περιέχουν τιμές σε μία ή περισσότερες διαστάσεις. Οι μέθοδοι για την επίλυση ερωτημάτων σημείων και των aggregate ερωτημάτων αναλύονται. Οι απαραίτητες τροποποιήσεις για τις λειτουργίες re-indexing περιγράφονται ώστε να μπορούν να εκτελεστούν σε ένα σύστημα, όπου οι διαφορετικές διαστάσεις δεικτοδοτούνται μεμονωμένα αλλά ταυτόχρονα διατηρούνται οι μεταξύ τους διασυνδέσεις. Επίσης, σε αυτό το Κεφάλαιο διερευνάται η μερική υλοποίηση όψεων ανάλογα με τα ερωτήματα, που βασίζεται στον υπολογισμό των πιθανών συνδυασμών τιμών που περιέχονται σε ένα ερώτημα εκ των προτέρων. Το Κεφάλαιο αυτό συνεχίζει

με τη μελέτη του παραδείγματος των Διασυνδεμένων Δεδομένων. Στη συνέχεια ακολουθεί η περιγραφή της αρχιτεκτονικής ενός συστήματος για τη διαχείριση ημι-δομημένων διαδικτυακών δεδομένων που δημοσιεύονται από ετερογενείς πόρους και περιγράφονται από οντολογίες στη μορφή των Διασυνδεμένων Δεδομένων. Η απόδοση και των δύο συστημάτων αξιολογείται στις αντίστοιχες Ενότητες του συγκεκριμένου Κεφαλαίου.

Το *Κεφάλαιο 5* περιλαμβάνει τα συμπεράσματα που προέκυψαν από τη διατριβή και περιγράφει περιληπτικά μερικές πιθανές προεκτάσεις των θεμάτων που μελετήθηκαν.

ΚΕΦΑΛΑΙΟ 2

Διαχείριση Δεδομένων με χρήση Τεχνολογιών Ομότιμων Κόμβων

Οι τεχνολογίες P2P έχουν χρησιμοποιηθεί ευρέως ως λύσεις για τη δημιουργία κατανεμημένων συστημάτων που προσφέρουν επεκτασιμότητα και ανεκτικότητα σε σφάλματα. Στις υπάρχουσες εφαρμογές, η πιο δημοφιλής χρήση τους είναι για τη δημιουργία συστημάτων διαμοιρασμού αρχείων παγκόσμιας κλίμακας. Ωστόσο, η υιοθέτηση των P2P τεχνικών έχει μελετηθεί εκτενώς από την ερευνητική κοινότητα και σε άλλους τύπους εφαρμογών. Διάφορες εργασίες επικεντρώνονται στην εφαρμογή των τεχνικών αυτών σε κατανεμημένες βάσεις δεδομένων και κατανεμημένα συστήματα δεικτοδότησης και αναζήτησης δεδομένων. Σε αυτό το κεφάλαιο παρουσιάζεται μία επισκόπηση των υπαρχόντων P2P συστημάτων που επικεντρώνεται κυρίως στις δομημένες P2P επικαλύψεις. Επίσης αναλύονται οι βασικές έννοιες των δομημένων / αδόμητων P2P επικαλύψεων και τα υπάρχοντα πρωτόκολλα, ενώ δίνεται ιδιαίτερη έμφαση στα DHTs. Άλλα ζητήματα που περιγράφονται σε αυτό το κεφάλαιο είναι οι διάφορες προτεινόμενες τεχνικές για συστήματα δεικτοδότησης που βασίζονται σε DHTs καθώς και η κατανεμημένη δεικτοδότηση πολυδιάστατων δεδομένων. Επιπρόσθετα, παρέχεται και μία επισκόπηση των βασικών εννοιών των Τεχνολογιών Πλέγματος. Τέλος, προτείνεται μία τεχνική δεικτοδότησης που βασίζεται σε Καμπύλες Πλήρωσης του Χώρου (Space Filling Curves) για τη δεικτοδότηση μεταδεδομένων (metadata) που περιγράφουν σχολιασμένο περιεχόμενο (annotated content).

2.1 Επικαλύψεις Ομότιμων Κόμβων

Οι *τεχνολογίες ομότιμών κόμβων* (*Peer-to-peer computing*) έχουν γίνει αποδεκτές ευρέως ως μία συμπαγής, ευκόλως αναπτυσσόμενη και αυτό-οργανούμενη λύση που μπορεί να χρησιμοποιηθεί για τη δημιουργία πλήρως κατανεμημένων και επεκτάσιμων συστημάτων. Η αυτό-οργάνωση των P2P δικτύων αποδεικνύεται από την ικανότητα τους "να προσαρμόζονται αυτόματα στις αφίξεις, απομακρύνσεις και αποτυχίες των κόμβων" [RD01]. Ένα άλλο σημαντικό χαρακτηριστικό αυτών των συστημάτων είναι η έλλειψη κεντρικών δομών (centralized structures) που ελέγχουν τις κρίσιμες διαδικασίες που εκτελούνται κατά τη λειτουργία τους. Κάθε κόμβος που συμμετέχει στην επικάλυψη αναφέρεται ως "*ομότιμός κόμβος*" (*peer*) και μπορεί να συμπεριφέρεται είτε ως πελάτης (client) είτε ως εξυπηρετητής (server) με αυτόνομο τρόπο. Τα peers συνεισφέρουν στο σύστημα τους πόρους που διαθέτουν, οι οποίοι χρησιμοποιούνται με συνεργατικό τρόπο για την εκτέλεση λειτουργιών ώστε να επιτυγχάνεται κατανεμημένη επεξεργασία και αποθήκευση. Μία P2P επικάλυψη είναι δυνατό να λειτουργεί χωρίς επίβλεψη και κεντρικό συντονισμό και να προσαρμόζεται δυναμικά σε απρόσμενες αφίξεις και αναχωρήσεις κόμβων. Αν και υπάρχουν πολλαπλές υλοποιήσεις πλήρως κατανεμημένων συστημάτων, στη βιβλιογραφία έχουν προταθεί και κάποια P2P συστήματα με κεντρικές δομές, ειδικά κατά τις πρώτες φάσεις της εμφάνισης των P2P τεχνολογιών. Σε αυτές τις κεντρικοποιημένες P2P προσεγγίσεις, ένας κεντρικός server (ή πολλαπλές υποστάσεις αυτού του server) χρησιμοποιείται (χρησιμοποιούνται) για τις λειτουργίες του συστήματος που είναι ζωτικής σημασίας για αυτό. Για παράδειγμα, μία τέτοια κεντρική δομή παρέχει την υπηρεσία καταλόγου με τις τοποθεσίες των αποθηκευμένων δεδομένων στην περίπτωση του Napster [SGG03]. Η δομή αυτή ερωτάται για την εύρεση των κόμβων που είναι υπεύθυνοι για τα δεδομένα προς ανάκτηση. Παρόλα αυτά, η ύπαρξη τέτοιων κεντρικών σημείων για τη διατήρηση πληροφορίας που αφορά το σύνολο της υποδομής συνήθως οδηγεί σε μη επεκτάσιμα συστήματα, δεδομένου ότι όλη τη κίνηση του δικτύου περνάει από ένα κόμβο, γεγονός που τον καθιστά πιθανό σημείο αποτυχίας (single point of failure). Σε γενικές γραμμές, τα κεντρικά σημεία που συγκεντρώνουν κρίσιμη πληροφορία πρέπει να αποφεύγονται σε κατανεμημένες αρχιτεκτονικές, εφόσον αυτό είναι εφικτό.

Οι κατανεμημένες εφαρμογές που εκτελούνται σε αρχιτεκτονικές τύπου P2P αποτελούν ένα ενεργό ερευνητικό πεδίο κυρίως λόγω του χαρακτηριστικού της επεκτασιμότητας που απορρέει από την υιοθέτηση του αποκεντρωμένου μοντέλου ελέγχου των P2P συστημάτων. Η πιο δημοφιλής κατηγορία εφαρμογών που στηρίζεται σε P2P δίκτυα επικεντρώνεται στο διαμοιρασμό αρχείων σε γεωγραφικά κατανεμημένα συστήματα ευρείας κλίμακας. Κάποια αντιπροσωπευτικά παραδείγματα εφαρμογών αυτού του τύπου είναι τα εξής: Gnutella2 [Gnu], Kazaa [KaZ], Bittorent [Bit], eMule [eMu], iMesh [iMe]. Ωστόσο, μερικά από τα αξιόλογα χαρακτηριστικά που επιδεικνύουν τα P2P συστήματα, όπως η ανεπίβλεπτη λειτουργία τους, η δυναμική προσαρμογή τους στις αλλαγές της τοπολογίας, η επεκτασιμότητα τους και η ανεκτικότητα τους

σε σφάλματα καθιστούν τις P2P τεχνολογίες κατάλληλες για την ανάπτυξη πολλών διαδικτυακών εφαρμογών και εφαρμογών απαιτητικών ως προς τη διαχείριση των δεδομένων. Στις περισσότερες από αυτές τις περιπτώσεις, η προσέγγιση που ακολουθείται είναι η κατασκευή μίας P2P επικάλυψης πάνω από τη φυσική τοπολογία του δικτύου, η οποία χρησιμοποιείται για την υλοποίηση διάφορων λειτουργιών που απαιτούνται από τις εφαρμογές των διάφορων τομέων. Για παράδειγμα, οι P2P τεχνολογίες έχουν χρησιμοποιηθεί για τη δημιουργία δικτύων για διανομή περιεχομένου (content delivery) και συνεχή ροή δεδομένων (streaming), για τη σύσταση δικτύων κοινωνικής δικτύωσης, για την παροχή κατανεμημένων μηχανών αναζήτησης και για τη δημιουργία δικτύων επικοινωνίας. Το Skype [Sky] είναι μία από τις πιο γνωστές P2P εφαρμογές στον τομέα των επικοινωνιών που χρησιμοποιείται από εκατομμύρια χρήστες.

Για την επίτευξη μίας συγκεκριμένης εργασίας σε μία P2P επικάλυψη απαιτείται η επικοινωνία ενός peer με άλλους που επιτυγχάνεται με την ανταλλαγή μηνυμάτων μέσω των διαθέσιμων διαύλων επικοινωνίας, οι οποίοι αναπαριστώνται ως *σύνδεσμοι* (*links*). Ένα σημαντικό ερώτημα που τίθεται σχετίζεται με τον τρόπο οργάνωσης των peers και των μεταξύ τους links που θα δημιουργηθούν. Αν και υπάρχουν ποικίλες υλοποιήσεις για P2P επικαλύψεις όπως αναφέρεται στις μελέτες [LCP+05] και [ATS04], δύο βασικές κλάσεις P2P επικαλύψεων ξεχωρίζουν: Αδόμητες (Unstructured) και Δομημένες (Structured) επικαλύψεις. Οι *αδόμητες επικαλύψεις* δεν απαιτούν τη οργάνωση των συνδέσμων σύμφωνα με μία συγκεκριμένη δομή [KXZ05], ενώ δεν υπάρχουν κεντρικοί κατάλογοι ή κάποια άλλη μορφή ελέγχου επί της τοπολογίας του δικτύου ή της τοποθέτησης των αρχείων [LCC+02]. Οι αδόμητες επικαλύψεις βασίζονται κυρίως σε τυχαίους αλγορίθμους για την κατασκευή της τοπολογίας τους. Σε αυτές τις προσεγγίσεις, κάθε peer διατηρεί μία λίστα των γειτόνων του και η προκύπτουσα δικτυακή επικάλυψη προσομοιάζει ένα τυχαίο γράφο. Όπως περιγράφεται στην εργασία [RM06], ο εντοπισμός (lookup) ενός αποθηκευμένου αντικειμένου καταλήγει σε "πλημμύρα" του αντίστοιχου αιτήματος στο δίκτυο. Αναλυτικότερα, ένα τέτοιο ερώτημα προωθείται από έναν κόμβο στο γείτονα του, ενώ κάθε κόμβος που το λαμβάνει προχωράει στην αποτίμηση του τοπικά βάσει του περιεχομένου που αποθηκεύει. Για την αποφυγή της επιβάρυνσης σε μηνύματα και σε επεξεργασία από μη αναγκαία κίνηση έχουν αναπτυχθεί μέθοδοι που στοχεύουν στην περιορισμένη εκπομπή μηνυμάτων (broadcast) και μέθοδοι "τυχαίων περιπάτων" (random walks) [TR06]. Οι αδόμητες επικαλύψεις δεν επιβάλλουν τη διασύνδεση της τοπολογίας και των δεδομένων, αν και η τοπολογία που προκύπτει μπορεί να χαρακτηρίζεται από κάποιες ιδιότητες. Συνεπώς, η προσέγγιση αυτή παρουσιάζει μία αδυναμία στην εξυπηρέτηση ερωτημάτων για δεδομένα που δεν εμφανίζουν πολλαπλά αντίγραφα; τα ερωτήματα αυτά αποστέλλονται σε ένα μεγάλο αριθμό από peers και αυτό έχει ως αποτέλεσμα την υπερβολική κατανάλωση εύρους ζώνης. Επιπλέον, δεν υπάρχουν εγγυήσεις ότι θα βρεθούν μη δημοφιλή δεδομένα ή δεδομένα που αποθηκεύονται σε απομακρυσμένους κόμβους λόγω του περιορισμού του "ορίζοντα" αναζήτησης που επιβάλλεται, ενώ ο εντοπισμός σχετικών αντικειμένων μπορεί να αποτελέσει πρόβλημα όσο αυξάνεται το μέγεθος του δικτύου. Μία *υβριδική*

(*hybrid*) προσέγγιση μπορεί να ακολουθηθεί σε αυτήν την κατηγορία επικαλύψεων, σύμφωνα με την οποία θεωρούνται κάποιοι peers ως "ισχυρότεροι" και αποκαλούνται *superpeers*. Οι peers αυτοί συμπεριφέρονται ως "brokers" στους οποίους προσκολλούνται τα κανονικά peers, όταν εντάσσονται στο δίκτυο. Όλοι η επικοινωνία από και προς ένα κανονικό peer γίνεται μέσω του superpeer με τον οποίο συσχετίζεται.

Οι *δομημένες επικαλύψεις* επιβάλλουν μία "αυστηρή" τοπολογία, δηλαδή το σύνολο των συνδέσεων μεταξύ των μελών του P2P δικτύου ελέγχεται από το πρωτόκολλο που ακολουθείται, ενώ τα δεδομένα δεν κατανέμονται με τυχαίο τρόπο στους κόμβους, αλλά τοποθετούνται σε συγκεκριμένες τοποθεσίες, ώστε να καταστήσουν ευκολότερη την επίλυση των ερωτημάτων που θα ακολουθήσουν [LCC+02]. Οι δομημένες επικαλύψεις είναι λιγότερο ευέλικτες και επιβάλλεται επιπρόσθετο κόστος για τη διατήρηση της τοπολογίας, ειδικά κατά την άφιξη και την αναχώρηση των κόμβων (π.χ. απαιτείται η ενημέρωση των πινάκων δρομολόγησης). Εντούτοις, η απόδοση των τεχνικών αναζήτησης βελτιώνεται, επειδή η επικάλυψη κατασκευάζεται σύμφωνα με μία ντετερμινιστική διαδικασία και η τοποθεσία ενός αντικειμένου μπορεί να προσδιοριστεί με ακρίβεια. Η πιο συνηθισμένη προσέγγιση που ακολουθείται για τη δημιουργία μίας δομημένης P2P επικάλυψης βασίζεται στους *Κατανεμημένους Πίνακες Κατακερματισμού* (*Distributed Hash Tables*), όπως περιγράφεται στην επόμενη Ενότητα.

2.2 Κατανεμημένοι Πίνακες Κατακερματισμού

Η βασική προσέγγιση που ακολουθείται και έχει ως αποτέλεσμα μία δομημένη επικάλυψη είναι η οργάνωση της επικάλυψης μέσω ενός *Distribute Hash Table* (*DHT*), δηλαδή ενός πίνακα κατακερματισμού (hash table) με εγγραφές που κατανέμονται στα διαφορετικά peers. Στο επίπεδο του DHT λαμβάνουν χώρα κάποιες λειτουργίες για τη δρομολόγηση των μηνυμάτων μεταξύ των κόμβων, ενώ παρέχεται μία διεπαφή γενικής χρήσης για ονοματοδοσία ανεξάρτητης της τοποθεσίας (location-independent naming) που μπορεί να αξιοποιηθεί από ποικίλες εφαρμογές.

Σε όλα τα αντικείμενα (items) – δηλαδή κόμβους και πόρους (π.χ. δεδομένα)- που εισάγονται σε ένα DHT ανατίθεται ένας μοναδικό αναγνωριστικό (identifier - ID), το οποίο αναφέρεται ως *κλειδί* (*key*). Στην πλειοψηφία των περιπτώσεων, το key ανατίθεται τυχαία από ένα πολυπληθή χώρο αναγνωριστικών (identifier space), όπως ένα χώρο των 128-bit ή των 160-bit. Ένα DHT πρωτόκολλο οδηγεί σε ένα αποτελεσματικό και ντετερμινιστικό σύστημα, όπου ένα αντικείμενο δεδομένου (data item) αντιστοιχίζεται με μοναδικό τρόπο στο identifier ενός κόμβου βάσει κάποιας μετρικής απόστασης [BKK+03], καθιστώντας το συγκεκριμένο κόμβο "*υπεύθυνο*" για αυτό το data item. Η αυστηρώς καθορισμένη δομή των DHTs επιτρέπει την αποτελεσματική ανακάλυψη των data items, εάν είναι γνωστά τα keys τους. Για τη διευκόλυνση της κατασκευής του

δικτύου και την επίτευξη της εξισορρόπησης του φορτίου δεδομένων, τα περισσότερα δομημένα P2P συστήματα χρησιμοποιούν ομοιόμορφες συναρτήσεις κατακερματισμού (hash functions) για την ανάθεση των identifiers στα peers και στους πόρους. Η χρήση των ομοιόμορφων συναρτήσεων κατακερματισμού εξασφαλίζει την τυχαία ανάθεση των identifiers στα αντικείμενα και συνεπώς την ομοιόμορφη κατανομή του φορτίου στο χώρο των κλειδιών (key-space).

Όταν αναζητείται κάποιο data item σε μία DHT επικάλυψη, επιστρέφεται η διεύθυνση δικτύου (network address) του κόμβου που είναι υπεύθυνος για το συγκεκριμένο data item, ενώ ο κόμβος αυτός εντοπίζεται βάσει των ιδιοτήτων του DHT, δεδομένου ότι οι peers δεν έχουν γνώση της συνολικής κατάστασης του δικτύου. Αυτό επιτυγχάνεται με τη δρομολόγηση ενός αιτήματος εντοπισμού (lookup) όλο και πιο κοντά προς τον υπεύθυνο κόμβο, μέχρι αυτός να βρεθεί. Ένας κόμβος διατηρεί κάποια πληροφορία (δηλαδή το NodeID του peer και την IP διεύθυνση) για ένα μικρό αριθμό κόμβων, οι οποίοι καλούνται "*γείτονες*" ("*neighbours*") στον *πίνακα δρομολόγησης* (*routing table*) του. Επίσης, κάθε κόμβος είναι υπεύθυνος για την αποθήκευση των keys και την διαχείριση των λειτουργιών για πόρους που είναι "κοντινοί" (βάσει κάποιας μετρικής απόστασης) με το ID του. Μία λειτουργία lookup μπορεί να ξεκινήσει από οποιονδήποτε κόμβο και τερματίζει στον κόμβο που είναι υπεύθυνος για το αναζητούμενο key. Ο αριθμός των peers που επισκέπτεται ένα lookup είναι μικρός και μία DHT επικάλυψη παρουσιάζει τις περισσότερες φορές λογαριθμικά μονοπάτια αναζήτησης (logarithmic search path lengths) σε σύγκριση με το μέγεθος του δικτύου, δηλαδή οποιοδήποτε data item μπορεί να εντοπιστεί σε $O(\log N)$ άλματα (hops) στην επικάλυψη, όπου N είναι ο αριθμός των peers που συμμετέχουν στο σύστημα. Η λογική κατά τη δρομολόγηση ενός lookup για ένα συγκεκριμένο key είναι ότι κάθε κόμβος που λαμβάνει ένα ερώτημα για ένα key πρέπει να δύναται να το δρομολογήσει σε έναν κόμβο με κοντινότερο ID σε αυτό, ώσπου ο υπεύθυνος κόμβος για το key να βρεθεί.

Στην εργασία [AAG+05] προτείνεται μία γενίκευση των υπαρχόντων προσεγγίσεων καθώς και ένα μοντέλο αναφοράς. Στην εργασία αυτή, αναγνωρίζονται επίσης οι βασικές απαιτήσεις που πρέπει να πληρούνται κατά τη δημιουργία μίας επικάλυψης: Ένα κατάλληλος χώρος αναγνωριστικών (identifier space) πρέπει να επιλεχθεί και να οριστεί μία μέθοδος για την αντιστοίχηση των πόρων και των peers σε αυτό το χώρο. Επιπλέον, πρέπει να οριστεί η δομή του και να εξασφαλιστεί η αποτελεσματική διαχείριση του χώρου των αναγνωριστικών από τους κόμβους. Τέλος, οι στρατηγικές δρομολόγησης και διατήρησης πρέπει να υλοποιούνται από το υιοθετημένο πρωτόκολλο. Είναι σημαντικό όλες αυτές οι απαιτήσεις να ικανοποιούνται με τέτοιο τρόπο ώστε να υποστηρίζονται τα ακόλουθα χαρακτηριστικά της επικάλυψης [AAG+05]:

- ***Αποτελεσματικότητα (Efficiency)*** των μηχανισμών δρομολόγησης και συντήρησης.
- ***Επεκτασιμότητα (Scalability)*** όσο αυξάνεται ο αριθμός των peers και των πόρων χωρίς σημαντική υποβάθμιση της απόδοσης.

- ***Αυτό-οργάνωση (Self-organization)*** κατά την άφιξη και την αναχώρηση των κόμβων χωρίς να απαιτούνται κεντρικές δομές ελέγχου.

- ***Ανεκτικότητα σε σφάλματα (Fault-tolerance)*** κατά την αναχώρηση των κόμβων, καθώς τα περιβάλλοντα αυτά χαρακτηρίζονται από δυναμικές αλλαγές και είναι επιρρεπή σε σφάλματα.

- ***Συνεργασία (Cooperation)*** μεταξύ των κόμβων ώστε να επιτευχθεί η αποτελεσματική δρομολόγηση, ανταλλαγή πληροφορίας, κλπ.

Οι ιδιότητες αυτές των DHT επικαλύψεων τα καθιστούν ιδιαίτερα δημοφιλή στα P2P δίκτυα: Η συνάθροιση τεράστιου αποθηκευτικού χώρου και επεξεργαστικής ισχύς επιτυγχάνεται, ενώ το κόστος των lookups ελαχιστοποιείται. Όπως έχει ήδη περιγραφεί, τα DHT πρωτόκολλα που έχουν προταθεί για τη δημιουργία δομημένων επικαλύψεων υλοποιούν τουλάχιστον δύο βασικούς μηχανισμούς (ή παρέχουν όλες τις απαιτούμενες λειτουργίες για την υλοποίηση τους) σε αναλογία με τη put/get διεπαφή των *Πινάκων Κατακερματισμού* (*Hash Tables*):

- *insert*(*key,value*): Ένα key αντιστοιχίζεται στην τιμή (value) προς εισαγωγή με τη χρήση μίας συνάρτησης κατακερματισμού και το ζεύγος (*key,value*) καταλήγει στον κόμβο που είναι υπεύθυνος για το κλειδί αυτό (ή κόμβους εάν είναι ενεργοποιημένη η δημιουργία πολλαπλών αντιγράφων). Επίσης, η ίδια διαδικασία ακολουθείται για την εισαγωγή ενός κόμβου στην επικάλυψη, που αντιστοιχίζεται και αυτός σε ένα identifier. Όταν ένας κόμβος εισάγεται στη επικάλυψη, τότε κάποιες διαδικασίες πρέπει να εκτελεστούν για τη διατήρηση της δομής της επικάλυψης.

- *lookup*(*key*) : Ένα ζεύγος (*key,value*) μπορεί να ανακτηθεί από την επικάλυψη, όταν ένα lookup μήνυμα δρομολογηθεί και καταλήξει στον κόμβο που είναι υπεύθυνος για το κλειδί αυτό.

Η διαδικασία που ακολουθείται κατά τη διάρκεια ενός lookup εγγυάται την αποτελεσματικότητα του μηχανισμού αναζήτησης στα DHTs αλλά ταυτόχρονα θέτει περιορισμούς στην υιοθέτηση των DHT επικαλύψεων από διάφορες εφαρμογές. Ο λόγος είναι ότι η χρήση των hash συναρτήσεων μολονότι διευκολύνει την κατασκευή της επικάλυψης και την εξισορρόπηση του φορτίου δεδομένων, ταυτόχρονα περιορίζει τη χρήση των επικαλύψεων μόνο για την απλή αναζήτηση identifirers, όπου το κλειδί ενός αποθηκευμένου data item πρέπει να είναι γνωστό εκ των προτέρων, δηλαδή πριν την αποστολή ενός lookup αιτήματος. Πιο πολύπλοκα ερωτήματα, όπως ερωτήματα συνάθροισης (aggregate), πολλαπλών ιδιοτήτων (multi-attribute), εύρους τιμών (range), ένωσης (join) και ομοιότητας (similarity) δεν μπορούν να επιλυθούν μόνο με τη χρήση μίας απλής λειτουργίας lookup και για το λόγο αυτό η πρόσβαση στα δεδομένα γίνεται

αρκετά μη αποδοτική. Η επεξεργασία τέτοιων ερωτημάτων προϋποθέτει την υλοποίηση επιπρόσθετων δομών και τεχνικών δεικτοδότησης. Για αυτό το λόγο, η επεξεργασία των δεδομένων βάσει της σημασιολογίας τους δε μπορεί να επιτευχθεί με συμβατικά DHT συστήματα.

Καθώς οι εφαρμογές απαιτούν την αποτελεσματική επίλυση όλο και πιο πολύπλοκων τύπων ερωτημάτων, τα οποία αναζητούν τα δεδομένα με βάση το περιεχόμενο τους και αξιοποιούν τις διασυνδέσεις μεταξύ διαφορετικών αντικειμένων, οι ερευνητικές προσπάθειες επικεντρώνονται στην ανάπτυξη των αντίστοιχων αλγορίθμων και μηχανισμών. Η επίλυση τέτοιων τύπων ερωτημάτων αντιμετωπίζεται σύμφωνα με τις ακόλουθες βασικές προσεγγίσεις που συναντώνται στη βιβλιογραφία:

- Προτείνονται επικαλύψεις που στηρίζονται σε ένα υπάρχον DHT πρωτόκολλο και είτε τροποποιούν/αντικαθιστούν τη συνάρτηση κατακερματισμού είτε προσθέτουν επιπλέον δομές δεικτοδότησης πάνω από τη DHT επικάλυψη και προτείνουν νέες μεθοδολογίες για την επίλυση των ερωτημάτων. Τα προτεινόμενα συστήματα δεικτοδότησης προσαρμόζονται στη δομή των δεδομένων και συνήθως επικεντρώνονται στην αποδοτική επίλυση ενός συγκεκριμένου τύπου ερωτήματος.

- Η κατανομή του συνόλου των δεδομένων ανάμεσα στα peers δεν διενεργείται με τη χρήση μίας συνάρτησης κατακερματισμού και συνεπώς αυτές οι επικαλύψεις δεν χρησιμοποιούν άμεσα κάποιο υπάρχον DHT πρωτόκολλο.

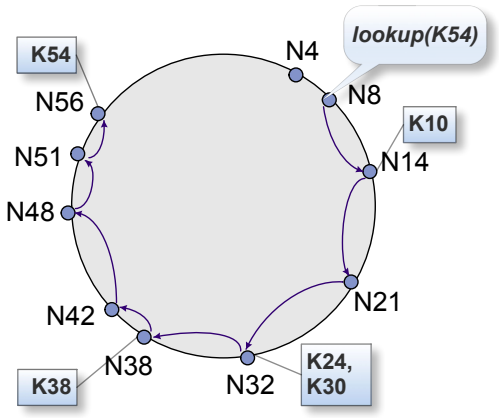
Οι μέθοδοι που περιγράφονται στη διατριβή αυτή έχουν ως αποτέλεσμα το σχεδιασμό αρχιτεκτονικών που ανήκουν στη πρώτη κατηγορία. Η επιλογή αυτής της προσέγγισης οφείλεται στο γεγονός ότι μία DHT επικάλυψη μπορεί να αντιμετωπίσει αποτελεσματικά ζητήματα που αφορούν την επεκτασιμότητα, την ανεκτικότητα σε σφάλματα, την εξισορρόπηση φόρτου και τη δημιουργία πολλαπλών αντιγράφων (replication), ενώ προτείνονται νέες μέθοδοι για την αποτελεσματική επίλυση πολύπλοκων ερωτημάτων, οι οποίες αφορούν κυρίως ερωτήματα συνάθροισης και εύρους τιμών και στοχεύουν στην αξιοποίηση των ιδιοτήτων που εμφανίζονται στη δομή των δεδομένων. Τα ερωτήματα συνάθροισης μπορούν να συσχετιστούν με τα ερωτήματα εύρους τιμών. Στο προτεινόμενο σύστημα, τα εύρη τιμών μπορούν να υποστηριχθούν εμμέσως από τις συνόψεις στα επίπεδα της ιεραρχίας. Ωστόσο, η μετατροπή ενός ερωτήματος συνάθροισης σε ερώτημα εύρους τιμών δεν είναι πάντα προφανής ή εφικτή, ειδικά όταν οι τιμές που ερωτούνται δεν είναι αριθμήσιμες.

DHT Υλοποιήσεις

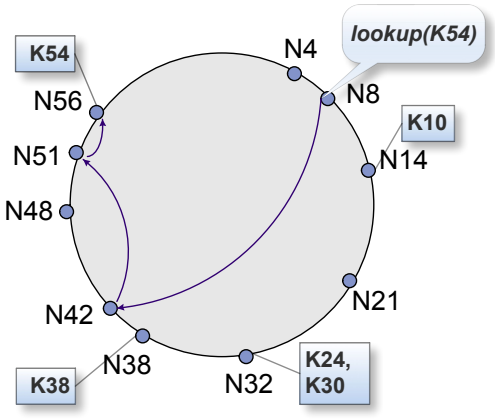
Τα DHT πρωτόκολλα που υπάρχουν είναι διάφορα και πολλά από αυτά υποστηρίζουν διαφορετικούς τρόπους οργάνωσης των δεδομένων και του key space και ακολουθούν διαφορετικές στρατηγικές δρομολόγησης. Μερικές από τις πιο δημοφιλείς υλοποιήσεις είναι τα Chord

[SMK+01], Pastry [RD01], Kademlia [MM02] και CAN [RFH+01]. Τα πρωτόκολλα αυτά διαφέρουν κυρίως ως προς το σχήμα του identifier space και συνεπώς χρησιμοποιούν διαφορετικές συναρτήσεις για τον υπολογισμό της απόσταση μεταξύ των IDs στη συγκεκριμένη εικονική δομή [LCC+02]. Σε αυτήν την ενότητα, περιγράφονται περιληπτικά κάποιοι από τους βασικούς τύπους των DHT πρωτοκόλλων. Έμφαση δίνεται στις DHT επικαλύψεις που σχετίζονται με τις επικαλύψεις που χρησιμοποιούνται στα συστήματα που προτάθηκαν στα πλαίσια της διατριβής.

Το *Chord* [SMK+01] χρησιμοποιεί consistent hashing [KLL+97] για να αντιστοιχήσει τα keys στους κόμβους που είναι υπεύθυνα για αυτά. Η συνάρτηση κατακερματισμού που χρησιμοποιείται εξισορροπεί το φόρτο ανάμεσα στους κόμβους με υψηλή πιθανότητα. Όλα τα identifiers συμπεριλαμβανομένων των identifiers για τους κόμβους και τα data items τοποθετούνται δεξιόστροφα πάνω σε έναν κύκλο. Κάθε peer διατηρεί πίνακες δρομολόγησης (που περιέχουν *finger tables*) με πληροφορία για άλλους $O(\log N)$ κόμβους. Σε ένα "Chord δακτυλίδι" ("Chord ring"), ένα κλειδί k ανατίθεται στον πρώτο κόμβο που το identifier του είναι το ίδιο με το k ή το ακολουθεί στο identifier space και ο κόμβος αυτός ονομάζεται *successor node* του key k. Κάθε κόμβος πρέπει να γνωρίζει με βεβαιότητα τον επόμενο του κόμβο στο δακτυλίδι των identifiers, έτσι ώστε να είναι δυνατό ένα lookup για ένα συγκεκριμένο key να κυκλοφορήσει στο δακτυλίδι μέσω των "succesor nodes" και να επιλυθεί με την απλή αυτή προσέγγιση. Το Σχήμα 2.1 απεικονίζει ένα δακτυλίδι που αποτελείται από δέκα κόμβους που αποθηκεύουν πέντε κλειδιά συνολικά και το μονοπάτι που ακολουθείται κατά την αναζήτηση ενός key. Ένα lookup αίτημα κυκλοφορεί γύρω γύρω στο Chord δακτυλίδι μέσω των successor δεικτών, μέχρι να συναντήσει τον κατάλληλο κόμβο. Μία πιο επεκτάσιμη λύση για τον εντοπισμό του κόμβου που είναι υπεύθυνος για ένα key που ρωτάται επιτυγχάνεται με τη διατήρηση επιπλέον πληροφορίας στα finger tables, η οποία δεν αναφέρεται μόνο στο successor node. Έτσι, σε αυτήν την προσέγγιση διατηρούνται μέχρι m εγγραφές στο finger table ενός κόμβου, όπου το m είναι ο αριθμός των bits στο ID space. Η i-τη εγγραφή στον πίνακα του κόμβου n περιέχει την ταυτότητα του πρώτου κόμβου s που διαδέχεται τον n κατά τουλάχιστον $2^{(i-1)}$ στο identifier space. Η ιδέα της προσέγγισης αυτής είναι ότι ένας κόμβος πρέπει να γνωρίζει περισσότερα για τους κοντινούς του κόμβους σε σύγκριση με αυτά που γνωρίζει για τους μακρινούς του κόμβους. Κατά τη διάρκεια μίας λειτουργίας lookup, το εμπλεκόμενο peer προωθεί το lookup μήνυμα στον κόμβο που περιλαμβάνεται στο finger table και έχει identifier που βρίσκεται σε υψηλότερη θέση αλλά δεν είναι μεγαλύτερο από το key που αναζητείται. Εφόσον κάθε κόμβος γνωρίζει τους κόμβους σε διαστήματα που είναι δυνάμεις-του-δύο, κάθε κόμβος μπορεί να προωθήσει το ερώτημα τουλάχιστον κατά το μισό της απόστασης που απομένει. Ο αριθμός των κόμβων που λαμβάνουν το lookup μέχρι να βρεθεί ο successor κόμβος σε ένα δίκτυο N κόμβων είναι $O(\log N)$. Το Σχήμα 2.2 παρουσιάζει την επίλυση ενός lookup για το ίδιο key όπως και στο Σχήμα 2.1. Όμως στο νέο Σχήμα επιδεικνύεται η πιο επεκτάσιμη λύση για την προώθηση του lookup που χρησιμοποιεί την επαυξημένη πληροφορία



Σχήμα 2.1: *Το μονοπάτι που ακολουθείται κατά τη διάρκεια ενός lookup για το key K54 ξεκινώντας από τον κόμβο N8 στο Chord δακτυλίδι. Το ερώτημα προωθείται απλά μέσω των successor δεικτών*



Σχήμα 2.2: *Το μονοπάτι που ακολουθείται κατά τη διάρκεια ενός lookup για το key K54 ξεκινώντας από τον κόμβο N8. Η πληροφορία που αποθηκεύεται στα finger tables χρησιμοποιείται για την προώθηση του ερωτήματος*

που είναι αποθηκευμένη στα finger tables. Οι αφίξεις και οι αναχωρήσεις των κόμβων απαιτούν την ενημέρωση των finger tables.

Το *Pastry* [RD01] είναι μία άλλη επικάλυψη που μπορεί να αυτό-οργανώνεται και χρησιμοποιεί ένα κυκλικό namespace, στο οποίο η θέση ενός κόμβου προσδιορίζεται από το identifier του (στη συνέχεια αναφέρεται ως *NodeId*). Το NodeId ανατίθεται τυχαία σε έναν κόμβο και λαμβάνει τιμές από το 0 έως το $2^{128} - 1$. Το Pastry είναι παρεμφερές με το Chord αλλά η δρομολόγηση σε αυτό βασίζεται στα προθέματα (prefixes) των διευθύνσεων, δηλαδή ένα μήνυμα προωθείται προς τους κόμβους που έχουν διαδοχικά μεγαλύτερα ίδια προθέματα με το identifier του προορισμού. Αυτό επιτυγχάνεται θεωρώντας ότι κάθε NodeId και key είναι μία ακολουθία ψηφίων με βάση 2^b, όπου το b είναι μία αριθμητική παράμετρος (η συνήθης τιμή του είναι 4). Όπως περιγράφεται στη εργασία [RD01], σε κάθε βήμα της δρομολόγησης ένας κόμβος προωθεί το μήνυμα σε έναν κόμβο του οποίου το NodeId έχει κοινό πρόθεμα με το key που είναι μακρύτερο κατά τουλάχιστον ένα ψηφίο σε σχέση με το κοινό πρόθεμα του NodeId του παρόντος κόμβου και του key. Υπό κανονικές συνθήκες, το Pastry μπορεί να δρομολογήσει ένα μήνυμα στον κοντινότερο κόμβο για ένα δεδομένο NodeId ή key σε λιγότερο από $\lceil \log_{2^b} N \rceil$. Σε περίπτωση που συμβαίνουν ταυτόχρονες αποτυχίες κόμβων, το πρωτόκολλο εγγυάται την παράδοση ενός μηνύματος εκτός και αν $\lfloor |L|/2 \rfloor$ κόμβοι με γειτονικά NodeIds αποτύχουν ταυτόχρονα, όπου $|L|$ είναι μία παραμετροποιημένη τιμή.

Η διαδικασία της δρομολόγησης εκτελείται με τη χρήση πληροφορίας που διατηρείται στον πίνακα δρομολόγησης (*routing table*), στο *leaf set* και στο *neighborhood set* ενός κόμβου. Ο πίνακας δρομολόγησης του κόμβου περιέχει $\lceil \log_{2^b} N \rceil$ γραμμές με $2^b - 1$ καταχωρήσεις για κάθε

γραμμή. Η n-τη γραμμή του πίνακα δρομολόγησης περιέχει μία αντίστοιχη καταχώρηση για ένα κόμβο που το NodeId έχει το ίδιο πρόθεμα για τα πρώτα n ψηφία, αλλά του οποίου το $n+1$-το ψηφία έχει μία από τις $2^b - 1$ πιθανές τιμές διαφορετική από το $n+1$-το ψηφίο του παρόντος NodeId. Εάν ένας κόμβος με ένα τέτοιο NodeId δεν υπάρχει, τότε η αντίστοιχη καταχώρηση στον πίνακα δρομολόγησης παραμένει κενή. Το leaf set είναι το σύνολο των κόμβων με τα $l/2$ αριθμητικώς κοντινότερα μεγαλύτερα NodeIds και $l/2$ αριθμητικώς κοντινότερα μικρότερα NodeIds, που σχετίζονται με το NodeId του παρόντος κόμβου. Το neighborhood set ενός κόμβου είναι ένα σύνολο από l κόμβους που είναι κοντά στον κόμβο αυτό σύμφωνα με τη μετρική της απόστασης και χρησιμοποιείται κατά την προσθήκη ενός νέου κόμβου ή κατά τη διαδικασία ανάκτησης.

Το *Kademlia* [MM02] είναι μία συμμετρική DHT επικάλυψη που χρησιμοποιεί την XOR μετρική για να μετρήσει την απόσταση μεταξύ δύο φύλλων ενός δυαδικού δέντρου. Τόσα τα data items όσο και οι κόμβοι λαμβάνουν μοναδικά identifiers από έναν ενοποιημένο χώρο διευθύνσεων. Τα αντικείμενα που εισάγονται στο DHT είναι ζεύγη της μορφής *(key, value)*, όπου το value είναι το δεδομένο που θα αποθηκευτεί στον κόμβο που έχει το "κοντινότερο" ID στο key. Η έννοια της απόστασης μεταξύ σημείων στο identifier space ορίζεται από την XOR μετρική. Το Kademlia πρωτόκολλο εντοπίζει τα data items βάσει των κλειδιών τους, μόνο εάν το ακριβές key είναι γνωστό εκ των προτέρων. Το μηνύματα των ερωτημάτων δρομολογούνται στην επικάλυψη σύμφωνα με πληροφορία που διατηρεί ο κάθε κόμβος για τους υπόλοιπους κόμβους. Η πληροφορία αυτή αποκτάται από τα μηνύματα που λαμβάνει ένα peer. Η συμμετρική ιδιότητα την XOR μετρικής επιτρέπει στους συμμετέχοντες κόμβους σε ένα Kademlia DHT να λαμβάνουν lookup ερωτήματα από την ίδια κατανομή κόμβων που βρίσκονται στους πίνακες δρομολόγησης τους. Το Kademlia ενδείκνυται ως μία κατάλληλη επιλογή λόγω της απλής δομής των πινάκων δρομολόγησης και του συνεπούς αλγόριθμου που ακολουθείται κατά τη διάρκεια της επίλυσης ενός lookup.

Οι βασικές λειτουργίες που υποστηρίζονται από το Kademlia πρωτόκολλο [MM02] είναι:

PING: Αυτό το RPC ανιχνεύει ένα κόμβο για να διαπιστώσει εάν είναι online.

STORE: Αρχικά, ένας κόμβος ξεκινάει τη STORE λειτουργία υπολογίζοντας το key για το συγκεκριμένο data item, που συνήθως είναι το αποτέλεσμα μίας hash συνάρτησης που εφαρμόζεται στο περιεχόμενο του. Στη συνέχεια, εκτελεί αναζητήσεις στο δίκτυο σε πολλαπλά βήματα, μέχρι να ανακαλυφθούν οι κοντινότεροι κόμβοι στο key. Στη συνέχεια, το data item αποθηκεύεται σε αυτούς τους κόμβους.

FIND_NODE: Η λειτουργία αυτή επιστρέφει τους *kappa* κοντινότερους κόμβους στο node ID που ζητήθηκε, όπου το *kappa* είναι μία παράμετρος του συστήματος.

FIND_VALUE: Όταν δίνεται εντολή σε ένα κόμβο ενός Kademlia δικτύου να αναζητήσει μία τιμή, ο κόμβος αυτός θέτει α παράλληλα ερωτήματα (FIND_NODE) στους *kappa* κοντινότερους κόμβους που γνωρίζει. Η διαδικασία αυτή συνεχίζεται μέχρι να ανακαλυφθούν οι *kappa* κοντινότεροι κόμβοι στο key που αποτελεί στόχο. Μόλις βρεθούν οι κόμβοι αυτοί, ο κόμβος αποστέλλει FIND_VALUE RPCs για την ανάκτηση της τιμής που αναζητείται. Το *kappa* είναι μία συστημική παράμετρος και προσδιορίζει επίσης τον αριθμό των αντιγράφων που διατηρούνται για κάθε data item, ενώ ελέγχει και το μέγεθος των πινάκων δρομολόγησης των peers. Οι τιμές των μεταβλητών α και *kappa* ορίζονται σε κάθε συμμετέχοντα κόμβο και επηρεάζουν μόνο την τοπική απόδοση της υπηρεσίας.

Όμως υπάρχει και μία πληθώρα από άλλες δομημένες επικαλύψεις, οι οποίες υλοποιούν διαφορετικές δομές, στρατηγικές δρομολόγησης, κλπ. Για παράδειγμα το CAN [RFH+01] δημιουργεί ένα Καρτεσιανό χώρο συντεταγμένων d-διαστάσεων, ο οποίος κατανέμεται δυναμικά σε όλα τα peers της επικάλυψης. Κάθε κόμβος είναι υπεύθυνος για μία διακριτή και ανεξάρτητη ζώνη του χώρου των d-διαστάσεων και διατηρεί ένα πίνακα δρομολόγησης με $O(d)$ καταχωρήσεις, ενώ οποιοσδήποτε κόμβος μπορεί να βρεθεί σε $O(dN^{1/d})$ άλματα δρομολόγησης. Δύο κόμβοι είναι "γείτονες" εάν οι ζώνες τους μοιράζονται ένα υπερ-επίπεδο (hyper-plane) των d-1 διαστάσεων. Κάθε data item ανατίθεται στον κόμβο που είναι υπεύθυνος για τη ζώνη που ανήκει. Το Tapestry [ZHS+04] ακολουθεί και αυτό μια διαδικασία δρομολόγησης που στηρίζεται στα προθέματα παρόμοια με αυτή του περιγράφηκε για το Pastry, αλλά διαφοροποιείται ως προς την προσέγγιση που ακολουθεί για να διατηρήσει την τοπικότητα και για τη δημιουργία των αντιγράφων των αντικειμένων. Το P-Grid [Abe01] είναι μία δομημένη επικάλυψη που υλοποιεί ένα κατανεμημένο δέντρο αναζήτησης. Κάθε κόμβος αναλαμβάνει ένα μέρος του συνολικού δέντρου και η θέση του στην επικάλυψη ορίζεται από το μονοπάτι του στο δέντρο, ενώ είναι υπεύθυνος για τα data items με κλειδιά που αρχίζουν με το αποτέλεσμα της αναπαράστασης σε bit του μονοπατιού του. Στο P-Grid, τα ερωτήματα επιλύονται με βάση την αντιστοίχηση του προθέματος, ενώ ο χώρος των κλειδιών κατανέμεται δυναμικά στους κόμβους ανάλογα με το φόρτο και αυτό έχει ως αποτέλεσμα την ομοιόμορφη κατανομή των keys μεταξύ των κόμβων ακόμα και όταν πρόκειται για μη ομοιόμορφες κατανομές φορτίων.

2.3 Τεχνικές Δεικτοδότησης σε P2P δίκτυα

Διάφορες ερευνητικές εργασίες διερευνούν πως μπορούν τα P2P συστήματα να επωφεληθούν από τη διαχείριση δεδομένων λαμβάνοντας υπόψη τη σημασιολογία, τους μετασχηματισμούς και τις σχέσεις των δεδομένων, έτσι ώστε να μπορούν να επεξεργάζονται πολύπλοκα ερωτήματα. Η εργασία των Risson και Moors [RM06] περιέχει επισκόπηση και σύγκριση των μεθόδων αναζήτησης σε P2P συστήματα. Επίσης, ένας μεγάλος αριθμός εργασιών δίνει έμφαση

στη δημιουργία ισχυρών δομών δεικτοδότησης και στον τρόπο που αυτές μπορούν να προσαρμοστούν για την επίλυση συγκεκριμένων τύπων ερωτημάτων, όπως ερωτήματα συνάθροισης (aggregate), δομής (structure), ομοιότητας (similarity), συνδυαστικά, κλπ. Έτσι έχουν προταθεί τεχνικές δεικτοδότησης που ακολουθούνται σε "παραδοσιακά" συστήματα βάσεων δεδομένων ή τεχνικές που συνδυάζουν τις ιδιότητες της υποκείμενης επικάλυψης με τον τύπο ερωτήματος που μελετάται. Αρχικά, παρουσιάζεται μία επισκόπηση των εργασιών που έχουν ως στόχο την επέκταση των λειτουργιών που συναντώνται σε βάσεις δεδομένων και την ενσωμάτωση τους σε κατανεμημένα περιβάλλοντα.

Το PIER [HHB+03], [HCH+05] προτείνει μία κατανεμημένη αρχιτεκτονική για μία κατανεμημένη βάση δεδομένων διαδικτυακής κλίμακας στοχεύοντας στην υποστήριξη πολλαπλών τελεστών, όπως selection, projection, union, join, group-by, κλπ. Μία DHT επικάλυψη χρησιμοποιείται για τη δρομολόγηση των ερωτημάτων, ώστε να δημιουργηθεί ένα επεκτάσιμο σύστημα που μπορεί να φιλοξενήσει ακόμα και χιλιάδες κόμβους που συνδέονται μέσω του Διαδικτύου. Κάθε πλειάδα που εισάγεται στην PIER επικάλυψη περιγράφεται με αυτόνομο τρόπο και αυτό επιτυγχάνεται γιατί περιέχει το όνομα του πίνακα (table name), τα ονόματα των στηλών (column names) και τους τύπους των στηλών (column types). Μία δευτερεύουσα δομή δεικτοδότησης υλοποιείται πάνω από τη DHT επικάλυψη, η οποία δημιουργεί ένα δέντρο που κατανέμεται στους κόμβους. Ο σκοπός αυτού του δείκτη είναι η μείωση του κόστους επικοινωνίας με τη συγκέντρωση των πλειάδων σε έναν κόμβο κατά την επίλυση των συναθροίσεων και των ενώσεων. Συνεπώς, οι κόμβοι οργανώνονται σε δέντρα και ένας κόμβος που συμμετέχει στην επίλυση τέτοιου είδους ερωτημάτων υπολογίζει το τοπικό αποτέλεσμα και το προωθεί σε κόμβο που είναι κατά ένα hop κοντινότερος προς το επίπεδο της ρίζας (root level). Κάθε κόμβος υποστηρίζει μία μέθοδο local Scan που του επιτρέπει την επισκόπηση όλων των δεδομένων του κατά την επεξεργασία ενός ερωτήματος.

Το Piazza [TH04] είναι ένα άλλο Peer Database Management System (PDMS), όπου κάθε κόμβος διατηρεί τα δεδομένα του μοντελοποιημένα σε XML. Η εργασία αυτή επικεντρώνεται κυρίως στην επίτευξη καλύτερης απόδοσης στρατηγικών αναζήτησης που βασίζονται σε αναδιατύπωση (reformulation) των ερωτημάτων, "κλάδεμα" (pruning) των περιττών βελτιστοποιήσεων και προ-υπολογισμό των σημασιολογικών μονοπατιών. Οι προτεινόμενες προσεγγίσεις καθιστούν δυνατή τη λειτουργία των κόμβων με ένα πλήρως αποκεντρικοποιημένο τρόπο χρησιμοποιώντας λειτουργίες lookup για να εμπλουτίσουν την τοπική τους γνώση.

Το PeerDB [OTZ+03] είναι ένα πλήρες σύστημα διαχείρισης δεδομένων που επιτρέπει το διαμοιρασμό σχεσιακών δεδομένων χωρίς να απαιτείται η ύπαρξη προκαθορισμένων σχημάτων, τα οποία πρέπει να είναι γνωστά σε όλους τους κόμβους. Σε κάθε κόμβο του PeerDB υπάρχει μία τοπική βάση δεδομένων, ενώ ένα "Τοπικό Λεξικό" ("Local Dictionary") αποθηκεύει τα μεταδεδομένα που σχετίζονται με κάθε σχέση και μία άλλη δομή λεξικού αποθηκεύει τα μεταδεδομένα που διαμοιράζεται με άλλους κόμβους. Στο PeerDB προτείνεται επίσης η χρήση πρακτόρων

(agents) που είναι υπεύθυνοι για την επεξεργασία των ερωτημάτων. Σε μία πρώτη φάση, οι κόμβοι ερωτούνται και επιστρέφουν υποψήφια μεταδεδομένα ως απάντηση. Σε μία δεύτερη φάση, ο χρήστης επιλέγει τα σχετικά μεταδεδομένα και οι πηγές που επιλέχθηκαν ερωτούνται απευθείας, ώστε μία αναδιατυπωμένη ερώτηση που ταιριάζει με το όνομα της σχέσης και τις ιδιότητες του κόμβου τίθεται στην τοπική βάση δεδομένων.

Το σύστημα pSearch [TXD03] υλοποιείται πάνω από μία CAN επικάλυψη που αποθηκεύει έγγραφα (documents) που δεικτοδοτούνται σύμφωνα με τα σημασιολογικά τους διανύσματα (semantic vectors) που παράγονται από το Latent Semantic Indexing (LSI). Το συγκεκριμένο σύστημα δε μπορεί να χειριστεί δυναμικές συλλογές εγγράφων, επειδή τα semantic vectors πρέπει να οριστούν εκ των προτέρων και επομένως τα έγγραφα νέων κόμβων με όρους που δεν εμπεριέχονται στο υπάρχον vector δε είναι δυνατόν να δεικτοδοτηθούν από αυτούς.

Το Mercury [BAS04] είναι ένα δίκτυο που αποτελείται από πολλαπλές κυκλικές επικαλύψεις, ενώ κάθε επικάλυψη αντιστοιχεί σε μία ξεχωριστή ιδιότητα. Οι κόμβοι χωρίζονται σε ομάδες που ονομάζονται "attribute hubs" και κάθε hub είναι υπεύθυνο για μία ξεχωριστή ιδιότητα. Οι κόμβοι ενός hub οργανώνονται με τέτοιο τρόπο ώστε να "διατηρείται η σειρά" (in an order preserving manner) και κάθε κόμβος είναι υπεύθυνος για ένα συγκεκριμένο εύρος τιμών μίας ιδιότητας. Το σύστημα αυτό επιτρέπει την επίλυση ερωτημάτων εύρους τιμών που απαιτούν την ταξινόμηση των αποθηκευμένων δεδομένων. Ένα ερώτημα για περισσότερες από μία ιδιότητες θεωρείται ως συνδυασμός κατηγορημάτων και δρομολογείται σε όλα τα σχετικά hubs. Η εξάλειψη της hash συνάρτησης μπορεί να οδηγήσει σε μη ομοιόμορφη κατανομή των δεδομένων στους κόμβους και για αυτό το λόγο δίνεται έμφαση στην εξισορρόπηση του φόρτου με τη χρήση τυχαίας δειγματοληψίας για την αποφυγή προβλημάτων σε πολωμένες κατανομές δεδομένων.

Όλες αυτές οι προσεγγίσεις προσφέρουν λύσεις για τη δημιουργία κατανεμημένων συστημάτων βάσεων δεδομένων που στοχεύουν στην αποθήκευση και στο διαμοιρασμό δομημένων και ετερογενών δεδομένων. Τα συστήματα που στηρίζονται σε δομημένες επικαλύψεις χαρακτηρίζονται από καλύτερη απόδοση κατά τη διάρκεια των αναζητήσεων αλλά και από αυξημένο κόστος συντήρησης. Και στις δύο περιπτώσεις, τα συστήματα που περιγράφηκαν υιοθετούν μία καθολική στρατηγική δεικτοδότησης για όλες τις πλειάδες του δικτύου και αυτό έχει ως αποτέλεσμα μία "δαπανηρή" διαδικασία όσο αυξάνει ο όγκος των δεδομένων και ο αριθμός των σχημάτων και των ιδιοτήτων. Η διαδικασία δεικτοδότησης και συντήρησης των δεικτών σε δεδομένα ευρείας κλίμακας καταλήγει να είναι "δαπανηρή" όσον αφορά την κατανάλωση του εύρους ζώνης και το χρόνο απόκρισης, ειδικά όταν παρατηρούνται υψηλοί ρυθμοί ενημερώσεων και αφίξεων / αναχωρήσεων κόμβων.

Μία πιο δυναμική λύση για τη δημιουργία ενός συστήματος διαχείρισης δεδομένων που στηρίζεται σε ομότιμους κόμβους είναι το PISCES [WLOT08], το οποίο δεικτοδοτεί μόνο ένα μέρος των αποθηκευμένων πλειάδων. Το συγκεκριμένο σύστημα υποστηρίζει μία μερική στρατηγική δεικτοδότησης (partial indexing strategy) για ένα υποσύνολο των αποθηκευμένων πλειάδων

σύμφωνα με κάποια κριτήρια, όπως η συχνότητα των ερωτημάτων και η συχνότητα των ενημερώσεων. Κάθε κόμβος διατηρεί τη δική του βάση δεδομένων, αλλά συμμετέχει και σε ένα δομημένο δίκτυο όπως το BATON [JOV05] και το CAN [RFH+01]. Ο partial δείκτης δημιουργείται με τη χρήση προσεγγιστικής πληροφορίας σχετικά με το συνολικό αριθμό των κόμβων, το συνολικό αριθμό των ερωτημάτων, την κατανομή των ερωτημάτων και τις αφίξεις και αναχωρήσεις των κόμβων, η οποία συλλέγεται από μία διαδικασία που στηρίζεται σε ιστογράμματα. Ωστόσο, η προσέγγιση αυτή στοχεύει κυρίως στην υποστήριξη ερωτημάτων που αφορούν σχεσιακά δεδομένα και δεν αντιμετωπίζει την ειδική περίπτωση των ιεραρχιών σε πολυδιάστατα δεδομένα, ειδικά όταν δίνεται έμφαση στις μεταξύ τους διασυνδέσεις.

Εκτός από τα γενικότερα συστήματα που στοχεύουν στο διαμοιρασμό δομημένων δεδομένων, υπάρχουν και διάφορες άλλες ερευνητικές προσπάθειες που επικεντρώνονται στο σχεδιασμό P2P επικαλύψεων και προσανατολίζονται προς της αποτελεσματική επίλυση μίας ειδικής κατηγορίας ερωτημάτων, δηλαδή οι προτεινόμενες τεχνικές δεικτοδότησης τους προορίζονται για αυτό το συγκεκριμένο σκοπό. Οι προσεγγίσεις που ακολουθούνται στοχεύουν είτε στην αντικατάσταση της hash συνάρτησης με κάποια που διατηρεί τη σειρά των δεδομένων στους κόμβους είτε στη δημιουργία επιπρόσθετων δομών, που μπορούν να εφαρμοστούν πάνω από υπάρχοντες δομημένες επικαλύψεις.

Μία προσέγγιση που στοχεύει να επηρεάσει την τοποθέτηση των σχεσιακών αντικειμένων σε μία DHT επικάλυψη με τη χρήση μιας νέας hash συνάρτησης παρουσιάζεται από τους [GAA03]. Οι συγγραφείς αναπτύσσουν μια P2P αρχιτεκτονική για τον υπολογισμό προσεγγιστικών απαντήσεων σε πολύπλοκα ερωτήματα που εκφράζονται με τη χρήση SQL δηλώσεων. Οι πλειάδες μίας σχεσιακής βάσης δεδομένων χωρίζονται οριζόντια και ανατίθενται σε κόμβους με τρόπο τέτοιο ώστε να διατηρείται η τοπικότητα με υψηλή πιθανότητα χρησιμοποιώντας *Locality Sensitive Hashing*. Η επεξεργασία ενός ερωτήματος περιλαμβάνει τον εντοπισμό των τμημάτων (partitions) που σχετίζονται με το ερώτημα και επιστρέφονται προσεγγιστικά αποτελέσματα. Η ποιότητα των αποτελεσμάτων εξαρτάται από την πολυπλοκότητα της hash συνάρτησης, η οποία όμως μπορεί να δημιουργεί προβλήματα που σχετίζονται με την εξισορρόπηση του φόρτου.

Στην εργασία [AX02] παρουσιάζεται μια προσπάθεια υποστήριξης ενός κατανεμημένου πληροφοριακού συστήματος για υπολογιστικά συστήματα Πλέγματος πάνω από μία CAN επικάλυψη. Κάθε ιδιότητα που περιγράφει μία πηγή δεικτοδοτείται σε μία διαφορετική επικάλυψη. Η υιοθέτηση μίας hash συνάρτησης που βασίζεται στη Hilbert Καμπύλη Πλήρωσης του Χώρου (Hilbert Space Filling Curve) στοχεύει να επιτύχει την αντιστοίχηση διαστημάτων τιμών σε κοντινές CAN ζώνες. Συνεπώς, η συγκεκριμένη προσέγγιση επικεντρώνεται στην επίλυση ερωτημάτων εύρους τιμών, τα οποία αρχικά δρομολογούνται στον κόμβο που είναι υπεύθυνος για το μεσαίο σημείο του εύρους τιμών που αναζητείται και στη συνέχεια προωθούνται αναδρομικά σύμφωνα με τρεις στρατηγικές πλημμύρας που έχουν προταθεί και αξιολογηθεί σε αυτήν την

εργασία. Εάν ένα ερώτημα περιλαμβάνει περισσότερες ιδιότητες και όχι μόνο μία, τότε το ερώτημα επιλύεται για κάθε ιδιότητα ξεχωριστά και τα αποτελέσματα συνενώνονται με ένα "τελεστή ένωσης" σαν αυτόν που συναντάται στις βάσεις δεδομένων, αυξάνοντας το κόστος επίλυσης των πολυπλοκότερων ερωτημάτων.

Για την αποδοτικότερη επεξεργασία ερωτημάτων που δε μπορούν να επιλυθούν με απλά DHT lookups, μία συνηθισμένη στρατηγική είναι η υιοθέτηση πρόσθετων μηχανισμών δεικτοδότησης που εφαρμόζονται πάνω από DHT επικαλύψεις. Μία δομή δεικτοδότησης που έχει χρησιμοποιηθεί σε διάφορες εκδοχές για αναζητήσεις βάσει των προθεμάτων είναι οι δομές tries. Οι δομές αυτές είναι μία γενίκευση των δέντρων που αποθηκεύουν και επεξεργάζονται συμβολοσειρές (strings), όπου κάθε κόμβος αντιστοιχεί σε ένα κοινό prefix. Τα πραγματικά δεδομένα αποθηκεύονται στους κόμβους-φύλλα (leaf nodes) των tries και επομένως τα lookups επιλύονται μόλις βρεθεί το leaf με ετικέτα που είναι ένα prefix της τιμής που ρωτάται. Σε διάφορες εργασίες, οι κόμβοι των tries κατανέμονται στους κόμβους της DHT επικάλυψης. Το Prefix Hash Tree (PHT) [RRHS04] είναι μία δομή που βασίζεται σε trie για τη δεικτοδότηση δυαδικών συμβολοσειρών σύμφωνα με το κοινό τους prefix. Ο βασικός τύπος ερωτημάτων που υποστηρίζονται σε αυτό το σύστημα είναι ερωτήματα εύρους τιμών σε μία διάσταση. Κάθε κόμβος διατηρεί ένα δείκτη στον αμέσως επόμενο αριστερό και δεξί κόμβο και ένα ερώτημα εύρους τιμών μπορεί να επιλυθεί είτε με γραμμικό είτε με δυαδικό τρόπο. Ένα ερώτημα εύρους τιμών εκτελείται με την αποστολή διάφορων DHT lookups και το κόστος του εξαρτάται από τα δεδομένα. Οι συγγραφείς Datta et al. [DHJ+05] ενσωματώνουν το trie στο ίδιο το δίκτυο προτείνοντας τις κατάλληλες τροποποιήσεις στους πίνακες δρομολόγησης μίας P-Grid επικάλυψης. Οι exact match αναζητήσεις επιλύονται από το μηχανισμό του P-Grid, ενώ δύο στρατηγικές προτείνονται για την επίλυση ερωτημάτων εύρους τιμών. Η min-max στρατηγική διάσχισης ξεκινάει από την επερώτηση ενός εκ των ορίων του εύρους τιμών. Ο υπεύθυνος κόμβος για αυτό επιστρέφει τα σχετικά δεδομένα και προωθεί το ερώτημα στον peer που είναι υπεύθυνος για το επόμενο partition του key space. Η διαδικασία αυτή επαναλαμβάνεται μέχρι να βρεθεί ο κόμβος που είναι υπεύθυνος για το όριο του εύρους τιμών που απέμεινε. Μία άλλη στρατηγική που υλοποιείται θεωρεί ότι το ερώτημα εύρους τιμών χωρίζεται σε ερωτήματα που αφορούν μικρότερα εύρη τιμών και τα ερωτήματα αυτά αποστέλλονται στο δίκτυο ταυτόχρονα. Η δομή *Distributed Lexical Placement Table* (DLPT) [CDT07] είναι ένα trie που κτίζεται δυναμικά πάνω από μία DHT επικάλυψη για την δεικτοδότηση και την ανακάλυψη υπηρεσιών σε περιβάλλοντα πλέγματος.

Όμως, υπάρχουν και άλλοι τύποι δέντρων που αξιοποιούνται για την δημιουργία κατανεμημένων δομών δεικτοδότησης. Τα P-Trees [CLGS04] είναι αποκεντρικοποιημένες δομές δεικτοδότησης που διατηρούν μέρη από ημιαυτόνομα B+ δέντρα πάνω από ένα Chord δακτυλίδι. Τα δέντρα αυτά χρησιμοποιούνται για τη δρομολόγηση ερωτημάτων ισότητας (equality queries) και εύρους τιμών κατά μήκος των μονοπατιών τους. Ωστόσο, ο τρόπος που εκτελείται η δρομολόγηση δεν εγγυάται τον τερματισμό της, για παράδειγμα σε περίπτωση που ένας κόμβος

υποστεί βλάβη λόγω της πολύπλοκης διαχείρισης. Το Range Search Tree (RST) είναι ένα άλλο δέντρο για τη δεικτοδότηση διαστημάτων τιμών πάνω από DHT επικαλύψεις (π.χ. Chord). Κάθε επίπεδο του δέντρου αντιστοιχεί σε διαφορετικό επίπεδο λεπτομέρειας των partitions των δεδομένων, ενώ το περιεχόμενο δηλώνεται σε όλα ή σε διαφορετικά επίπεδα του RST και στους αντίστοιχους κόμβους του DHT.

Μία διαφορετική προσέγγιση που στηρίζεται σε πολυεπίπεδα bloom φίλτρα (multi-level bloom filters) προτείνεται στο [KP04]. Στην εργασία αυτήν, οι περιγραφόμενες δομές χρησιμοποιούνται για τη δημιουργία περιλήψεων ιεραρχικών δεδομένων και υποστηρίζουν ερωτήματα με εκφράσεις κανονικών μονοπατιών (regular path expresion queries). Τα δεδομένα υπό μελέτη είναι XML δέντρα που περιέχονται σε έγγραφα και τα πολυεπίπεδα φίλτρα χρησιμοποιούνται για να συνοψίσουν τα έγγραφα που αποθηκεύονται τοπικά σε έναν κόμβο και τα έγγραφα των γειτονικών του κόμβων, ενώ οι κόμβοι με παρεμφερή φίλτρα (και συνεπώς παρεμφερές περιεχόμενο) διασυνδέονται. Οι αναφερόμενες δομές για τη σύνοψη των δεδομένων λαμβάνονται υπόψη κατά τη δρομολόγηση των ερωτημάτων που αφορούν κατανεμημένες συλλογές και όχι την αναζήτηση των ίδιων των εγγράφων.

Ένα πιο σχετικό σύστημα που στοχεύει στην online επεξεργασία ερωτημάτων συνάθροισης περιγράφεται στο Distributed Online Aggregation (DOA) Scheme [WJOT09]. Στην περίπτωση αυτήν, ο υπολογισμός των αποτελεσμάτων γίνεται κατά προσέγγιση και πολλαπλές επαναλήψεις λαμβάνουν χώρα: σε κάθε επανάληψη ένα μικρό σύνολο από τυχαία δείγματα ανακτάται από τις τοποθεσίες των δεδομένων και ανατίθεται στους διαθέσιμους κόμβους για επεξεργασία. Η τυχαία δειγματοληψία από τους διάφορους κόμβους μπορεί να πολώσει το αποτέλεσμα και προϋποθέτει ότι κάθε κόμβος πρέπει να παρέχει μία διεπαφή για την επιλογή των τυχαίων δειγμάτων από τη βάση δεδομένων του. Εκτός από αυτό το ζήτημα, όλα τα δεδομένα πρέπει να αντιστοιχίζονται σε ένα καθολικό σχήμα, αλλά αυτό δεν είναι συνήθως εφικτό και εισάγει πολυπλοκότητα στο σύστημα.

2.4 Δεικτοδότηση Σημασιολογικών Δεδομένων σε P2P επικαλύψεις

Το Resource Description Framework (RDF) είναι ένα ευρέως αποδεκτό πρότυπο στο Σημασιολογικό Ιστό και έχει υιοθετηθεί από πολλές εφαρμογές για την αναπαράσταση ημι-δομημένων δεδομένων. Πολλές προσεγγίσεις συνδυάζουν τεχνικές από τις σχεσιακές βάσεις δεδομένων για την κατασκευή κεντρικών αποθηκών (repositories) για τη δεικτοδότηση και επερώτηση δεδομένων μεγάλου όγκου, όπως αναφέρεται και στην εργασία [HSTW11]. Οι πιο αντιπροσωπευτικές κατηγορίες στις κεντρικοποιημένες προσεγγίσεις είναι οι αποθήκες για triples (triple stores) και οι οριζοντίως διαμερισμένοι πίνακες (vertically partitioned tables). Στην πλειοψηφία των περιπτώσεων, τα triple stores όπως το Virtuoso [Vir], το 3store [HG03] και το RDF-3X [NW10] αποθηκεύουν RDF triples σε έναν απλό σχεσιακό πίνακα. Η περιγραφόμενη λύση

στην εργασία [AMMH07] προτείνει μια φυσική οργάνωση των triples που βασίζεται σε vertically partitioning tables. Σε ένα τέτοιο σχήμα, τα triples κατανέμονται σε πολλαπλούς πίνακες που περιέχουν δύο στήλες. Κάθε πλειάδα ενός συγκεκριμένου πίνακα περιέχει τις στήλες για το subject και το object για τη συγκεκριμένη ιδιότητα που τα συνδέει. Η προσέγγιση αυτή υπερτερεί σαφώς των απλών λύσεων που εκτελούν πολλαπλά joins σε έναν τεράστιο RDF πίνακα ή σε *πίνακες ιδιοτήτων* (*property tables*), οι οποίοι δε μπορούν να διαχειριστούν ιδιότητες πολλαπλών τιμών (π.χ. μία τιμή ενός subject που συνδέεται με πολλαπλές τιμές).

Ωστόσο, οι κεντρικοποιημένες λύσεις δεν είναι επεκτάσιμες [MAYU05] και για αυτό το λόγο έχουν υλοποιηθεί πιο προηγμένα συστήματα δεικτοδότησης πάνω από triple stores για την ενίσχυση της απόδοσης τους [AMMH07, WKB08]. Στο σύστημα Hexastore [WKB08], η προαναφερόμενη λύση τροποποιείται, δεδομένου ότι εξυπηρετεί αποδοτικά μόνο ερωτήματα που αφορούν τις ιδιότητες και όχι γενικής φύσεως ερωτήματα. Η προτεινόμενη δομή δεικτοδότησης αποτελείται από έξι διαφορετικούς δείκτες που αντιμετωπίζουν ισότιμα τα subject, object και property ενός triple. Καθένας από τους δείκτες επικεντρώνεται σε ένα μόνο στοιχείο των triples και ορίζει μία ιεράρχηση στα εναπομείναντα στοιχεία. Στην πράξη, οι δείκτες αυτοί υλοποιούν όλες τις δυνατές διατάξεις για τις διαφορετικές προτεραιότητες των τριών RDF στοιχείων που έχει ως αποτέλεσμα την αύξηση της αποθήκευσης κατά πέντε φορές στη χειρότερη περίπτωση. Τα δύο συστήματα [AMMH07] και [WKB08] βασίζονται σε παρόμοιες αντιλήψεις και προσπαθούν να εξαλείψουν τα προβλήματα που αφορούν την επεκτασιμότητα των κεντρικών RDF stores που υπάρχουν βελτιώνοντας την απόδοση τους σε πολύπλοκα ερωτήματα. Τα δύο αυτά συστήματα χαρακτηρίζονται από κάποιους περιορισμούς (οι δυνατότητες αναζήτησης είναι περιορισμένες σε μερικές μόνο ιδιότητες στο [AMMH07] ενώ στο [WKB08] η αποτίμηση των ερωτημάτων γίνεται μόνο στην κύρια μνήμη ενώ απαιτείται επιπλέον αποθηκευτικός χώρος) και προσφέρουν μόνο δυνατότητες διαχείρισης κεντρικών αποθηκών.

Όσο αυξάνει ο όγκος των δεδομένων, η διαχείριση τους δε μπορεί να γίνει αποτελεσματικά από τις κεντρικοποιημένες λύσεις και για αυτό το λόγο έχουν προταθεί διάφορα κατανεμημένα συστήματα στη βιβλιογραφία. Όσον αφορά τα δομημένα P2P συστήματα, η πλειοψηφία των προσπαθειών εστιάζει στον τρόπο με τον οποίο θα κατανεμηθούν τα RDF δεδομένα σε πολλαπλούς peers. Το σύστημα RDFPeers [CF04] αποτελεί μία από τις αρχικές προσπάθειες για την αποθήκευση triples σε μία MAAN επικάλυψη [CFCS04], όπου το κάθε triple εισάγεται με τρία διαφορετικά κλειδιά στην επικάλυψη που παράγονται από το hashing του subject, του object και του property. Το σύστημα ATLAS [KKK+10] χρησιμοποιεί το RDF-peers για την επερώτηση των RDF δεδομένων. Το GridVine [ACMHP04] ακολουθεί μία παρόμοια προσέγγιση με το RDFPeers για την εισαγωγή των RDF triples σε μία P-Grid επικάλυψη [Abe01] και τα αποθηκεύει σε μία τοπική βάση δεδομένων του κόμβου. Μία άλλη πλατφόρμα που υποστηρίζει ερωτήματα μονοπατιού βάσει δεικτών είναι η IMAGINE-P2P πλατφόρμα [ZSL+05] και στην οποία κατασκευάζεται μία σημασιολογική επικάλυψη πάνω από μία δομημένη P2P επικάλυψη για τον

εντοπισμό των αντικειμένων και τη διαχείριση των υπηρεσιών. Τα triples που περιέχουν τις τιμές και τη σχέση που συνδέει αυτές τις τιμές αποθηκεύονται στη σημασιολογική επικάλυψη, όπου χρησιμοποιούνται δομές trie. Τα ερωτήματα μονοπατιού χωρίζονται σε υποερωτήματα, ενώ η επεξεργασία τους γίνεται σειριακά και επιλύονται κατά μήκος των index paths που ορίζονται από τις σχέσεις μεταξύ των σημασιολογικών αντικειμένων που αποθηκεύονται στο Chord δακτυλίδι.

Όλες αυτές οι προσπάθειες που αφορούν τα P2P δίκτυα είναι πιθανό να οδηγήσουν σε υπερφορτωμένους κόμβους που δεν αποδίδουν ικανοποιητικά στην περίπτωση που αποθηκεύουν τα δημοφιλή triples (όπως για παράδειγμα ο κόμβος που είναι υπεύθυνος για το rdf:type). Επίσης δεν αξιοποιείται η σημασιολογική πληροφορία που εμπεριέχεται στα ίδια τα δεδομένα και έτσι δημιουργείται η ανάγκη της δημιουργίας επιπρόσθετων σημασιολογικών επιπέδων ή της παρέμβασης στη οργάνωση των κόμβων ανάλογα με τα δεδομένα (π.χ. [KSHS08], [ZHDR09]) ή της επέκτασης του RDF μοντέλου. Αυτές οι αλλαγές προσθέτουν επιπλέον πολυπλοκότητα στο σύστημα και αυξάνουν το κόστος διατήρησης λόγω των επιπρόσθετων δομών, ειδικά κατά τη διάρκεια των ενημερώσεων. Επίσης, όλες οι προσεγγίσεις αυτές αντιμετωπίζουν όλα τα προβλήματα της κατανεμημένης επεξεργασίας ερωτημάτων, όπου μεγάλοι όγκοι δεδομένων πρέπει να ανακτηθούν στον κόμβο που έθεσε το ερώτημα και να γίνουν joined πριν συνεχίσει η αποτίμηση των υπόλοιπων τμημάτων του ερωτήματος.

2.5 Διαχείριση Πολυδιάστατων Δεδομένων σε P2P επικαλύψεις

2.5.1 Το Πολυδιάστατο Μοντέλο Δεδομένων

Το πολυδιάστατο μοντέλο δεδομένων είναι ένα μοντέλο που χρησιμοποιείται σε αποθήκες δεδομένων (data warehouses) και σε εφαρμογές για *On-Line Αναλυτική Επεξεργασία* (*On-Line Analytical Processing* - OLAP). Ο σκοπός του μοντέλου αυτού είναι να εξυπηρετήσει τη γρήγορη επίλυση πολύπλοκων ερωτημάτων, αφού όλη η διαδικασία της αναλυτικής επεξεργασίας διαδραματίζεται online. Το μοντέλο αυτό θεωρεί τα δεδομένα σε μορφή *κύβων δεδομένων* (*data cube*) ή *κύβων* που ορίζονται από διαστάσεις και facts, όπου τα facts είναι αριθμητικές μετρήσεις (numerical measures) [HK00]. Οι διαστάσεις είναι προοπτικές (perspectives) ή οντότητες που χαρακτηρίζουν ή κατηγοριοποιούν τα δεδομένα και σχηματίζουν τις άκρες του κύβου. Η οργάνωση των measures σε ένα data cube απαιτεί να έχουν όλα τα measures το ίδιο σχήμα, δηλαδή να έχουν το ίδιο αριθμό διαστάσεων. Ένα τυπικό παράδειγμα είναι το data cube για τις πωλήσεις μίας εταιρείας. Σε αυτήν την περίπτωση, οι διαστάσεις μπορεί να είναι `Location`, `Time`, `Customer` και `Product` και το measure που περιγράφουν να είναι ο συνολικός όγκος των πωλήσεων.

Στις σχεσιακές βάσεις δεδομένα, τα δεδομένα N διαστάσεων μοντελοποιούνται ως τομείς με N ιδιότητες. Ο τελεστής του κύβου είναι η γενίκευση σε N διαστάσεις των απλών συναρτήσεων

συνάθροισης [GCB+97] και εκτελεί τον υπολογισμό της συνάρτησης συνάθροισης για όλους τους πιθανούς συνδυασμούς ομαδοποίησης των ιδιοτήτων. Η σχεσιακή υλοποίηση ενός κύβου είναι συνήθως ένα star σχήμα ή ένα snowflake σχήμα. Στα star σχήματα, τα δεδομένα οργανώνονται σε *πίνακες διαστάσεων* (*dimension tables*), πίνακες των fact (*fact tables*) και υλοποιημένες όψεις (*materialized views*). Ένας fact table είναι ένας μεγάλος, κεντρικός πίνακας δεδομένων που περιέχει τη μάζα των δεδομένων χωρίς πλεονασμούς. Ο dimension table αναπαριστά μία διάσταση και περιέχει ένα σύνολο ιδιοτήτων, που μπορεί να σχηματίζουν είτε μία ιεραρχία (συνολική διάταξη) ή ένα πλέγμα (μερική διάταξη). Το snowflake σχήμα είναι μία παραλλαγή του star σχήματος, όπου μερικοί dimension tables έχουν κανονικοποιηθεί, ώστε με το διαχωρισμό των δεδομένων σε επιπλέον πίνακες να αποφεύγεται η πλεονάζουσα πληροφορία.

Η οργάνωση των δεδομένων σε διαφορετικά επίπεδα σύνοψης με τη χρήση ιεραρχιών είναι σύνηθες φαινόμενο στις αποθήκες δεδομένων. Μία εννοιολογική ιεραρχία ορίζει μία ακολουθία αντιστοιχίσεων από ένα σύνολο εννοιών χαμηλού επιπέδου σε υψηλότερου επιπέδου, γενικότερες έννοιες. Στο πολυδιάστατο μοντέλο δεδομένων, οι διαστάσεις μπορεί να περιέχουν πολλαπλά επίπεδα αφαιρετικότητας που ορίζονται από εννοιολογικές ιεραρχίες επιτρέποντας στους χρήστες την επισκόπηση των δεδομένων σε διαφορετικά επίπεδα σύνοψης. Σύμφωνα με το [HK00], οι ακόλουθες OLAP λειτουργίες για το πολυδιάστατο μοντέλο δεδομένων ορίζονται:

- Το ***Roll-up*** είναι η λειτουργία που εκτελεί τη συνάθροιση σε ένα data cube είτε "ανεβαίνοντας" μία εννοιολογική ιεραρχία μίας διάστασης είτε μειώνοντας τις διαστάσεις. Όταν μειώνονται οι διαστάσεις, μία ή περισσότερες διαστάσεις αφαιρούνται από το συγκεκριμένο κύβο.

- Το ***Drill-down*** επιτρέπει την πλοήγηση από λιγότερο λεπτομερή δεδομένα σε περισσότερο λεπτομερή, είτε κατεβαίνοντας μία εννοιολογική ιεραρχία μίας διάστασης είτε με την εισαγωγή πρόσθετων διαστάσεων.

- Η ***Slice and dice*** λειτουργία εκτελεί μία επιλογή σε μία διάσταση ενός συγκεκριμένου υποκύβου με αποτέλεσμα έναν υποκύβο.

- Το ***Pivot (or rotate)*** είναι μία λειτουργία απεικόνισης που περιστρέφει τους άξονες σε μία όψη ώστε να παρέχει μία εναλλακτική παρουσίαση των δεδομένων.

Ο υπολογισμός ενός κύβου είναι μια διαδικασία που περιλαμβάνει τη σάρωση των αρχικών δεδομένων έτσι ώστε να υπολογίσει τη συνάρτηση συνάθροισης σε όλες τις ομαδοποιήσεις και να δημιουργήσει τις σχεσιακές όψεις με το περιεχόμενο του κύβου. Η αύξηση των διαστάσεων έχει ως αποτέλεσμα την εκθετική αύξηση του μεγέθους του data cube και για αυτόν το λόγο έχουν αναπτυχθεί διάφορες μέθοδοι για τον υπολογισμό των data cubes σε κεντρικοποιημένα

συστήματα. Στις MOLAP προσεγγίσεις (π.χ. [ZDN97]), το data cube αποθηκεύεται σε ένα πολυδιάστατο πίνακα. Το κύριο μειονέκτημα αυτής της μεθόδου είναι ότι τα cubes είναι αραιά στις περισσότερες πρακτικές εφαρμογές με ένα μεγάλο αριθμό άδειων κελιών; το γεγονός αυτό έχει ως αποτέλεσμα τη μη αποτελεσματική διαχείριση του χώρου αποθήκευσης. Μία άλλη βασική κατηγορία είναι οι ROLAP μέθοδοι. Η μέθοδος Bottom-Up Cube (BUC) [BR99] περιγράφει τον υπολογισμό του cube με ένα bottom-up τρόπο, ξεκινώντας από τον υπολογισμό της τιμής *ALL*. Επίσης υπολογίζει μόνο τις group-by πλειάδες για τις οποίες η συνάρτηση συνάθροισης λαμβάνει τιμή μεγαλύτερη από κάποιο προκαθορισμένο ελάχιστο κατώφλι. Διάφορες άλλες μέθοδοι έχουν στηριχθεί στο BUC, π.χ. το CURE [MI06] που υποστηρίζει ένα παρόμοιο σχέδιο εκτέλεσης, ενώ ταυτόχρονα υποστηρίζει ιεραρχικά δεδομένα και αφαιρεί όλες τις μορφές πλεονάζουσας πληροφορίας για την αποτελεσματική χρήση του αποθηκευτικού χώρου. Μία τρίτη κατηγορία είναι οι Graph-Based μέθοδοι, οι οποίες χρησιμοποιούν δομές δενδρικής μορφής για την κατασκευή του cube και την αποτελεσματική αποθήκευση. Το Dwarf [SDRK02] είναι μια δομή που συμπιέζει σε μεγάλο βαθμό τα δεδομένα και αυτό επιτυγχάνεται με την εξάλειψη των κοινών προθεμάτων και καταλήξεων. Στην εργασία [SDKR03], η αρχική δομή Dwarf έχει επεκταθεί έτσι ώστε να υποστηριχθούν ερωτήματα συνάθροισης σε κάθε επίπεδο της ιεραρχίας μίας διάστασης. Τα QC-trees [LPZ03] είναι μία άλλη δομή που επιτυγχάνει συμπίεση των δεδομένων χρησιμοποιώντας παρεμφερείς τεχνικές και υποστηρίζει roll-up και drill-down λειτουργίες. Ωστόσο, οι προσεγγίσεις αυτές παρουσιάζουν αυξημένη πολυπλοκότητα όσον αφορά την κατασκευή των δομών τους.

2.5.2 Τεχνικές Δεικτοδότησης σε P2P επικαλύψεις

Όσο αυξάνει η χρήση των δεδομένων που ακολουθούν το πολυδιάστατο μοντέλο δεδομένων σε διάφορες εφαρμογές, οι χρήστες τείνουν να απαιτούν την επεξεργασία πολύπλοκων ερωτημάτων (π.χ. ερωτημάτων συνάθροισης, εύρους τιμών, κλπ.). Ο απλός μηχανισμός lookup των "παραδοσιακών" P2P πρωτοκόλλων δεν είναι επαρκής και η χρήση μόνο των απλών lookups παρουσιάζει χαμηλή απόδοση σε ερωτήματα πολλαπλών ιδιοτήτων (ή πολυδιάστατα). Για παράδειγμα, ένα ερώτημα για όλα τα προϊόντα μίας εταιρείας που δημιουργήθηκαν από ένα συγκεκριμένο κατασκευαστή και πουλήθηκαν κατά τη διάρκεια μίας συγκεκριμένης περιόδου δεν είναι δυνατόν να επιλυθεί αποδοτικά με την βασική λειτουργία lookup του DHT.

Η δεικτοδότηση πολλαπλών ιδιοτήτων για την επίλυση τέτοιου είδους ερωτημάτων απαιτεί τα keys που χρησιμοποιούνται στη DHT επικάλυψη να εμπεριέχουν τη σημασιολογική πληροφορία που χρειάζεται. Μία λύση για την επίτευξη αυτού του σκοπού είναι η χρήση των τιμών των ιδιοτήτων προς δεικτοδότησης (ή των διαστάσεων) στη δημιουργία των κλειδιών. Όσον αφορά τα ιεραρχικά, πολυδιάστατα δεδομένα που μελετώνται σε αυτή τη διατριβή, το αποτέλεσμα μίας

τέτοιας λύσης θα ήταν η πολλαπλή εισαγωγή του ίδιου fact στην επικάλυψη, μία φορά για κάθε επίπεδο της ιεραρχίας και μία φορά για κάθε πιθανό συνδυασμό μεταξύ των διαστάσεων. Η πολλαπλή εισαγωγή του ίδιου fact είναι μία προσέγγιση που μπορεί να ακολουθηθεί στις DHT επικαλύψεις για να εξασφαλιστεί η εξισορρόπηση του φόρτου εργασίας και η ύπαρξη πολλαπλών αντιγράφων που συμβάλλει στην ανεκτικότητα του συστήματος σε σφάλματα. Ωστόσο, η αρνητική επίπτωση της προσέγγισης αυτής είναι οι αυξημένες απαιτήσεις για αποθηκευτικό χώρο στην περίπτωση των πολυδιάστατων, ιεραρχικών δεδομένων, όπου ο αριθμός των πιθανών συνδυασμών είναι μεγάλος και αυξάνει εκθετικά όσο αυξάνει ο αριθμός των διαστάσεων.

Μία άλλη προσέγγιση που ακολουθείται είναι η τοποθέτηση των δεδομένων στους κόμβους του δικτύου με τέτοιο τρόπο, ώστε να επιτυγχάνεται η δρομολόγηση τέτοιου είδους ερωτημάτων στους υπεύθυνους κόμβους με χαμηλό κόστος επικοινωνίας και μικρή κατανάλωση εύρους ζώνης. Σε αυτή την κατηγορία προτεινόμενων λύσεων εντάσσεται η αξιοποίηση των Καμπύλων Πλήρωσης του Χώρου (Space Filling Curves - SFC) για τον ορισμό συναρτήσεων που λαμβάνουν υπόψη τους τις τιμές των ιδιοτήτων για τη δημιουργία των IDs. Σε τέτοιου είδους προσεγγίσεις, θεωρείται ότι οι ιδιότητες σχηματίζουν ένα πολυδιάστατο χώρο, ο οποίος αντιστοιχίζεται μοναδικά σε μια διάσταση και αντίστροφα με τη χρήση μιας SFC. Η προσέγγιση αυτή εκμεταλλεύεται την ιδιότητα των SFCs να δημιουργούν διατάξεις που διατηρούν την τοπικότητα με υψηλή πιθανότητα. Για αυτό το λόγο, τα SFCs έχουν χρησιμοποιηθεί για τη δεικτοδότηση πολυδιάστατων δεδομένων και σε πιο συνηθισμένες εφαρμογές όπως συστήματα βάσεων δεδομένων [Bay97, TH81, LK00] και επεξεργασίας εικόνας. Οι αντιστοιχίσεις που παράγονται από ένα Hilbert SFC παρουσιάζουν καλύτερες δυνατότητες ομαδοποίησης όπως παρατηρείται στο [MJFS01], δηλαδή η τοπικότητα μεταξύ των αντικειμένων στο πολυδιάστατο χώρο διατηρείται καλύτερα μετά την αντιστοίχηση τους στο γραμμικό χώρο. Επίσης, P2P συστήματα για δεικτοδότηση δεδομένων με χρήση των SFC μπορούν να βρεθούν σε σχετικές εργασίες.

Το Squid [SP04] είναι μία Chord επικάλυψη για δεδομένα πολλαπλών ιδιοτήτων και υποστηρίζει μερικές αναζητήσεις κλειδιών και αναζητήσεις με wildcards, καθώς και αναζητήσεις εύρους τιμών. Κάθε data item περιγράφεται από ένα σύνολο από d ιδιότητες και ανατίθεται σε αυτό ένα ID με χρήση μίας συνάρτησης που αντιστοιχίζει τις τιμές των ιδιοτήτων σε ένα Hilbert δείκτη. Εάν ένα ερώτημα περιλαμβάνει εύρη τιμών, τότε το ερώτημα μετατρέπεται μέσω της αντιστοίχισης του Hilbert SFC σε μία σειρά από ερωτήματα διαστημάτων, τα οποία επιλύονται στο Chord δαχτυλίδι. Για την αποφυγή της επίλυσης ενός ερωτήματος με πλημμύρα, ακολουθείται μία αναδρομική διαδικασία που χρησιμοποιεί ένα ενσωματωμένο prefix δέντρο: Σε κάθε βήμα, μεγαλύτερα prefixes από υποψήφιες ομάδες (clusters) αναζητούνται. Εάν ένας κόμβος είναι υπεύθυνος για τα IDs με το prefix που ερωτάται, τότε το ερώτημα προωθείται στην τοπική του βάση. Στην αντίθετη περίπτωση, ξεκινάνε νέα lookups για πιο συγκεκριμένες υποομάδες. Παρά το γεγονός ότι η αναζήτηση των SFC ομάδων γίνεται αναδρομικά και ότι αποκλείονται

οι κόμβοι που δεν περιέχουν σχετικά δεδομένα, η συνολική διαδικασία χαρακτηρίζεται από σημαντικό υπολογιστικό και επικοινωνιακό κόστος, ενώ προκύπτουν προβλήματα που σχετίζονται με την εξισορρόπηση του φόρτου εργασίας. Για την αποφυγή αυτών των προβλημάτων, το Squid εκμεταλλεύεται το γεγονός ότι αναμένεται ο χώρος κλειδιών των d διαστάσεων να είναι αραιός και για αυτό το λόγο τα δεδομένα θα κατανέμονται στους peers με ομοιόμορφο τρόπο.

Το CISS [LLK+07] είναι ένα άλλο πλαίσιο που χρησιμοποιεί SFCs για την αναζήτηση πολυδιάστατων δεδομένων σε online παιχνίδια πολλαπλών παικτών και P2P συστήματα καταλόγων. Σε αυτήν την εργασία, μία ιεραρχική δομή ονοματαδοσίας για καταλόγους θεωρείται. Μία συνάρτηση, η οποία "διατηρεί την τοπικότητα" (locality preserving) και βασίζεται στο Hilbert SFC, παράγει αντιστοιχίσεις από keys πολλαπλών bits σε keys της μιας διάστασης, έτσι ώστε να επιτευχθεί ομαδοποίηση των δεδομένων στους κόμβους για να είναι εφικτή η αποτελεσματική ενημέρωση των δεδομένων και η επίλυση ερωτημάτων για εύρη τιμών σε πολλαπλές διαστάσεις. Το CISS επικεντρώνεται σε ερωτήματα για εύρη τιμών ιδιοτήτων που σχηματίζουν ένα χώρο *δύο* διαστάσεων. Τα εύρη τιμών των διαστάσεων που αναζητούνται αντιστοιχίζονται σε υποψήφια διαστήματα από keys και γίνεται προσπάθεια εύρεσης του κόμβου που είναι υπεύθυνος για το πρώτο key μίας ομάδας. Μόλις βρεθεί ο υπεύθυνος κόμβος προωθεί το ερώτημα στους επόμενους του κόμβους μέχρι να ανακτηθούν όλα τα σχετικά αντικείμενα. Οι συγγραφείς του CISS υιοθετούν μία forwarding-based στρατηγική δρομολόγησης που βελτιστοποιεί τη διαδικασία της επεξεργασίας των ερωτημάτων και αποτρέπει τη συμφόρηση των κόμβων σε αντίθεση με τα αποτελέσματα που πιστεύουν ότι επιφέρει η δρομολόγηση του Squid.

Μία συνάρτηση που στηρίζεται σε SFCs για την ανάθεση των keys μίας διάστασης χρησιμοποιείται και στο SCRAP [GYGM04] και τα παραγόμενα κλειδιά αποθηκεύονται στα peers ενός SkipGraph δικτύου [HJS+03]. Σε αυτό το σύστημα, τα ερωτήματα εύρους τιμών μελετώνται και η δρομολόγηση των ερωτημάτων για υποψήφιες SFC ομάδες γίνεται μέσω των μηχανισμών του SkipGraph. Ωστόσο, ο αριθμός των SFC clusters μπορεί να είναι πολύ μεγάλος και είναι βέβαιο ότι εξαρτάται από το refinement επίπεδο του SFC. Το ZNet [SOTZ05] προτείνει μίας παρόμοια προσέγγιση με αυτή του SCRAP; η κύρια διαφορά είναι ότι ο πολυδιάστατος χώρος χωρίζεται δυναμικά με τη χρήση μίας Z-curve. Είναι πιθανό οι κόμβοι να είναι υπεύθυνοι για συνεχόμενα διαστήματα Z-διευθύνσεων με διαφορετικά μήκη ανάλογα με την τάξη της καμπύλης για το συγκεκριμένο διάστημα. Ένα ερώτημα εύρους τιμών προωθείται αρχικά στους κόμβους που τους έχουν ανατεθεί τα μικρότερα prefixes, πριν υπολογιστεί η επόμενη αναδρομή της καμπύλης για τον καθορισμό υποψήφιων διαστημάτων με μεγαλύτερη λεπτομέρεια.

Η αξιοποίηση των SFCs για την ανάθεση keys μίας διάστασης σε αντικείμενα που περιγράφονται από πολλαπλές ιδιότητες στοχεύει στην τοποθέτηση των δεδομένων στους κόμβους με

τρόπο τέτοιο ώστε να διατηρείται η τοπικότητα τους και επομένως να επιτυγχάνεται η ομαδοποίηση παρόμοιων αντικειμένων. Ωστόσο, σε αυτές τις προσεγγίσεις θεωρείται ότι υπάρχουν οι κατάλληλοι τρόποι κωδικοποίησης για την αντιστοίχιση των τιμών των ιδιοτήτων σε ένα προκαθορισμένο αριθμό σημείων σε κάθε διάσταση. Για διαφορετικούς τύπους ιδιοτήτων (π.χ. αριθμούς, ημερομηνίες, συμβολοσειρές) πρέπει να αναπτυχθούν διαφορετικές μέθοδοι κωδικοποίησης. Η ανάπτυξη αυτών των μεθόδων καθιστά αναγκαία τη γνώση ή την ύπαρξη μίας καλής εκτίμησης των πιθανών τιμών μίας ιδιότητας; διαφορετικά οι μέθοδοι αυτοί είναι ιδιαίτερα αναποτελεσματικές. Ταυτόχρονα, ένα άλλος περιορισμός από τη χρήση των SFC συναρτήσεων είναι ότι όλες οι διαστάσεις πρέπει να χωριστούν στον ίδιο αριθμό διαστημάτων. Το πλήθος τιμών της κάθε διάστασης αυξάνει για μεγαλύτερες προσεγγίσεις (approximations) της καμπύλης, αλλά αυτό έχει ως αποτέλεσμα την αύξηση της πολυπλοκότητας του υπολογισμού των αντιστοιχίσεων, ειδικά όταν υπολογίζονται οι αντιστοιχίσεις μεταξύ διαστημάτων τιμών στον πολυδιάστατο χώρο σε μία σειρά διαστημάτων τιμών στο επίπεδο του SFC. Μειώνοντας τον αριθμό των πιθανών συντεταγμένων των διαστάσεων μπορεί να έχει ως αποτέλεσμα μία εντελώς μη ισορροπημένη ανάθεση αντικειμένων στους κόμβους.

Ακόμα υπάρχουν διάφορες εργασίες που ορίζουν έναν διαφορετικό τρόπο οργάνωσης της δεικτοδότησης πολυδιάστατων δεδομένων σε δομημένες επικαλύψεις. Εκτός από τις συναρτήσεις που χρησιμοποιούν SFC, μερικές εργασίες ορίζουν τις δικές τους συναρτήσεις για την κατανομή των αντικειμένων στα peers. Για παράδειγμα, οι ιδιότητες ενός Kautz γράφου χρησιμοποιούνται από αλγόριθμους για hashing μίας ή περισσότερων ιδιοτήτων που διατηρούν απόλυτη ή μερική ταξινόμηση σε μία δομημένη επικάλυψη που ονομάζεται Armanda [LCLC09]. Κάθε κόμβος στην επικάλυψη είναι υπεύθυνος για ένα συγκεκριμένο διάστημα από συνεχόμενες τιμές μίας ιδιότητας και επιτρέπονται οι αναζητήσεις για διαστήματα τιμών με κοινά prefixes. Μία άλλη κατηγορία συστημάτων που προορίζονται για τέτοιου είδους δεδομένα προτείνουν τις κατάλληλες προεκτάσεις υπάρχοντων DHT, π.χ. του Chord ή του CAN. Το MAAN [CFCS04] επεκτείνει το Chord κατά τέτοιο τρόπο ώστε να υποστηρίζονται ερωτήματα για πολλαπλές ιδιότητες και ερωτήματα εύρους τιμών. Υλοποιεί μία locality-preserving hashing συνάρτηση και εκτελούνται διαφορετικές εγγραφές για κάθε ιδιότητα των πόρων. Το MAAN υποστηρίζει μόνο προκαθορισμένα σχήματα με ένα συγκεκριμένο αριθμό ιδιοτήτων. Το MURK [GYGM04] κατασκευάζει ένα κατανεμημένο kd-tree, ώστε να χωριστεί ο χώρος των δεδομένων σε παραλληλόγραμμα, τα οποία με τη σειρά τους ανατίθενται στους διαθέσιμους κόμβους. Κάθε κόμβος γνωρίζει τους γείτονες του και τα ερωτήματα προωθούνται κατά μήκος των συνδέσμων του. Ένα κατανεμημένο quadtree παρουσιάζεται για τη δεικτοδότηση και επερώτηση χωρικών δεδομένων στο [THS07]. Το κεντροειδές του κάθε quadtree μπλοκ κατακερματίζεται και εισάγεται σε μία Chord επικάλυψη. Σε όλους τους τύπους των προσεγγίσεων που αναφέρθηκαν, ο στόχος προς επίτευξη είναι η αποφυγή της τυχαίας ανάθεσης των δεδομένων στους κόμβους μία δομημένης επικάλυψης

και η διατήρηση μίας ταξινόμησης στον τρόπο που αποθηκεύονται οι τιμές. Επίσης, οι εργασίες αυτές επικεντρώνονται κυρίως στην αποτελεσματική επίλυση ενός συγκεκριμένου τύπου ερωτημάτων και απαιτούν την ανάπτυξη επιπρόσθετων δομών για την επαναφορά χαρακτηριστικών που υποστηρίζονται πρωταρχικά από το DHT, όπως η λογαριθμική δρομολόγηση, η αναδιοργάνωση της τοπολογίας του δικτύου μετά από τις αφίξεις και αναχωρήσεις κόμβων, η εξισορρόπηση του φόρτου εργασίας και η δημιουργία πολλαπλών αντιγράφων.

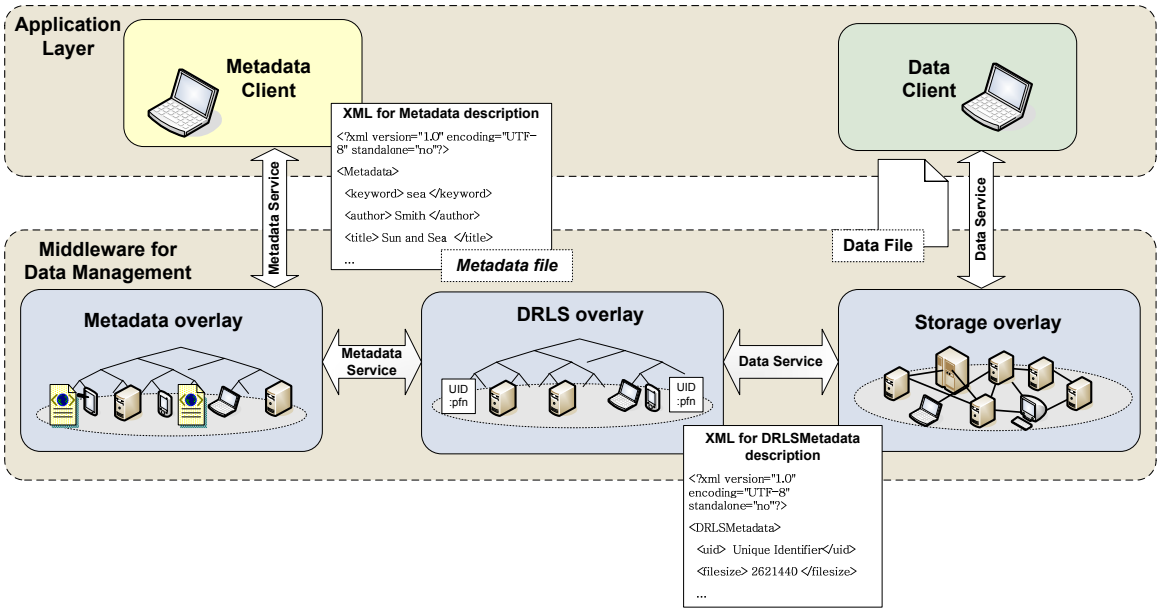
2.6 Τεχνολογίες Πλέγματος

Οι *τεχνολογίες Πλέγματος* (*Grid computing*) [FK99], [FK03] επικεντρώνονται στη μελέτη κατανεμημένων υποδομών, όπου οι χρήστες διαμοιράζονται γεωγραφικά κατανεμημένους πόρους που ενοποιούνται από ένα κοινό μεσσισμικό (middleware). Οι τεχνολογίες αυτές υιοθετούν τις αρχές του μοντέλου λογισμικού που ονομάζεται *Service Oriented Architecture* (SOA) και παρέχει απρόσκοπτη πρόσβαση στις υπηρεσίες που εκτελούνται με κατανεμημένο τρόπο. Το βασικό χαρακτηριστικό των συστημάτων αυτών είναι η ύπαρξη σαφώς καθορισμένων κανόνων και πολιτικών που επιτρέπουν τον ευέλικτο, ασφαλή και συντονισμένο διαμοιρασμό πόρων σε δυναμικές, εικονικές ομάδες χρηστών, οι οποίες ονομάζονται *Εικονικοί Οργανισμοί* (*Virtual Organizations* - VO). Ένα VO είναι ένα σύνολο από ανεξάρτητα άτομα ή/και οργανισμούς που ορίζονται από αυστηρώς καθορισμένους κανόνες διαμοιρασμού. Ανάμεσα στους πάροχους και στους καταναλωτές των πόρων ορίζεται σαφώς τι διαμοιράζεται, πως γίνεται η πρόσβαση στους διαμοιραζόμενους πόρους και τις συνθήκες κάτω από τις οποίες λαμβάνει χώρα ο διαμοιρασμός [FK99].

Το middleware του Πλέγματος είναι υπεύθυνο για την απόκρυψη της ετερογένειας και της πολυπλοκότητας της υποκείμενης φυσικής υποδομής και παρέχει εννιαία πρόσβαση στους χρήστες. Εκτός από αυτό, το middleware αποτελείται από υπηρεσίες που είναι κρίσιμες για τη λειτουργία της υποδομής, όπως ο συντονισμός της χρήσης των πόρων, η παρακολούθηση και η καταγραφή της κατάστασης της υποδομής, η διαχείριση των δεδομένων, η εκτέλεση των εργασιών και άλλες βασικές λειτουργίες που σχετίζονται με την υποδομή. Αν και η εκτέλεση των εργασιών και η διαχείριση των δεδομένων μπορεί να γίνεται σε πολλαπλά site, ο σχεδιασμός κάποιων βασικών λειτουργιών γίνεται αναγκαστικά με κεντρικοποιημενο τρόπο. Για την αποφυγή υπερφορτομένων κόμβων, την επίτευξη καλύτερης διαθεσιμότητας των υπηρεσιών και πιο γρήγορων χρόνων απόκρισης δημιουργούνται πολλαπλές υποστάσεις της ίδιας υπηρεσίας. Ωστόσο, η στρατηγική αυτή έχει αποδειχθεί μη αποτελεσματική σε πολλές περιπτώσεις καταδεικνύοντας την ανάγκη για την ανάπτυξη πιο κατανεμημένων λύσεων.

Οι υποδομές Πλέγματος έχουν χρησιμοποιηθεί ευρέως για την εκτέλεση πολύπλοκων επιστημονικών και επιχειρησιακών εφαρμογών. Οι εφαρμογές αυτές έχουν ανάγκη για μία υποδομή που τους επιτρέπει να εκτελούν πολύπλοκους υπολογισμούς σε ευρείας κλίμακας και κατανεμημένα σύνολα δεδομένων, αλλά και την αποθήκευση και διάδοση τους σε πολλαπλά site. Ανάλογα με το στόχο που έχει τεθεί, τα συστήματα Πλέγματος μπορούν να κατηγοριοποιηθούν βασικά σε *Υπολογισικά Πλέγματα* (*Computational Grids*) και *Πλέγματα Δεδομένων* (*Data Grids*). Σύμφωνα με το [FKT01], ένα Computational Grid σύστημα ορίζεται ως ένα περιβάλλον που αποτελείται από ένα ή περισσότερα υλικά και λογισμικά περιβάλλοντα που παρέχουν αξιόπιστη, συνεπή και μη δαπανηρή πρόσβαση σε high-end υπολογιστικές δυνατότητες. Σε αυτά τα περιβάλλοντα δίνεται μεγαλύτερη έμφαση στον απρόσκοπτο διαμοιρασμό των πόρων τη δημιουργία εικονικών οργανισμών που επιτρέπουν τη συνεργασία, ενώ η πλειοψηφία των πόρων παρέχουν κυρίως υπολογιστική ισχύ. Τα *Data Grids* είναι κατανεμημένες υποδομές ευρείας κλίμακας που αποτελούνται από ετερογενείς πόρους και είναι ικανές να διαχειρίζονται μεγάλους όγκους δεδομένων. Οι βασικές υπηρεσίες για την πρόσβαση σε ετερογενείς, αποθηκευτικούς πόρους, την αποθήκευση, τη μεταφορά και την αναζήτηση δεδομένων μεγάλου όγκου περιγράφονται στο [CFK+00]. Στην πραγματικότητα συναντώνται υποδομές που συνδυάζουν χαρακτηριστικά και από τις δύο κατηγορίες ώστε να ικανοποιούνται οι ανάγκες των χρηστών.

Ωστόσο, τα υπολογιστικά πλέγματα που χρησιμοποιούνται κυρίως για τη διαχείριση των δεδομένων έχουν αναγνωριστεί ως υποψήφιες πλατφόρμες, όπου μπορούν να ενσωματωθούν μηχανισμοί από τις DHT επικαλύψεις ώστε να παρέχουν αποδοτικές και επεκτάσιμες υπηρεσίες για την ανακάλυψη, αναζήτηση και μεταφορά των αποθηκευμένων δεδομένων. Μερικές από τις βασικές υπηρεσίες μπορούν να εκτελεστούν μόνο ως μεμονωμένες υποστάσεις δεδομένου ότι απαιτούν τη γνώση πληροφορίας για τη συνολική κατάσταση της υποδομής κατά την εκτέλεση τους. Ο σχεδιασμός και η ανάπτυξη τέτοιων συστημάτων μπορεί να επωφεληθεί από μεθοδολογίες που χρησιμοποιούνται στα P2P δίκτυα για τη διαχείριση δεδομένων μεγάλου όγκου, οι οποίες οδηγούνται σε λύσεις που εγγυώνται τη διαθεσιμότητα των δεδομένων όταν αυξάνει ο όγκος τους ή ο αριθμός των αιτημάτων. Οι μεθοδολογίες αυτές μπορούν να χρησιμοποιηθούν για τη μετατροπή των κεντρικοποιημένων υπηρεσιών σε επεκτάσιμες, κατανεμημένες και αξιόπιστες υπηρεσίες.



Σχήμα 2.3: *Επισκόπηση της αρχιτεκτονικής του middleware για τη διαχείριση των δεδομένων του Gredia*

2.7 Εκμετάλλευση των Καμπύλων Πλήρωσης του Χώρου για τη Δημιουργία ενός Κατανεμημένου Κατάλογου Μεταδεδομένων

Οι ιδιότητες των SFCs μπορούν να χρησιμοποιηθούν για δεδομένα που περιγράφονται από πολλαπλές διαστάσεις, ενώ η σημασιολογία αυτών των τιμών διατηρούνται. Όπως έχει ήδη συζητηθεί, πολλές εργασίες στη βιβλιογραφία εξετάζουν πως μπορεί να γίνει η διαχείριση τέτοιου είδους δεδομένων με αποτελεσματικό τρόπο και η επίλυση των exact match ερωτημάτων αλλά και των ερωτημάτων εύρους τιμών γρήγορα και με ακρίβεια.

Η ενότητα αυτή περιγράφει μία τεχνική που προτάθηκε για την αξιοποίηση των ιδιοτήτων των SFCs ώστε να σχεδιαστεί μία επεκτάσιμη και κατανεμημένη υπηρεσία για τη διαχείριση μεταδεδομένων (Metadata service), η οποία χρησιμοποιήθηκε και στην πλατφόρμα Gredia [GRE]. Η πλατφόρμα αυτή παρέχει μία υποδομή που βασίζεται σε τεχνολογίες Πλέγματος και επιτρέπει στους προγραμματιστές, στους επαγγελματίες και στους χρήστες να διαμοιράζονται πολυμεσικό υλικό που έχει περιγράφει από μεταδεδομένα. Μια επισκόπηση της αρχιτεκτονικής του middleware που αναπτύχθηκε για την πλατφόρμα Gredia απεικονίζεται στο Σχήμα 2.3.

Το Metadata service που φιλοξενείται από τη *Metadata* επικάλυψη παρέχει ένα μηχανισμό αναζήτησης για τις περιγραφές του πολυμεσικού υλικού που αποθηκεύεται στην πλατφόρμα

Gredia. Κάθε αρχείο δεδομένων που περιέχει το πολυμεσικό υλικό περιγράφεται από ένα σύνολο ιδιοτήτων που ακολουθούν ένα προκαθορισμένο σχήμα μεταδεδομένων που σχεδιάστηκε σύμφωνα με τις ανάγκες των χρηστών. Το πολυμεσικό υλικό περιγράφεται από πολλαπλές διαστάσεις που θεωρούνται σημαντικές για το χαρακτηρισμό του αρχείου και την αναζήτηση του. Οι τιμές των μεταδεδομένων του κάθε αρχείου δεδομένων περιλαμβάνονται στο *αρχείο μεταδεδομένων* (*metadata file*). Οι χρήστες πρέπει να είναι ικανοί να εκτελούν point ερωτήματα και ερωτήματα εύρους τιμών για τις τιμές των πολλαπλών ιδιοτήτων που δεικτοδοτούν κάθε αρχείο.

Ο ρόλος του Metadata service είναι πολύ σημαντικός για την αναζήτηση και την ανάκτηση του πραγματικού περιεχομένου. Η προσέγγιση που ακολουθείται συνήθως στα συστήματα Πλέγματος είναι η δημιουργία ενός κεντρικού καταλόγου για κάθε VO, ο οποίος συγκεντρώνει την πληροφορία για τα αποθηκευμένα αρχεία των χρηστών και χαρακτηρίζεται από περιορισμένες δυνατότητες. Όσο αυξάνει ο αριθμός των χρηστών και ο όγκος των δεδομένων, η συγκεκριμένη λύση αποτυγχάνει να εξυπηρετήσει αποτελεσματικά τα αιτήματα των χρηστών. Στη προτεινόμενη προσέγγιση, τα μεταδεδομένα αποθηκεύονται στους κόμβους που συμμετέχουν σε μία DHT επικάλυψη που δομείται σύμφωνα με το Kademlia πρωτόκολλο και ο αριθμός των peers που επισκέπτονται τα ερωτήματα μειώνεται δυναμικά και συνεπώς συμβαίνει το ίδιο και στην κατανάλωση του εύρους ζώνης που απαιτείται για τη συλλογή των αποτελεσμάτων. Επιπλέον, η ανεκτικότητα του συστήματος σε σφάλματα εξασφαλίζεται κατευθείαν από τους μηχανισμούς δημιουργίας πολλαπλών αντιγράφων από τους μηχανισμούς της ίδιας της DHT επικάλυψης χωρίς να επηρεάζει την αποτελεσματικότητα των μηχανισμών επεξεργασίας των ερωτημάτων.

Η προσέγγιση που ακολουθείται για να καταστεί δυνατή η επίλυση των point ερωτημάτων και των ερωτημάτων εύρους τιμών για πολυδιάστατα δεδομένα βασίζεται στη βασική ιδιότητα των SFCs. Ένα SFC αντιστοιχεί συνεχόμενα ένα συμπαγές διάστημα στο χώρο των d διαστάσεων και το αντίστροφο. Επίσης τα SFC διατηρούν την τοπικότητα, ώστε τα σημεία στο χώρο της μίας διάστασης να αντιστοιχίζονται σε κοντινά σημεία στο χώρο των d διαστάσεων.

Τα SFCs χρησιμοποιούνται στη *Metadata* επικάλυψη ώστε να επηρεαστεί η τοποθέτηση των δεδομένων στο DHT χωρίς να αλλάξει ο αλγόριθμος δρομολόγησης του DHT πρωτοκόλλου. Ο στόχος προς επίτευξη είναι να καταλήξουν τα αρχεία των metadata με παρεμφερείς τιμές ιδιοτήτων σε κόμβους με κοντινά IDs, ώστε να μειωθεί το flooding στο δίκτυο για τα ερωτήματα εύρους ζώνης.

Στην προτεινόμενη τεχνική δεικτοδότησης θεωρείται ότι το σύνολο των d ιδιοτήτων προς δεικτοδότηση σχηματίζει ένα χώρο των d διαστάσεων. Κάθε σημείο στο χώρο αναπαριστά ένα συνδυασμό τιμών για τις ιδιότητες που δεικτοδοτούνται. Τα σημεία του χώρου των d διαστάσεων αντιστοιχίζονται σε μία διάσταση από ένα SFC, όπως η Hilbert καμπύλη ή το Z καμπύλη. Το αποτέλεσμα είναι ο χωρισμός του χώρου των d διαστάσεων σε 2^{kd} κελιά, τα οποία με τη σειρά τους αντιστοιχίζονται μέσω του SFC σε 2^{kd} σημεία της μίας διάστασης. Αναλυτικότερα, ο χώρος των d διαστάσεων χωρίζεται σε d κύβους που αντιστοιχίζονται σε διαστήματα του SFC. Χωρίς

απώλεια της γενικότητας, κάθε υπερκύβος θεωρείται ως ένα σημείο του χώρου των d διαστάσεων και το αντίστοιχο διάστημα στο SFC ως ένα σημείο της μίας διάστασης.

Μία βασική ιδιότητα των SFCs είναι η αναδρομική δημιουργία τους. Αρχικά, ο χώρος των d διαστάσεων χωρίζεται σε 2^d κελιά που αντιστοιχίζονται στο First Order Curve, όπου θεωρείται ότι το k ισούται με ένα. Στην επόμενη αναδρομή, κάθε κελί χωρίζεται 2^d φορές ακόμα και αντιστοιχίζεται στο Second Order Curve. Ο αριθμός των αναδρομών σχετίζεται με τον παράγοντα k που ονομάζεται *approximation order*. Ο παράγοντας αυτός καθορίζει τον αριθμό των διαμερίσεων του χώρου και συνεπώς την ακρίβεια του αλγορίθμου.

Οι τιμές που προκύπτουν στη μία διάσταση αποτελούν το σύνολο των keys της Kademlia επικάλυψης. Κάθε key αποτελείται από kd bits και κάθε κόμβος της επικάλυψης διαχειρίζεται δεδομένα ενός συνεχόμενου διαστήματος του SFC. Η αντιστοίχιση ενός συνδυασμού τιμών σε ένα συγκεκριμένο key γίνεται από το SFC module, το οποίο υπολογίζει το αποτέλεσμα της hash συνάρτησης που χρησιμοποιεί το SFC για να παράγει το key που θα αποθηκευτεί στη DHT επικάλυψη. Το παραγόμενο key διατηρεί την πληροφορία που περιλαμβάνεται στις δοσμένες τιμές και η αντιστοίχιση εκτελείται μεταξύ των τιμών του χώρου των d διαστάσεων στη μία διάσταση και αντίστροφα.

Το SFC module μπορεί να χρησιμοποιεί είτε τη Hilbert καμπύλη είτε τη Z καμπύλη. Το Hilbert SFC εμφανίζει καλύτερες ιδιότητες ομαδοποίησης [MAK03], [MJFS01]. Η αντιστοίχιση ενός σημείου του χώρου των d διαστάσεων στη θέση του στο SFC και αντιστοίχως δεν είναι μία απλή διαδικασία και η δυσκολία αυξάνεται ανάλογα με τον αριθμό των διαστάσεων. Ο υπολογισμός των δεικτών του SFC μπορεί να γίνει μη αναδρομικά με τη χρήση κυκλικών ολισθήσεων και λειτουργίες αποκλειστικού-η (exclusive-or) στα bytes σύμφωνα με τον αλγόριθμο του Butz [But71]. Η διαδικασία της αντιστοίχισης από τον χώρο των d διαστάσεων στη μία διάσταση και αντίστροφα έχει υλοποιηθεί σύμφωνα με αυτόν τον αλγόριθμο. Η διάσχιση του SFC γίνεται με τον υπολογισμό της καμπύλης αναδρομικά. Το Hilbert SFC μπορεί να προσεγγιστεί σε μία υψηλότερη τάξη με ένα συνδυασμό των First Order Curves που έχουν προσανατολιστεί κατάλληλα. Ο υπολογισμός των αντιστοιχίσεων υπολογίζεται με τη Z καμπύλη είναι απλούστερος και μια αντιστοίχιση υπολογίζεται με τη διεμπλοκή των δυαδικών ψηφίων των δεικτών από κάθε διάσταση.

Ένα πρόβλημα που πρέπει να αντιμετωπιστεί κατά την εφαρμογή των προτεινόμενων τεχνικών δεικτοδότησης είναι η κατάτμηση κάθε διάστασης σύμφωνα με τον αριθμό των πιθανών τιμών για κάθε ιδιότητα. Η μέθοδος που βασίζεται σε SFC και καθιστά δυνατή την επίλυση ερωτημάτων εύρους τιμών προϋποθέτει δείκτες με σταθερό αριθμό bits; για αυτό το λόγο χρειάζονται συναρτήσεις αντιστοίχισης των πιθανών τιμών μίας ιδιότητας στις διαθέσιμες τιμές. Οι τύποι των ιδιοτήτων που αναγνωρίστηκαν για τη συγκεκριμένη εφαρμογή είναι *συμβολοσειρές*, *ημερομηνίες* και *κατηγορίες*.

Οι ιδιότητες που είναι τύπου κατηγορίας είναι ευκολότερα διαχειρίσιμες, δεδομένου ότι τα εύρη των πιθανών τιμών είναι γνωστά εκ των προτέρων και μπορούν να αντιστοιχιθούν στους διαθέσιμους δείκτες. Αν και τα ερωτήματα εύρους τιμών δεν είναι μία συνηθισμένη περίπτωση για τις κατηγορικές ιδιότητες, η αναπόφευκτη διάταξη των τιμών που επιβάλλεται στο επίπεδο της διάστασης επιτρέπει την ανάκτηση των αρχείων μεταδεδομένων που περιέχουν περισσότερες από μία κατηγορίες. Σε αυτήν την περίπτωση, το ερώτημα αναδιατυπώνεται από το metadata module σύμφωνα με τη σύμβαση που ακολουθείται για τη διάταξη των τιμών κατά τη φάση της κωδικοποίησης. Είναι πιθανό ότι πολλαπλές διακριτές τιμές συνενώνονται σε ένα ή περισσότερα διαστήματα όταν ένα ερώτημα φτάνει στην επικάλυψη.

Μία απλή μέθοδος ανάλογα με τους χαρακτήρες έχει αναπτυχθεί για τον υπολογισμό της αναπαράστασης σε bit των ιδιοτήτων που μπορούν να κατηγοριοποιηθούν ως συμβολοσειρές ή ημερομηνίες. Ωστόσο, ο αριθμός των χαρακτήρων που κωδικοποιούνται και συμπεριλαμβάνονται στο δείκτη μίας διάστασης εξαρτάται από τον αριθμό των bits που είναι διαθέσιμα για την ιδιότητα στο επίπεδο της επικάλυψης. Λαμβάνοντας υπόψη ότι ο αριθμός των bits των δεικτών της κάθε διάστασης είναι ίσος, το πλήθος των τιμών του SFC κυριαρχείται από την πολυπληθέστερη ιδιότητα. Για να αποφευχθεί η υπερβολική αύξηση του αριθμού των bits που χρησιμοποιούνται για την αναπαράσταση των τιμών των ιδιοτήτων επιλέγεται το approximation order της καμπύλης έτσι ώστε να συνεπάγεται έναν λογικό αριθμό από bits – ανάλογα και με τις ανάγκες της εφαρμογής – και οι τιμές που κωδικοποιούνται περικόπτονται αντίστοιχα. Η τεχνική της κωδικοποίησης έχει ως αποτέλεσμα την αλφαβητική διάταξη των συμβολοσειρών στις αντίστοιχες διαστάσεις. Ο στόχος είναι να αποφευχθεί η χρήση ενός μεγάλου αριθμού bits και έτσι οι αναζητήσεις αναφέρονται σε πιο περιορισμένους χώρους αναζήτησης; άρα οι αναζητήσεις καταλήγουν σε λιγότερους κόμβους και εξυπηρετούνται πιο αποτελεσματικά. Περαιτέρω φιλτράρισμα μπορεί να ακολουθήσει σε περίπτωση που η τιμή της συμβολοσειράς περιέχει περισσότερους χαρακτήρες από τον επιτρεπόμενο αριθμό. Για να αποφευχθεί η υπερβολική αύξηση του αριθμού των ιδιοτήτων που δεικτοδοτούνται και συνεπώς η πολυπλοκότητα του υπολογισμού των απεικονίσεων, τα πιο κρίσιμα χαρακτηριστικά πρέπει να επιλεχθούν για να δεικτοδοτηθούν σύμφωνα με τη συχνότητα αναζήτησης τους από τους χρήστες. Φυσικά, τα υπόλοιπα χαρακτηριστικά μπορούν να συμπεριληφθούν στο αρχείο των μεταδεδομένων ώστε να επιλέξιμα στο επίπεδο της εφαρμογής (δηλαδή, τα χαρακτηριστικά που ορίζονται από το MPEG-7 πρότυπο [MPE7] για πολυμεσικό περιεχόμενο).

Η χρήση του SFC module δεν επηρεάζει τη διαδικασία της εισαγωγής ενός αρχείου μεταδεδομένων στην επικάλυψη. Ένα key δημιουργείται βάσει των τιμών των ιδιοτήτων που έχουν επιλεχθεί να δεικτοδοτηθούν και διατηρεί τη σημασιολογική πληροφορία αυτών των τιμών. Τέλος, το αρχείο καταλήγει στον κόμβο που είναι υπεύθυνος για αυτό το key.

Η ανάκτηση του αρχείου μεταδεδομένων είναι εφικτή μόνο εάν είναι γνωστό το ακριβές key που χρησιμοποιήθηκε κατά την εισαγωγή του στην επικάλυψη. Αυτό σημαίνει ότι η λειτουργία

lookup του DHT είναι πολύ περιοριστική. Εάν ένας συνδυασμός των δεικτοδοτημένων τιμών για ένα συγκεκριμένο αρχείο είναι γνωστός εκ των προτέρων, τότε το SFC module παράγει το key που χρησιμοποιήθηκε κατά τη φάση της εισαγωγής του αρχείου και το DHT lookup επιστρέφει το αναζητούμενο αρχείο.

Ωστόσο, η λειτουργία της αναζήτησης δεν είναι συνήθως επαρκής και οι DHT μηχανισμοί πρέπει να επεκταθούν και να εμπλουτιστούν για την υποστήριξη πολυπλοκότερων ερωτημάτων. Για παράδειγμα, όταν τουλάχιστον μία τιμή των δεικτοδοτημένων ιδιοτήτων δεν προσδιορίζεται, τότε όλες οι τιμές της διάστασης αυτής αναζητούνται. Ένα διάστημα τιμών σε μία ή περισσότερες διαστάσεις μπορεί να περιοριστεί εάν καθοριστεί μία αρχική και μία τελική τιμή για το αντίστοιχο χαρακτηριστικό (ή χαρακτηριστικά). Λαμβάνοντας υπόψη τις ιδιότητες των SFCs, τα εύρη τιμών στις διαστάσεις του πολυδιάστατου χώρου μπορούν να μετατραπούν σε διαστήματα τιμών στο SFC. Ωστόσο, πρέπει να θεωρηθεί το γεγονός ότι ένα απλό ερώτημα που περιέχει μόνο ένα εύρος ερωτούμενων τιμών σε κάποια διάσταση μπορεί να παράγει πολλαπλά μη συνεχόμενα τμήματα στην καμπύλη, ενώ μερικά από αυτά τα τμήματα μπορεί να περιέχουν μόνο ένα identifier.

Η επίλυση των ερωτημάτων εύρους τιμών αποτελείται από δύο βασικές φάσεις. Στην πρώτη φάση προσδιορίζονται οι ομάδες των SFC σημείων που απαντούν το ερώτημα. Στην επόμενη φάση, οι λειτουργίες lookup για αυτές τις ομάδες σημείων ξεκινούν. Ο προσδιορισμός των τμημάτων του SFC μπορεί να είναι πολύπλοκος και υπολογιστικά απαιτητικός και για αυτό το λόγο ο υπολογισμός των υποψηφίων τμημάτων μπορεί να γίνει χρησιμοποιώντας πληροφορία από τους κόμβους που επισκέφτηκε το ερώτημα. Θεωρώντας ότι το churn δε χρειάζεται να αντιμετωπιστεί στις εφαρμογές που μελετώνται, κάθε κόμβος γνωρίζει τον κόμβο με το κοντινότερο ID στο δίκτυο και διατηρεί ένα δείκτη προς αυτό. Κάθε φορά που ένα ερώτημα για το αρχικό ID του εύρους τιμών φτάνει σε έναν κόμβο, ο κόμβος αυτός σαρώνει τα keys του και τα αρχεία μεταδεδομένων που αποθηκεύονται τοπικά και επιστρέφει αυτά που απαντάνε το ερώτημα. Επισημαίνεται ότι η δεικτοδοτημένη τιμή αποθηκεύεται σε μία τοπική βάση δεδομένων, ώστε όταν ένας κόμβος λάβει ένα ερώτημα μπορεί να εντοπίσει εύκολα τα αρχεία των μεταδεδομένων που απαντούν στο ερώτημα με μία απλή αναζήτηση στην τοπική βάση δεδομένων. Εάν ένας κόμβος είναι υπεύθυνος για ένα τμήμα του ζητούμενου διαστήματος τιμών, τότε προωθεί το ερώτημα στον επόμενο κόμβο. Η προώθηση του ερωτήματος συνεχίζει μέχρι τα αντικείμενα που ανήκουν στο διάστημα των ζητούμενων IDs να ανακτηθούν. Μετά από αυτό, το επόμενο εύρος IDs υπολογίζεται, το οποίο περιέχει υποψήφια IDs που απαντούν το ερώτημα και ανήκουν σε κόμβους που δεν έχει επισκεφτεί ήδη το ερώτημα. Η διαδικασία της αναζήτησης για αυτό το εύρος IDs διενεργείται όπως έχει ήδη περιγραφεί.

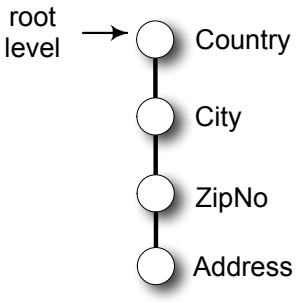
Συμπερασματικά, αποδείχθηκε ότι η τοποθέτηση των δεδομένων και η δεικτοδότηση των αποθηκευμένων τιμών μπορεί να συντελέσουν στην κατασκευή μίας προηγμένης μηχανής αναζήτησης για πολύπλοκα ερωτήματα, ενώ το υπολογιστικό και διαχειριστικό κόστος μπορεί να

ελαχιστοποιηθεί όταν αξιοποιηθούν οι DHT μηχανισμοί που οδηγούν σε επεκτάσιμες και ανεκτικές σε σφάλματα λύσεις.

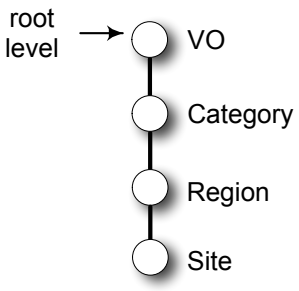
ΚΕΦΑΛΑΙΟ 3

Προσαρμοστικές Μέθοδοι για την Κατανομή και Αναζήτηση Εννοιολογικών Ιεραρχιών

Το Κεφάλαιο 3 παρουσιάζει ένα σύστημα για την επεξεργασία μαζικών, ιεραρχικών δεδομένων (bulk data) σε δομημένες επικαλύψεις που χρησιμοποιούν Κατανεμημένους Πίνακες Κατακερματισμού (DHT επικαλύψεις). Οι εννοιολογικές ιεραρχίες συνεισφέρουν σημαντικά στην οργάνωση και επαναχρησιμοποίηση της πληροφορίας και χρησιμοποιούνται ευρέως σε μια πληθώρα εφαρμογών στα πληροφοριακά συστήματα. Ο στόχος προς επίτευξη είναι η αποτελεσματική αποθήκευση και επερώτηση δεδομένων που οργανώνονται σε εννοιολογικές ιεραρχίες και κατανέμονται σε μία DHT επικάλυψη. Η χρησιμοποίηση μίας DHT επικάλυψης δίνει τη δυνατότητα της "άμεσης" αντιμετώπισης ζητημάτων που προκαλούνται από τις αφίξεις/αναχωρήσεις των κόμβων (node churn), παρέχει μηχανισμούς για τη δημιουργία πολλαπλών αντιγράφων (replication) και προσφέρει μία αξιόπιστη λύση για την αποθήκευση δεδομένων με κατανεμημένο τρόπο. Εκτός από την αξιοποίηση των ιδιοτήτων των DHT επικαλύψεων, οι προτεινόμενες τεχνικές προσφέρουν τη δυνατότητα για προσαρμοστική δεικτοδότηση ανάλογα με το επίπεδο λεπτομέρειας των εισερχόμενων ερωτημάτων και για online ενημέρωση των δεδομένων με χαμηλό κόστος, ενώ δεν απαιτείται η διακοπή της λειτουργίας του συστήματος. Οι προτεινόμενοι μηχανισμοί για τη προσαρμογή της δεικτοδότησης επιτρέπουν στους κόμβους να αποφασίζουν μεμονωμένα το επίπεδο της δεικτοδότησης ανάλογα με τα εισερχόμενα ερωτήματα. Οι λειτουργίες roll-up και drill-down που ορίστηκαν για την προσαρμογή της δεικτοδότησης εκτελούνται στο επίπεδο του κόμβου (on a per node basis) για την ελαχιστοποίηση του απαιτούμενου εύρους ζώνης και την αποτελεσματική



Σχήμα 3.1: *Μία εννοιολογική ιεραρχία για το* "Location".



Σχήμα 3.2: *Μία εννοιολογική ιεραρχία με το "VO" ως root level.*

επίλυση ερωτημάτων που αφορούν διαφορετικά επίπεδα λεπτομέρειας. Οι μέθοδοι που χρησιμοποιήθηκαν στο προκύπτον σύστημα εφαρμόζονται για την κατανεμημένη εκτέλεση του Πληροφοριακού Συστήματος μίας υποδομής Πλέγματος (Grid Information System - GIS). Στην περίπτωση αυτή, ένα πλήρως αποκεντρωμένο σύστημα προτείνεται για τη δημιουργία, την επερώτηση και την ενημέρωση μεγάλων όγκων ιεραρχικών δεδομένων online. Το σύστημα αυτό μπορεί να θεωρηθεί ως μία εναλλακτική λύση σε σύγκριση με τα "παραδοσιακά" κεντρικοποιημένά συστήματα που προσανατολίζονται σε τέτοιου είδους εφαρμογές και δεικτοδοτούν πλήρως όλα τα δεδομένα. Τα εκτενή πειραματικά αποτελέσματα υποστηρίζουν το επιχείρημα ότι οι προτεινόμενες τεχνικές αποδεικνύονται πολύ αποτελεσματικές σε ποικίλα φορτία, τα οποία μπορεί να είναι πολωμένα είτε προς ένα επίπεδο είτε προς πολλαπλά επίπεδα ταυτόχρονα. Το προκύπτον σύστημα μπορεί να προσαρμόζεται στις ξαφνικές αλλαγές της δημοτικότητας των ζητούμενων τιμών, ενώ αποθηκεύει και ενημερώνει μεγάλους όγκους δεδομένων με μικρός κόστος.

3.1 Εισαγωγή

Τα P2P συστήματα χρησιμοποιούνται ως μία κατανεμημένη και αξιόπιστη λύση για τη διαχείριση μεγάλου όγκου δεδομένων. Για αυτό το λόγο, η υιοθέτηση των P2P επικαλύψεων συνιστάται για το σχεδιασμό κατανεμημένων συστημάτων ευρείας κλίμακας που προορίζονται για τη διαχείριση δεδομένων μεγάλου όγκου και για την παροχή λειτουργικότητας αντίστοιχης με αυτήν που συναντάται στα συστήματα βάσεων δεδομένων.

Σε πολλές εφαρμογές, τα δεδομένα συνήθως προσδιορίζονται από πολλαπλές ιδιότητες (ή διαστάσεις), που μπορούν να δομηθούν με διαφορετικά επίπεδα λεπτομέρειας με τη χρήση εννοιολογικών ιεραρχιών. Μία *εννοιολογική ιεραρχία* (ή *ταξινομία*) προσδιορίζει μία ακολουθία απεικονίσεων από γενικότερες σε ειδικότερες έννοιες. Για παράδειγμα, το Σχήμα 3.1 απεικονίζει τη διάταξη των αντίστοιχων εννοιών για το *Location* από το γενικότερο επίπεδο που αντιστοιχεί στο `Country` προς το επίπεδο μεγαλύτερης λεπτομέρειας, δηλαδή το επίπεδο `Address`. Παρόλα αυτά, σε πολλές εφαρμογές συναντώνται εννοιολογικές ιεραρχίες με "μερική διάταξη" (*partial*

ordering), όπως η εννοιολογική ιεραρχία `Time` που περιγράφει το χρόνο και απεικονίζεται στο Σχήμα 3.3. Θεωρείται ότι το προτεινόμενο σύστημα δεν υποστηρίζει αυτήν την κατηγορία εννοιολογικών ιεραρχιών. Όταν οι προτεινόμενες τεχνικές χρησιμοποιούνται σε εφαρμογές για τη διαχείριση δεδομένων που ακολουθούν εννοιολογικές ιεραρχίες με μερική δομή, τότε πρέπει να γίνει μία αναδιάταξη των επιπέδων ώστε τα επίπεδα να είναι πλήρως διατεταγμένα, όπως φαίνεται στο Σχήμα 3.4 για την ιεραρχία του `Time`. Η διάταξη των επιπέδων στις εννοιολογικές ιεραρχίες προσδιορίζεται συνήθως από ειδικούς για το είδος της εφαρμογής ή για τον τομέα στον οποίο ανήκει. Η οργάνωση της πληροφορίας σύμφωνα με μία ακολουθία απεικονίσεων από γενικότερες σε ειδικότερες έννοιες είναι σημαντική τόσο για την αναζήτηση της όσο και για την επαναχρησιμοποίηση της. Τα δεδομένα μπορούν να αναζητηθούν σε διαφορετικά επίπεδα λεπτομέρειας, ενώ τα διάφορα επίπεδα σύνοψης υποστηρίζονται από την ίδια τη δομή των δεδομένων.

Αν και έχει σημειωθεί πρόοδος στον αποτελεσματικό διαμοιρασμό απλών σχεσιακών δεδομένων σε δομημένες και αδόμητες επικαλύψεις (όπως για παράδειγμα στις εργασίες [HHB+03], [OTZ+03]), δεν έχει δοθεί ιδιαίτερη προσοχή σε δεδομένα που υποστηρίζουν τη χρήση ιεραρχιών. Ειδικότερα, αν θεωρηθεί ότι μία DHT επικάλυψη χρησιμοποιείται για τη δημιουργία ενός κατανεμημένου συστήματος ευρείας κλίμακας που προορίζεται για τέτοιου είδους δεδομένα, το πρόβλημα που πρέπει να αντιμετωπιστεί είναι η ανάπτυξη των απαραίτητων τεχνικών και μηχανισμών για τη διαχείριση και επεξεργασία δεδομένων με εννοιολογικές ιεραρχίες.

Η ιδιαιτερότητα που παρουσιάζεται κατά την εισαγωγή πλειάδων που περιέχουν τιμές για κάθε επίπεδο μίας εννοιολογικής ιεραρχίας είναι ότι η αποθήκευση και η δεικτοδότηση τους πρέπει να γίνει με τέτοιο τρόπο ώστε να είναι εφικτή η επίλυση ερωτημάτων για κάθε επίπεδο της ιεραρχίας. Ωστόσο, οι DHT επικαλύψεις υποστηρίζουν μόνο τον εντοπισμό των κλειδιών που έχουν χρησιμοποιηθεί κατά την εισαγωγή των αντικειμένων σε αυτές. Η επιλογή της τιμής ενός επίπεδου για τη δημιουργία του κλειδιού που θα αντιστοιχηθεί στη συγκεκριμένη πλειάδα θα είχε σαν αποτέλεσμα τη δημιουργία ενός συστήματος που απαντά αποδοτικά μόνο τα ερωτήματα που αφορούν τιμές αυτού του επιπέδου, ενώ θα χρειάζονταν η επεξεργασία των ερωτημάτων που αφορούν τα υπόλοιπα επίπεδα από όλους τους κόμβους του δικτύου. Μία εναλλακτική προσέγγιση είναι η δημιουργία διαφορετικών κλειδιών για την κάθε τιμή της πλειάδας και συνεπώς η πολλαπλή εισαγωγή της πλειάδας για κάθε ένα κλειδί. Η προσέγγιση αυτή δεν μπορεί να αποτελέσει μία βιώσιμη λύση: Όσο αυξάνει ο αριθμός των επιπέδων, αυξάνουν και οι απαιτήσεις για τον αποθηκευτικό χώρο που καταλαμβάνεται για τη συγκεκριμένη πλειάδα. Αν και όλα τα ερωτήματα θα μπορούσαν να επιλυθούν χωρίς να απαιτείται η προώθηση τους σε όλους τους κόμβους του δικτύου, η συγκεκριμένη λύση θα αποτύγχανε να ενσωματώσει και να υποστηρίξει τις σχέσεις που εμπεριέχονται στις ιεραρχίες. Η λύση που ακολουθείται στις προτεινόμενες τεχνικές είναι υβριδική: Οι τιμές ενός επιπέδου αναφοράς χρησιμοποιούνται ως τα κλειδιά της

DHT επικάλυψης, ενώ μία προσαρμοστική δομή δεικτοδότησης αναπτύσσεται για τα υπόλοιπα επίπεδα.

Ένα άλλο σημαντικό χαρακτηριστικό που υποστηρίζεται από τις προτεινόμενες τεχνικές είναι η προσαρμοστικότητα του προκύπτοντος συστήματος στο φορτίο ερωτημάτων που του ανατίθεται, δηλαδή η προσαρμογή των δομών δεικτοδότησης για την επίτευξη καλύτερης απόδοσης και την εξισορρόπηση του φόρτου. Η προσαρμοστικότητα του συστήματος εξαρτάται από την ικανότητα του να αποφασίζει σωστά με ένα online και κατανεμημένο τρόπο για την εκτέλεση των λειτουργιών προσαρμογής της δεικτοδότησης. Οι τάσεις στα ερωτήματα καταγράφονται σε κάθε κόμβο ξεχωριστά, έτσι ώστε να εντοπίζονται τα επίπεδα που ζητιούνται περισσότερο για τις διαφορετικές υποστάσεις ιεραρχιών με την ίδια τιμή στο γενικότερο επίπεδο. Η προσαρμογή της δεικτοδότησης των εισαγόμενων πλειάδων ανάλογα με τα πιο δημοφιλή επίπεδα αυξάνει την αποδοτικότητα των μηχανισμών για την επίλυση των ερωτημάτων και συνεισφέρει στην αποτελεσματικότερη κατανάλωση του εύρους ζώνης. Η προσαρμογή της δεικτοδότησης επιτυγχάνεται με τις λειτουργίες *roll-up* προς γενικότερα επίπεδα της ιεραρχίας και *drill-down* προς χαμηλότερα επίπεδα της ιεραρχίας που περιέχουν πληροφορία με μεγαλύτερη λεπτομέρεια.

Οι ανώτερες απαιτήσεις λήφθηκαν υπόψη, ώστε να αναπτυχθούν τεχνικές που μπορούν να χρησιμοποιηθούν σε ένα κατανεμημένο σύστημα που επιτυγχάνει με αποτελεσματικό τρόπο την αποθήκευση, επερώτηση και ενημέρωση δεδομένων που οργανώνονται σε εννοιολογικές ιεραρχίες. Μία DHT επικάλυψη επιλέχθηκε σαν ένα αξιόπιστο υποκείμενο επίπεδο για τη διαχείριση των δεδομένων που αποθηκεύονται, δεικτοδοτούνται και επερωτώνται, ενώ ταυτόχρονα εξαλείφονται πιθανά σημεία συμφόρησης που δημιουργούνται από κεντρικοποιημένες και ιεραρχικές προσεγγίσεις. Οι πάροχοι των δεδομένων εισάγουν μεμονωμένα τα δεδομένα τους στο σύστημα. Τα ερωτήματα συνεχίζουν να επιλύονται, ενώ η επεξεργασία των σταδιακών ενημερώσεων γίνεται με αποδοτικό τρόπο. Η περιγραφόμενη λύση λαμβάνει υπόψη τα επίπεδα που ρωτιούνται και προσαρμόζει τις δομές δεικτοδότησης κατάλληλα ώστε να ευνοηθεί η επίδοση του συστήματος. Επιπλέον, το προτεινόμενο σύστημα διατηρεί τη σημασιολογική πληροφορία που εμπεριέχεται στην ιεραρχία, χρησιμοποιεί "δενδρικές δομές" (tree-like data structure) για την αποθήκευση των δεδομένων και διατηρεί δείκτες προς συσχετιζόμενα κλειδιά. Η εισαγωγή των δεδομένων με τον περιγραφόμενο τρόπο καθιστά δυνατή και την επίλυση ερωτημάτων που αναφέρονται σε συσχετισμούς μεταξύ διαφορετικών επιπέδων της ιεραρχίας όπως: "*Ποια* `sites` *ανήκουν στο* `VO` *'Biomed';*" ή "*Σε ποιο* `category` *ανήκει το* `region` *'AsiaPacific';*".

Η εφαρμογή που μελετάται και χρησιμοποιείται πάνω από το προκύπτον σύστημα είναι ένα κατανεμημένο και πλήρως λειτουργικό *Πληροφοριακό Σύστημα* για υποδομές Πλέγματος (Grid Information System - GIS). Τα υπολογιστικά πλέγματα επιτρέπουν το διαμοιρασμό πόρων με τη συνεργασία των χρηστών που συμμετέχουν σε *εικονικούς οργανισμούς* (*VO*). Ένα VO είναι μια ομάδα χρηστών που προέρχονται από διάφορους οργανισμούς και συνεργάζονται για την επίτευξη ενός συγκεκριμένου σκοπού. Το όραμα που υλοποιείται στα υπολογιστικά πλέγματα είναι

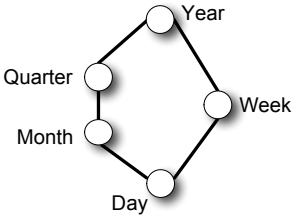
η δημιουργία ενός δικτύου από πόρους που ενεργούν σαν ένας "ενιαίος υπερυπολογιστής", ενώ ταυτόχρονα εξασφαλίζεται η εύκολη πρόσβαση των χρηστών στους πόρους αυτούς. Έχοντας ως στόχο την ανάθεση των εργασιών στους πιο κατάλληλους πόρους για την εκτέλεση τους, το GIS αποθηκεύει όλη την απαιτούμενη πληροφορία για τα χαρακτηριστικά των διαθέσιμων πόρων που παράγεται κατά τη διάρκεια της λειτουργίας μίας υποδομής πλέγματος.

Υπάρχουν διάφορα συστήματα που μπορούν να εκτελέσουν τις εργασίες ενός συστήματος που χρησιμοποιείται σε τέτοιου είδους εφαρμογές, όπως αυτά που παρουσιάζονται στις εργασίες [MDS], [RGM]. Όμως τα συστήματα αυτά είτε χρησιμοποιούν κεντρικές αποθήκες πληροφορίας είτε ιεραρχικές προσεγγίσεις για το συνδυασμό και τη σύνοψη της παραγόμενης πληροφορίας: Οι προσεγγίσεις αυτές δημιουργούν προβλήματα που αφορούν την επεκτασιμότητα και την απόδοση τέτοιου είδους συστημάτων. Επίσης, συστήματα αντίστοιχα με αυτά που συναντώνται στις κατανεμημένες βάσεις δεδομένων μπορούν να χρησιμοποιηθούν για αποθήκευση και επερώτηση τέτοιου είδους πληροφορίας. Ωστόσο, τα συγκεκριμένα συστήματα δεν παρέχουν τη δυνατότητα της συνάθροισης της παραγόμενης πληροφορίας και της συνοπτικής παρουσίασης της σε διαφορετικά επίπεδα λεπτομέρειας. Αυτό όμως αποτελεί σημαντικό παράγοντα για εφαρμογές σε information systems ευρείας κλίμακας, όπου τα ερωτήματα στοχεύουν διαφορετικά επίπεδα λεπτομέρειας των ιεραρχιών: Για παράδειγμα, ερωτήματα που αφορούν παρελθοντική πληροφορία στοχεύουν κυρίως τα γενικότερα επίπεδα των ιεραρχιών (όπως group-by `VO` ή group-by `Year`), ενώ τα online ερωτήματα στοχεύουν επίπεδα που εμπεριέχουν μεγαλύτερη λεπτομέρεια (π.χ. group-by `Site` ή by `Day`).

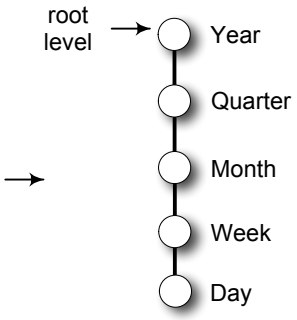
Για να αντιμετωπιστούν τέτοιου είδους ζητήματα, στη συνέχεια προτείνεται μία πλήρως κατανεμημένη και προσαρμοστική αρχιτεκτονική. Για παράδειγμα, έστω ότι θεωρείται ότι η βάση δεδομένων του συστήματος αποθηκεύει πληροφορία που μπορεί να συσχετιστεί με την εννοιολογική ιεραρχία του `VO` (όπως φαίνεται στο Σχήμα 3.2). Αυτή η πληροφορία που προέρχεται από τη συνεχή παρακολούθηση (*monitoring*) της λειτουργίας της υποδομής μπορεί να περιέχει διάφορα γεγονότα (*facts*) σχετικά με την κατάσταση της υποδομής, όπως ο αριθμός των εργασιών που εκτελούνται στην υποδομή, ο αριθμός των εργασιών που αναμένουν την εκτέλεση τους, ο όγκος του διαθέσιμου αποθηκευτικού χώρου, ο συνολικός όγκος του αποθηκευτικού χώρου, κ.ο.κ. Κάποιοι αναμενόμενοι τύποι ερωτημάτων περιλαμβάνουν ερωτήματα όπως "*Ποιος είναι ο μέσος CPU χρόνος*" ή "*Ποιος είναι ο ελάχιστος διαθέσιμος αποθηκευτικός χώρος σε GB*" για κάποια συγκεκριμένη τιμή για το VO επίπεδο.

3.2 Συμβολισμός

Τα δεδομένα προς αποθήκευση εισάγονται στο σύστημα με τη μορφή *πλειάδων* (*tuples*), όπου κάθε πλειάδα περιέχει τις τιμές για κάθε επίπεδο ℓ_i μίας εννοιολογικής ιεραρχίας με L



Σχήμα 3.3: *Μια μερικώς διατεταγμένη εννοιολογική ιεραρχία για το* Time



Σχήμα 3.4: *Μια πλήρως διατεταγμένη εννοιολογική ιεραρχία για το* Time

επίπεδα. Επίσης, κάθε πλειάδα περιέχει και το αριθμητικό fact που περιγράφεται από τις προηγούμενες τιμές (π.χ. `CPU Time, Available Memory`, κτλ.) ή την τοποθεσία που βρίσκονται τα δεδομένα. Το υψηλότερο επίπεδο της ιεραρχίας (ℓ_0) ονομάζεται *root level* και η αντίστοιχη τιμή που προκύπτει *root key*. Επίσης ορίζεται ότι $\ell_a < \ell_b$, όπου $(a, b \in [0, L-1])$, αν και μόνο αν ℓ_a βρίσκεται υψηλότερα στην ιεραρχία (δηλαδή είναι πιο κοντά στο επίπεδο ℓ_0) από το ℓ_b. Οι τιμές των επιπέδων της ιεραρχίας οργανώνονται σε δενδρικές δομές, μία για κάθε root key. Χωρίς απώλεια της γενικότητας, θεωρείται ότι κάθε τιμή στο επίπεδο ℓ_i έχει μόνο ένα πατέρα στο επίπεδο ℓ_{i-1}. Κατά την εισαγωγή μίας πλειάδας επιλέγεται ένα επίπεδο της ιεραρχίας και η τιμή που προκύπτει από την συνάρτηση κατακερματισμού (hash function) χρησιμοποιείται ως το *κλειδί* της πλειάδας στην υποκείμενη DHT επικάλυψη. Η αναφορά στο επίπεδο αυτό γίνεται με τον όρο *pivot level* και η αναφορά στην τιμή του γίνεται ως *pivot key* (ή *pivot value*). Τέλος, το υψηλότερο και το χαμηλότερο pivot level μίας ιεραρχίας ενός συγκεκριμένου root key ονομάζονται *MinPivotLevel* και *MaxPivotLevel* αντιστοίχως.

3.3 Εισαγωγή των Δεδομένων

Το κλειδί (*key*) για κάθε πλειάδα προκύπτει από τη συνάρτηση κατακερματισμού που εφαρμόζεται στην τιμή του επιλεγμένου pivot level. Οι πλειάδες ανατίθενται στους κόμβους με "κοντινότερα" IDs στα παραγόμενα κλειδιά, σύμφωνα με τις τυπικές λειτουργίες των DHT επικαλύψεων.

Στο προτεινόμενο σύστημα, τόσο η αρχική εισαγωγή (insertion) των πλειάδων όσο και οι σταδιακές ενημερώσεις τους (update) αντιμετωπίζονται με έναν ενιαίο τρόπο. Για την εκτέλεση των συγκεκριμένων λειτουργιών υιοθετείται η δημιουργία ενός πλήρως κατανεμημένου καταλόγου που αποθηκεύει όλα τα root keys και τα αντίστοιχα pivot keys τους. Κάθε root key αποθηκεύεται στον κόμβο που είναι "υπεύθυνος" για αυτό μαζί με μία λίστα από τα pivot keys που

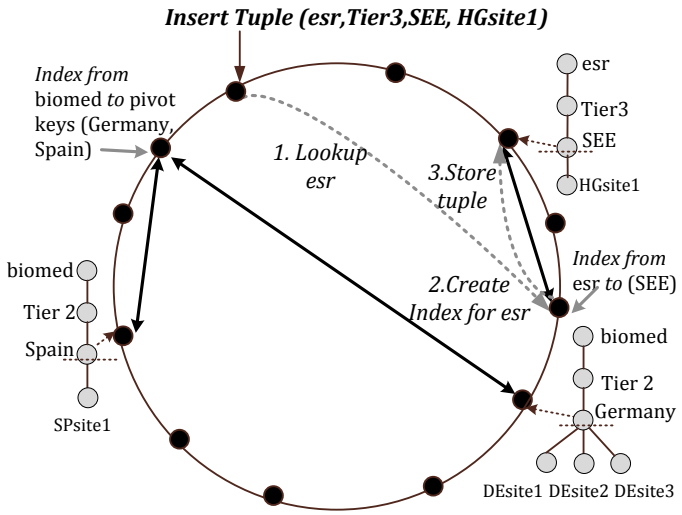
έχουν ήδη εισαχθεί στο σύστημα. Επιπρόσθετα, το root key γνωρίζει και το MaxPivotLevel που χρησιμοποιήθηκε κατά την εισαγωγή των πλειάδων που περιέχουν την τιμή του.

Η διαδικασία που ακολουθείται κατά την εισαγωγή των πλειάδων έχει ως εξής: Αρχικά παράγεται το κλειδί για τη συγκεκριμένη πλειάδα και εκτελείται ένα lookup για το συγκεκριμένο κλειδί στη DHT επικάλυψη. Εάν το υπό εξέταση root key υπάρχει, τότε η πλειάδα καταλήγει στον κόμβο που είναι υπεύθυνος για αυτό. Η διαδικασία που ακολουθείται όταν το root key δεν υπάρχει ήδη στην επικάλυψη θεωρείται ως "*εισαγωγή*". Διαφορετικά, η διαδιακασία της "*ενημέρωσης*" ακολουθείται (βλ. Ενότητα 3.5.1).

Σε περίπτωση που εκτελείται μία εισαγωγή, μία πλειάδα ή μία ομάδα πλειάδων με το ίδιο root key φτάνουν στον κόμβο που είναι υπεύθυνος για αυτό. Ο κόμβος επιλέγει ένα pivot level (είτε με τυχαίο τρόπο είτε σύμφωνα με κάποιον προκαθορισμένο) και υπολογίζεται το pivot key (ή τα pivot keys της ομάδας των πλειάδων) για το pivot level που επιλέχθηκε. Κάθε καινούργιο pivot key προστίθεται στη λίστα με τα pivot keys. Τέλος, κάθε πλειάδα καταλήγει και αποθηκεύεται στον κόμβο που έχει το κοντινότερο ID στο pivot key της.

Κάθε κόμβος οργανώνει τις πλειάδες του σε "δέντρα" που διατηρούν την ιεραρχική δόμηση των τιμών τους. Ως εκ τούτου, κάθε ξεχωριστή τιμή ενός pivot level αντιστοιχεί σε ένα διαφορετικό δέντρο, το οποίο "αποκαλύπτει" και ένα μέρος της συνολικής υπόστασης της ιεραρχίας. Όταν μία νέα πλειάδα φτάνει στον κόμβο που είναι υπεύθυνος για αυτήν, ο κόμβος ψάχνει τα κλειδιά του. Εάν δεν έχει ήδη αποθηκευτεί κάποια πλειάδα για το συγκεκριμένο pivot key, τότε δημιουργείται ένα νέο δέντρο με ένα μόνο κλάδο. Σε αντίθετη περίπτωση, προστίθεται ένας νέος κλάδος που τοποθετείται κάτω από το pivot level με τις νέες τιμές της πλειάδας που δεν υπάρχουν ήδη στο δέντρο.

Ένα παράδειγμα εισαγωγής μίας πλειάδας απεικονίζεται στο Σχήμα 3.5. Μία σύμβαση που ακολουθείται στη γραφική απεικόνιση του συγκεκριμένου παραδείγματος είναι ότι τα συμπαγή βέλη αναπαριστούν δείκτες που έχουν δημιουργηθεί, ενώ τα διακεκομμένα βέλη αντιστοιχούν σε λογικά βήματα που ακολουθούνται κατά την περιγραφόμενη διαδικασία της εισαγωγής της πλειάδας. Έστω ότι θεωρείται ότι οι πλειάδες ακολουθούν την εννοιολογική ιεραρχία του VO που απεικονίζεται στο Σχήμα 3.2 και ότι έχει επιλεχθεί το ℓ_2 (`Region`) ως το γενικό επίπεδο που χρησιμοποιείται κατά τις αρχικές εισαγωγές. Η νέα πλειάδα που εισάγεται στο σύστημα έχει το root key *'esr'* . Εφόσον το συγκεκριμένο root key δεν υπάρχει ήδη, ένας νέος δείκτης (*index*) δημιουργείται στον κόμβο που είναι υπεύθυνος για αυτό το root key. Στη συνέχεια, το επίπεδο που αντιστοιχεί στο `Region` επιλέγεται ως pivot level και η πλειάδα "προωθείται" στον κόμβο που είναι υπεύθυνος για το pivot key της. Ο κόμβος αυτός δημιουργεί ένα νέο δέντρο το οποίο σε αυτή τη φάση αποτελείται από μόνο έναν κλάδο.



Σχήμα 3.5: *Παράδειγμα εισαγωγής μίας καινούργιας πλειάδας, ενώ δεν υπάρχει ήδη κάποια άλλη πλειάδα με το ίδιο root key στην επικάλυψη*

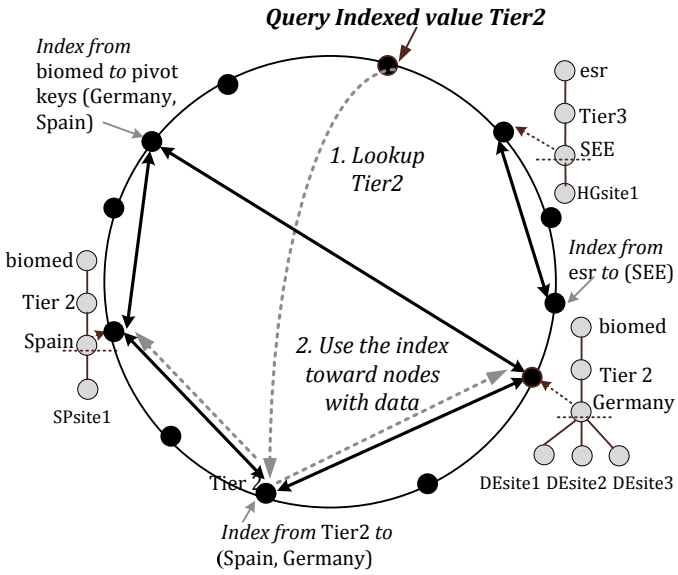
3.4 Αναζήτηση των Δεδομένων

Τα ερωτήματα που αφορούν το pivot level αναφέρονται ως "*exact match*" ερωτήματα και μπορούν να απαντηθούν απευθείας από το μηχανισμό της DHT επικάλυψης που υλοποιεί τη λειτουργία "*lookup*". Τα ερωτήματα που αφορούν τις τιμές των υπόλοιπων επιπέδων δεν μπορούν να επιλυθούν, εκτός και αν "*πλημμυρίσουν*" την επικάλυψη και καταλήξουν σε όλους τους κόμβους της. Θέτοντας ως στόχο την αξιοποίηση της γνώσης που αποκτιέται κατά τη διάρκεια της "πλημμύρας" ενός ερωτήματος (query flooding), προτείνεται η δημιουργία *προσωρινών δεικτών διπλής κατεύθυνσης* (*soft-state bidirectional indices*). Όταν ένας κόμβος απαντάει ένα ερώτημα που έχει πλημμυριστεί, ελέγχει επίσης εάν είναι απαραίτητο να εκτελεστεί κάποια από τις λειτουργίες roll-up ή drill-down. Εάν δεν απαιτείται η εκτέλεση κάποιας τέτοιας λειτουργίας, ο κόμβος που έθεσε το ερώτημα (ο κόμβος αυτός αναφέρεται ως query initiator) ξεκινάει τη διαδικασία της δημιουργίας ενός δείκτη, μόλις λάβει την ολοκληρωμένη απάντηση. Για αυτό το λόγο, ο κόμβος query initiator εισάγει το κλειδί που αντιστοιχεί στην τιμή που ρωτήθηκε μαζί με τα αναγνωριστικά των κόμβων που αποθηκεύουν τις σχετικές πλειάδες στη DHT επικάλυψη. Επίσης, οι κόμβοι που αποθηκεύουν τις πλειάδες αυτές επισημαίνουν τη συγκεκριμένη τιμή ως *δεικτοδοτημένη* (*indexed*). Την επόμενη φορά που θα εκτελεστεί ένα ερώτημα για αυτό το κλειδί, η βασική lookup λειτουργία του DHT εντοπίζει τον κόμβο που είναι υπεύθυνος για αυτό και συνεπώς το δείκτη που έχει δημιουργηθεί και περιέχει την πληροφορία για τους κόμβους με σχετικές πλειάδες που πρέπει να ανακτηθούν.

Οι δείκτες που δημιουργούνται είναι soft-state έτσι ώστε να μειωθεί η διατήρηση πλεονάζουσας πληροφορίας. Αυτή η παραδοχή έχει ως αποτέλεσμα τη λήξη των δεικτών μετά το τέλος μίας προκαθορισμένης χρονικής περιόδου (Time-to-Live ή *TTL*). Κάθε φορά που χρησιμοποιείται ένας υπάρχον δείκτης, το TTL του ανανεώνεται. Ο περιορισμός αυτός εξασφαλίζει ότι οι αλλαγές στο σύστημα (π.χ. αλλαγές στις τοποθεσίες των δεδομένων, οι αφίξεις ή αναχωρήσεις κόμβων, κλπ.) δεν θα έχουν ως συνέπεια την ύπαρξη "λανθασμένων" δεικτών (stale indices), οι οποίοι θα έχουν αρνητική επίπτωση στην αποτελεσματικότητα του lookup μηχανισμού. Επιπλέον, όταν ο αριθμός των δεικτών φτάνει ένα όριο I_{max}, τότε η δημιουργία ενός καινούργιου δείκτη προκαλεί τη διαγραφή ενός ή περισσοτέρων παλαιότερων δεικτών. Γενικώς, το σύστημα τείνει να διατηρεί του πιο "χρήσιμους" δείκτες, δηλαδή αυτούς που αναφέρονται σε δεδομένα που αναζητούνται συχνότερα.

Οι κόμβοι με τις πλειάδες που εμπεριέχουν μία δεικτοδοτημένη τιμή πρέπει να γνωρίζουν την ύπαρξη ενός δείκτη, ώστε να τον διαγράψουν μετά από μία λειτουργία προσαρμογής της δεικτοδότησης (re-indexng operation). Η διπλή κατεύθυνση (bidirectionallity) των δεικτών υιοθετείται μόνο για την εξασφάλιση της συνοχής των δεδομένων (data consistency), παρά το γεγονός ότι οι δείκτες είναι προσωρινοί. Κατά τη διάρκεια μίας λειτουργίας προσαρμογής της δεικτοδότησης, οι τοποθεσίες των αποθηκευμένων πλειάδων αλλάζουν και οι δείκτες που συσχετίζονται με αυτές τις πλειάδες πρέπει είτε να ενημερωθούν είτε να διαγραφούν, ώστε με αυτόν τον τρόπο να αποτραπεί η δημιουργία "λανθασμένων" δεικτών (stale indices). Η επιλογή που ακολουθείται είναι η πλήρης διαγραφή των προσωρινών δεικτών, ώστε να αποφευχθεί η αύξηση της πολυπλοκότητας των μηχανισμών του συστήματος. Επιπλέον, η ύπαρξη λεπτομερούς πληροφορίας για έναν υπάρχοντα δείκτη δεν είναι καθοριστικής σημασίας για τον κόμβο που αποθηκεύει τις πλειάδες της δεικτοδοτημένης τιμής. Αρκεί μία απλή σήμανση των τιμών που είναι δεικτοδοτημένες έτσι ώστε να διαγραφεί ο δείκτης αν χρειαστεί. Όμως σε αυτήν την περίπτωση μπορεί να γίνουν κάποιες πλεονάζουσες ενέργειες για τη διαγραφή μη έγκυρων δεικτών που έχουν ήδη λήξει. Αν όμως δεν υπάρχουν περιορισμοί στο διαθέσιμο αποθηκευτικό χώρο και η επεξεργασία στο επίπεδο του κόμβου είναι προτιμότερη σε σύγκριση με την επιβάρυνση της κατανάλωσης εύρους ζώνης (bandwidth consumption), τότε μπορεί να καταγράφεται και ένα timestamp για τις δεικτοδοτημένες τιμές. Σε αυτήν την περίπτωση, κάθε lookup για μία δεικτοδοτημένη τιμή ανανεώνει το TTL και στις δύο πλευρές του δείκτη και μόνο οι έγκυροι δείκτες διαγράφονται κατά την εκτέλεση μίας λειτουργίας re-indexing.

Στο παράδειγμα του Σχήματος 3.6 ένα ερώτημα για την τιμή *'SEE'* επιλύεται απευθείας με τη χρήση της lookup λειτουργίας του DHT πρωτοκόλλου. Κατά την αναζήτηση τιμών που ανήκουν στο root level χρησιμοποιούνται οι δείκτες που δημιουργήθηκαν κατά την εισαγωγή των πλειάδων. Ωστόσο, κάθε lookup που αφορά οποιοδήποτε άλλο επίπεδο εκτός από το pivot level ή το root level δεν επιστρέφει κάποιο αποτέλεσμα. Για παράδειγμα, ένα ερώτημα για τα δεδομένα που περιγράφονται από την τιμή *'Tier2'* δεν καταλήγει στους δύο κόμβους με τα αντίστοιχα



Σχήμα 3.6: *Παράδειγμα αναζήτησης με χρήση δεικτών για μία τιμή που δεν ανήκει στο επιλεγμένο pivot level*

δέντρα, όταν εκτελείται μία lookup λειτουργία και δεν έχει δημιουργηθεί ήδη ένας δείκτης για την τιμή αυτήν. Στην επόμενη φάση, το ερώτημα αυτό πλημμυρίζεται στο DHT και επομένως φθάνει στους κόμβους που είναι υπεύθυνοι για τα κλειδιά *'Spain'* και *'Germany'*. Ο κόμβος που έθεσε το ερώτημα είναι πλέον ενήμερος για τα υπάρχοντα pivot keys και δημιουργεί δείκτες που αποθηκεύουν την πληροφορία για αυτά τα pivot keys στον κόμβο που είναι υπεύθυνος για το *'Tier2'*. Ο συγκεκριμένος κόμβος διατηρεί πλέον ένα δείκτη που δείχνει στον κόμβο *'Spain'* και έναν άλλο που δείχνει στον κόμβο *'Germany'*. Στο μέλλον, τα ερωτήματα που αφορούν το *'Tier2'* θα επιλύονται χωρίς πλημμύρα λόγω της χρήσης των προσωρινών δεικτών που έχουν δημιουργηθεί. Όπως φαίνεται αναλυτικά στο Σχήμα 3.6, ένα ερώτημα για την τιμή *'Tier2'* του `Category` που εκτελείται μετά τη δημιουργία των αντίστοιχων δεικτών φτάνει στον κόμβο που είναι υπεύθυνος για το κλειδί του, ο οποίος με τη σειρά του προωθεί το ερώτημα σε όλους τους σχετικούς κόμβους. Οι κόμβοι που αποθηκεύουν τα δέντρα με την τιμή αυτή επιστρέφουν μόνο σχετικές πλειάδες.

Εφόσον τα root indices δημιουργούνται κατά την εισαγωγή των πλειάδων, οι δείκτες αυτοί μπορούν να χρησιμοποιηθούν για να βελτιστοποιηθεί η διαδικασία της πλημμύρας και να μειωθεί ο αριθμός των μηνυμάτων και η κατανάλωση του εύρους ζώνης. Όταν ένα ερώτημα επιλύεται με πλημμύρα, το ερώτημα αυτό προωθείται από έναν κόμβο στον κοντινότερο γείτονα του στην επικάλυψη και καταγράφεται το εύρος των αναγνωριστικών του επισκεπτόμενου κόμβου (αναφέρεται ως *CoveredIdRange*). Σύμφωνα με την περιγραφόμενη προσέγγιση για την αποθήκευση

των δεδομένων, εάν ένα ερώτημα αφορά μία τιμή που ανήκει σε ένα επίπεδο που βρίσκεται κάτω από το pivot level του αντίστοιχου δέντρου, τότε οι αντίστοιχες πλειάδες που περιέχουν τη συγκεκριμένη τιμή μπορούν να ανακτηθούν μόνο από έναν κόμβο. Συνεπώς, η προώθηση του πλημμυρισμένου ερωτήματος ολοκληρώνεται μόλις βρεθεί ο κόμβος με την ερωτηθείσα τιμή. Όσον αφορά την περίπτωση ενός ερωτήματος για κάποια τιμή που βρίσκεται πάνω από το pivot level, η προώθηση του ερωτήματος δε μπορεί να σταματήσει μόλις βρεθεί ο πρώτος κόμβος που είναι υπεύθυνος για κάποιο δέντρο που περιέχει την τιμή αυτή. Παρόλα αυτά, ο κόμβος μπορεί να απαντήσει με τα σχετικά tuples στον κόμβο που έθεσε το ερώτημα και να προωθήσει το ερώτημα στον κόμβο που είναι υπεύθυνος για το root key. Μόλις γίνει η παραλαβή του ερωτήματος από αυτόν τον κόμβο, εξετάζεται το CoveredIdRange όλων των κόμβων που έχει επισκεφτεί το ερώτημα. Στη συνέχεια το ερώτημα προωθείται παράλληλα σε όλους τους κόμβους που αποθηκεύουν τα pivot keys του δείκτη που δεν περιλαμβάνονται στο CoveredIdRange. Σύμφωνα με αυτή τη στρατηγική, η πληροφορία που αποθηκεύεται μαζί με το root key χρησιμοποιείται και μπορεί να αποφευχθεί η επίσκεψη όλων των κόμβων του συστήματος κατά τη διαδικασία της πλημμύρας. Για παράδειγμα, έστω ότι θεωρείται η οργάνωση των αποθηκευμένων πλειάδων όπως φαίνεται στο Σχήμα 3.5. Αρχικά ένα ερώτημα για το *'Tier2'* πλημμυρίζεται στο δίκτυο και η προώθηση του ξεκινάει από έναν κόμβο στον ακόλουθο του με δεξιόστροφο τρόπο. Σύμφωνα με αυτό το σενάριο, το ερώτημα φτάνει πρώτα στον κόμβο με το pivot key *'Germany'*. Ο συγκεκριμένος κόμβος προωθεί το ερώτημα στον κόμβο που είναι υπεύθυνος για το root key όπως προδιαγράφει η περιγραφόμενη στρατηγική. Τελικά, ο κόμβος με το root index στέλνει το ερώτημα που πλημμυρίζεται μόνο στον κόμβο που είναι υπεύθυνος για το pivot key *'Spain'*, του οποίου το δέντρο δεν έχει εξεταστεί.

3.5 Προσαρμοστική Δεικτοδότηση

Ένας σημαντικός στόχος που επιτυγχάνεται στις περιγραφόμενες τεχνικές είναι η δυναμική προσαρμογή του συστήματος στα online ερωτήματα "στο επίπεδο του κόμβου" (on a per node basis), έτσι ώστε να αυξηθεί η αναλογία (ratio) των ερωτημάτων που δεν πλημμυρίζονται (non-flooded ερωτήματα). Για την επίτευξη αυτού του στόχου, προτείνονται δύο λειτουργίες που σχετίζονται με την επιλογή του pivot level: *Roll-up* προς τα γενικότερα επίπεδα της εννοιολογικής ιεραρχίας και *drill-down* προς επίπεδα χαμηλότερα από το pivot level.

Η ιδέα για την εκτέλεση των re-indexing λειτουργιών για τις αποθηκευμένες πλειάδες βασίζεται στο γεγονός ότι ο κάθε κόμβος έχει μια καθολική όψη όλων των ερωτημάτων που κατευθύνονται προς τις τιμές που βρίσκονται πάνω από το pivot level, αλλά μία ειδικότερη όψη για τα ερωτήματα που κατευθύνονται μόνο προς τις τιμές του για τα επίπεδα κάτω από το pivot level. Για αυτό το λόγο, ένας κόμβος έχει αρκετή πληροφορία για να αποφασίσει εάν ένα drill-down θα ευνοήσει την αύξηση των exact match ερωτημάτων για τις τιμές του. Από την άλλη πλευρά,

ένας κόμβος πρέπει να συνεργαστεί και με τους υπόλοιπους κόμβους που αποθηκεύουν μία τιμή για ένα επίπεδο $\ell_i < pivotlevel$ ώστε να αποφασίσει αν αυτό το επίπεδο είναι πιο κατάλληλο.

Το re-indexing των πλειάδων (μέσω της επιλογής ενός διαφορετικού pivot level) εκτελείται στο "επίπεδο του δέντρου" (on a per-tree basis), χωρίς να απαιτείται καθολικός συντονισμός όλων των κόμβων. Κάθε κόμβος συγκεντρώνει πληροφορία βάσει των εισερχόμενων ερωτημάτων (incoming queries) και με αυτόν τον τρόπο ανακαλύπτει αν το επιλεγμένο pivot level ενός δέντρου παραμένει το πιο δημοφιλές του επίπεδο. Η "δημοτικότητα" (popularity) των επιπέδων των δέντρων εκτιμάται σύμφωνα με τους μέσους ρυθμούς των εισερχόμενων ερωτημάτων (αναφέρεται ως *InQ*) για μία χρονική περίοδο. Κάθε κόμβος διατηρεί μία εγγραφή (record) ανά δέντρο για πληροφορία που δημιουργείται σε ένα περιορισμένο χρονικό διάστημα W. Η παράμετρος αυτή πρέπει να επιλεχθεί κατάλληλα ώστε να γίνονται αντιληπτές οι μεταβολές των κατανομών των φορτίων, ενώ συγχρόνως να μην επηρεάζεται η λήψη των αποφάσεων από στιγμιαία και παραπλανητικά φαινόμενα.

Αναλυτικότερα, ο συγκεκριμένος μηχανισμός λειτουργεί ως ακολούθως: Ένας κόμβος μπορεί να ελέγξει εάν απαιτείται μία re-indexing λειτουργία σύμφωνα με τον στόχο που έχει τεθεί προς επίτευξη. Η στρατηγική, που επιλέχθηκε να υλοποιηθεί, συνεπάγεται ότι ένας κόμβος αποφασίζει εάν ένα roll-up ή drill-down είναι αναγκαίο όταν απαντήσει ένα flooded ερώτημα ή όταν έχει ληφθεί ένας αριθμός από ερωτήματα για τιμές που έχουν δεικτοδοτηθεί (indexed τιμές). Έτσι, ενώ ο βασικός στόχος είναι η αύξηση των ερωτημάτων που επιλύονται χωρίς flooding, η στρατηγική αυτή στοχεύει επίσης και στην αύξηση των exact match ερωτημάτων.

Ο αριθμός των ερωτημάτων για indexed τιμές που ενεργοποιεί τη διαδικασία για την εξέταση ενός πιθανού re-indexing σε κάποιον κόμβο μπορεί να διαφέρει και η επιλογή του αριθμού αυτού έχει επίπτωση στην ικανότητα προσαρμογής του συστήματος. Η επιλογή μίας μικρής τιμής συνεπάγεται ότι το ενδεχόμενο ενός re-indexing εξετάζεται συχνότερα και για αυτό το λόγο περισσότερες λειτουργίες re-indexing είναι πιθανό να εκτελεστούν. Ωστόσο, εάν μία απόφαση έχει ληφθεί εσφαλμένα, μπορεί να διορθωθεί ευκολότερα. Πρέπει όμως να ληφθεί υπόψη, ότι κατά τη διάρκεια των re-indexing λειτουργιών, οι υπάρχοντες δείκτες διαγράφονται και αυτό μπορεί να έχει αρνητική επίπτωση στην αποδοτικότητα του συστήματος. Σε περίπτωση επιλογής μίας μεγάλης τιμής, το σύστημα τείνει να εξαρτάται περισσότερο από την αποτελεσματικότητα των δεικτών. Κατά τη διάρκεια της λειτουργίας του συστήματος, παρατηρήθηκε ότι οι re-indexing λειτουργίες είναι απαραίτητες, όταν οι πιο δημοφιλείς τιμές ανήκουν σε επίπεδα με περισσότερες διακριτές τιμές. Η δημιουργία δεικτών ενδείκνυται περισσότερο για τις τιμές των υψηλότερων επιπέδων που έχουν λιγότερες τιμές, δεδομένου ότι για αυτές τις τιμές αυξάνεται η πιθανότητα για επαναλαμβανόμενη χρησιμοποίηση ενός δείκτη.

Ένας κόμβος αποφασίζει εάν ένα re-indexing θα ευνοήσει την αύξηση των non-flooded ερωτημάτων σύμφωνα με το '*popularity*' του κάθε επιπέδου και ακολουθώντας τη διαδικασία με τα βασικά βήματα που περιγράφονται στον Αλγόριθμο 3. Στη λήψη της απόφασης χρησιμοποιείται

Algorithm 3 Decision Algorithm in the node answering a query

pivotlevel: current pivot level
ℓ_q: the queried level of the flooded or indexed value
NotPivotKey: the flooded or indexed value
InQ_{tot}: rate of incoming queries for the tree with *NotPivotKey*
InQ_{ini}: initial minimum rate to allow re-index operations
$InQ_{l_{pop}}$: rate of incoming queries for the most popular level
action: the decided action
$\ell_{pop} \Leftarrow FindMostPopularLevel$
if $\ell_q > pivotlevel$ **then**
  if $(InQ_{tot} > InQ_{ini})$ AND $(\ell_{pop} > pivotlevel)$ AND $(InQ_{\ell_{pop}} > thr \times InQ_{tot})$ **then**
    Drill-down to l_{pop}
    $action \Leftarrow NoAction$
  else if *NotPivotKey* is NOT indexed **then**
    $action \Leftarrow CreateIndex$
  else
    $action \Leftarrow NoAction$
  end if
else if $\ell_q < pivotlevel$ **then**
  if $(InQ_{tot} > InQ_{ini})$ AND $(\ell_q = \ell_{pop})$ AND
  $(InQ_{\ell_q} > thr \times InQ_{tot})$ **then**
    $action \Leftarrow PositiveToRollup$
  else if *NotPivotKey* is NOT indexed **then**
    $action \Leftarrow CreateIndex$
  else
    $action \Leftarrow NoAction$
  end if
end if

η παράμετρος *thr* για να υποδείξει εάν απαιτείται ένα re-indexing. Το ακόλουθο κριτήριο ορίζει αν επιτρέπεται ένα re-indexing σε ένα επίπεδο ℓ_q:

$$InQ_{\ell_q} > thr \times \sum_{i=0}^{i=L-1} InQ_{\ell_i}, \ell_q \neq pivotlevel, thr \in [0,1]$$

Στον περιγραφόμενο αλγόριθμο θεωρούνται δύο πιθανές περιπτώσεις που υποδεικνύουν την αναγκαιότητα μίας re-indexing λειτουργίας:

Το επίπεδο που ρωτήθηκε (queried level) ℓ_q βρίσκεται χαμηλότερα στην ιεραρχία από το pivot level του δέντρου ($\ell_q > pivotlevel$). Σε αυτήν την περίπτωση, μόνο ένα δέντρο αποθηκεύει τις τιμές ενός επιπέδου που βρίσκεται κάτω από το pivot level. Ως εκ τούτο, ένας συγκεκριμένος κόμβος έχει επίγνωση για το ακριβές popularity αυτών των τιμών και θεωρεί τον εαυτό του

"ικανό" να αποφασίσει αν χρειάζεται ένα drill-down. Εάν το πιο δημοφιλές επίπεδο ℓ_{pop} βρίσκεται κάτω από pivot level και το αναφερόμενο κριτήριο ισχύει για το InQ του, τότε αποφασίζεται ένα drill-down. Αφού ληφθεί η απόφαση για το drill-down, ο κόμβος βρίσκει όλες τις διακριτές τιμές του καινούργιου pivot level, δημιουργεί τα κλειδιά για κάθε μια από αυτές και στέλνει τις νέες ομάδες των πλειάδων στους αντίστοιχους κόμβους. Η στατιστική πληροφορία που έχει ήδη συλλεχθεί αποστέλλεται μαζί με μία ομάδα πλειάδων που επιλέγεται τυχαία, έτσι ώστε να διατηρηθεί η πληροφορία για την κατανομή των ερωτημάτων που σχετίζονται με τις τιμές που περιέχονται στο δέντρο που εκτελεί drill-down για το χρονικό διάστημα W. Κάθε δείκτης που έχει δημιουργηθεί για οποιαδήποτε τιμή αυτού του δέντρου διαγράφεται. Εάν δε απαιτείται μια drill-down λειτουργία, τότε ο κόμβος επισημαίνει στην απάντηση προς τον κόμβο που έθεσε το ερώτημα ότι το επίπεδο που ρωτήθηκε βρίσκεται κάτω από το pivot level και συνεπώς ότι μπορεί να συνεχιστεί άμεσα η διαδικασία για τη δημιουργία του soft-state δείκτη.

Το επίπεδο που ρωτήθηκε (queried level) ℓ_q βρίσκεται υψηλότερα στην ιεραρχία από το pivot level του δέντρου ($\ell_q < pivotlevel$). Όσον αφορά αυτή τη περίπτωση πρέπει να ληφθεί υπόψη ότι υπάρχουν περισσότερα από ένα δέντρα με την τιμή που περιλαμβάνεται στο ερώτημα και τα δέντρα αυτά πρέπει να συμμετέχουν σε ένα πιθανό roll-up προς αυτό το επίπεδο. Διαφορετικά, η αναζήτηση αυτής της τιμής δε θα επιστρέψει όλα τα αποτελέσματα. Εάν το threshold κριτήριο ικανοποιείται για το ℓ_q, τότε ο κόμβος που το έλεγξε είναι θετικός στο ενδεχόμενο της υιοθέτησης αυτού του επιπέδου ως pivot level για το συγκεκριμένο δέντρο. Μέχρι τώρα, το βήμα αυτό είναι αναγνωριστικό για την πιθανή ύπαρξη κάποιας ανισορροπίας και ο κόμβος που έθεσε το ερώτημα ενημερώνεται για αυτό το ενδεχόμενο. Ο κόμβος που έθεσε το ερώτημα αποφασίζει για την εκτέλεση ενός re-indexing σύμφωνα με τη διαδικασία που περιγράφεται στον Αλγόριθμο 4. Ειδικότερα, αν ο κόμβος αυτός γνωρίζει τουλάχιστον έναν κόμβο που είναι πρόθυμος να εκτελέσει ένα roll-up προς αυτό το επίπεδο, ξεκινάει τη διαδικασία για να επιβεβαιώσει τη μερική αυτή ένδειξη με τη χρήση στατικής πληροφορίας που παρέχεται από όλους τους κόμβους που απάντησαν στο ερώτημα. Μετά τη συγκέντρωση όλων των εγγραφών που περιέχουν την πληροφορία σχετικά με το InQ για κάθε επίπεδο των σχετικών δέντρων, ο κόμβος αυτός υπολογίζει τη συνολική τιμή του InQ για κάθε επίπεδο.

Ο υπολογισμός του συνολικού InQ για κάθε επίπεδο είναι σύνθετος. Τα ερωτήματα που αφορούν ένα επίπεδο ℓ_i για $\ell_i \geq pivotlevel$ καταλήγουν μόνο σε ένα κόμβο και για αυτό το λόγο υπολογίζονται μόνο μία φορά για στατιστικούς λόγους. Η ίδια ιδιότητα δεν ισχύει για τα ερωτήματα που αφορούν τιμές επιπέδων που βρίσκονται υψηλότερα από το pivot level. Η επεξεργασία αυτών των ερωτημάτων μπορεί να γίνει σε περισσότερους από έναν κόμβους και η καταμέτρηση τους γίνεται σε όλους αυτούς. Κατά τη συλλογή της στατιστικής πληροφορίας που χρησιμοποιείται στις αποφάσεις για roll-up, το πρόβλημα της πολλαπλής καταμέτρησης τέτοιων ερωτημάτων για τον υπολογισμό του πραγματικού InQ για κάθε επίπεδο πρέπει να αντιμετωπιστεί. Η πολυπλοκότητα του υπολογισμού του συνολικού InQ αυξάνει, δεδομένου ότι περισσότερα από ένα

Algorithm 4 Decision Algorithm in the querying node

l_q: the queried level of the flooded or indexed value
NotPivotKey: the flooded or indexed value
action: the required action by involved nodes {*action* = *PositiveToRollup* if at least one node is possitive to roll-up}
if *action* = *PositiveToRollup* **then**
  Gather statistic information
  Calculate *InQ* for each level
  $\ell_{pop} \Leftarrow FindMostPopularLevel$
  $MaxPivotLevel \Leftarrow FindMaxPivotLevel$
  if $(\ell_q = \ell_{pop})$ AND $(InQ_{\ell_{pop}} > thr \times InQ_{tot})$ **then**
    Roll-up to l_{pop}
  else if $(\ell_{pop} \geq MaxPivotLevel)$ AND $(InQ_{\ell_{pop}} > thr \times InQ_{tot})$ **then**
    Group-Drill-down to ℓ_{pop}
  else if *NotPivotKey* is NOT indexed **then**
    Create Index for *NotPivotKey*
  end if
else if *action* = *CreateIndex* **then**
  Create Index for *NotPivotKey*
end if

pivot level μπορεί να υπάρχουν για τα εμπλεκόμενα δέντρα στη διαδικασία του re-indexing. Για παράδειγμα, έστω ότι θεωρείται η μορφή των δέντρων με την τιμή *'biomed'* στο root level που φαίνεται στο Σχήμα 3.7(b). Στην περίπτωση αυτήν, η τιμή του InQ για την τιμή *'Tier2'* στέλνεται δύο φορές από τους κόμβους που αποθηκεύουν τα δέντρα με την τιμή *'Tier2'*. Για την αποφυγή αυτού το συμβάντος, το μονοπάτι (path) που περιλαμβάνει τις τιμές για όλα τα επίπεδα στο $[0, pivotlevel]$ συμπεριλαμβάνεται μαζί με την αποστολή της στατιστικής πληροφορίας στον κόμβο που έθεσε το ερώτημα, έτσι ώστε να λάβει τη σωστή απόφαση. Με αυτήν την τακτική, ο κόμβος που θέτει το ερώτημα ενημερώνεται για το MinPivotLevel και το MaxPivotLevel των υπαρχόντων δέντρων με την τιμή που ρωτήθηκε (που από εδώ και στο εξής αναφέρεται ως *NotPivotKey*).

Εάν το InQ για το επίπεδο ℓ_q είναι μεγαλύτερο κατά *thr* φορές από το συνολικό αριθμό του InQ, τότε ο κόμβος που έθεσε το ερώτημα ειδοποιεί τους εμπλεκόμενους κόμβους να εκτελέσουν roll-up στα σχετικά δέντρα τους προς αυτό το επίπεδο και να εισάγουν ξανά τις πλειάδες τους. Εάν το κριτήριο για την εκτέλεση του re-indexing για το ℓ_q δεν ικανοποιείται και δεδομένου ότι έχει υπολογιστεί η στατιστική πληροφορία, ο κόμβος που έθεσε το ερώτημα εξετάζει αν απαιτείται ένα drill-down σε ένα επίπεδο $\ell_i \geq MaxPivotLevel$ (η ισότητα αναφέρεται στην περίπτωση που όλα τα εμπλεκόμενα δέντρα δεν έχουν το ίδιο pivot level αλλά κάποια από αυτά έχουν ήδη μεταβεί στο MaxPivotLevel) σύμφωνα με τα συγκεντρωμένα στατιστικά. Η προσέγγιση αυτή στοχεύει στην αξιοποίηση του γεγονότος ότι ο κόμβος που έχει συγκεντρώσει τα

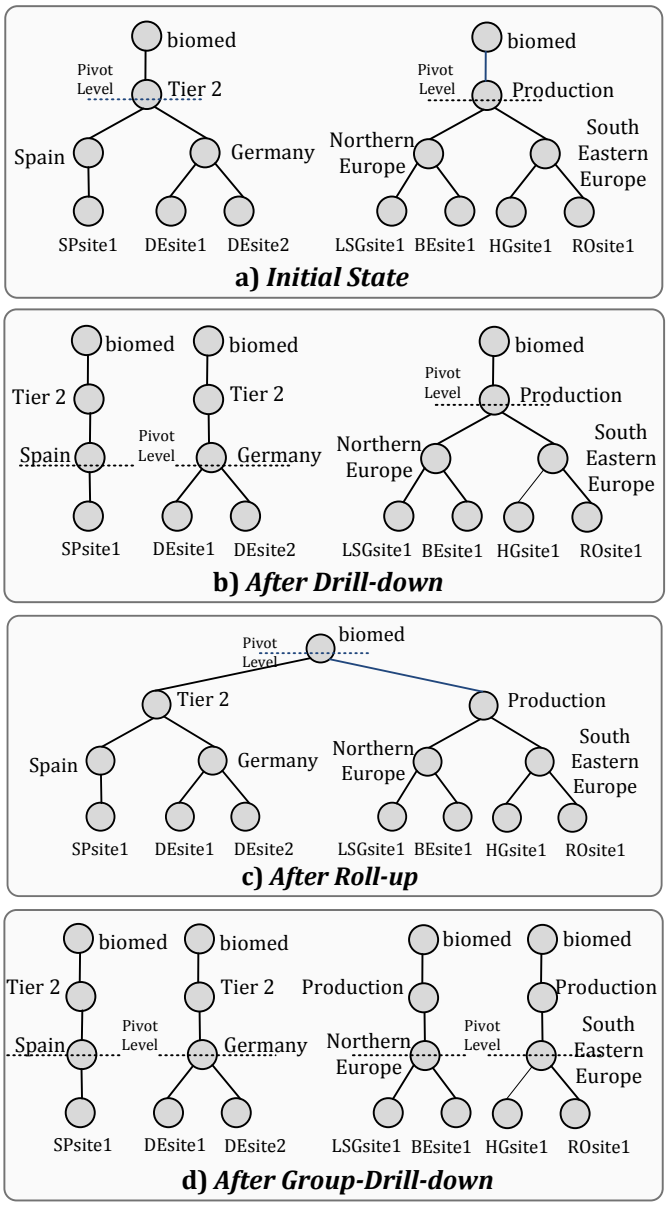
στατιστικά έχει πιο ολοκληρωμένη άποψη για το InQ του κάθε επιπέδου. Επομένως, είναι πιθανό να ανακαλύψει ότι ένα επίπεδο $\ell_i \geq MaxPivotLevel$ είναι πιο δημοφιλές αλλά αυτή η τάση να μην έχει εμφανιστεί στη μερική εικόνα που έχει ένας κόμβος για το δέντρο του. Εάν κάτι τέτοιο ισχύει, τότε ο κόμβος που συγκέντρωσε τα στατιστικά ενημερώνει τους εμπλεκόμενους κόμβους ότι απαιτείται ένα drill-down προς αυτό το επίπεδο. Η λειτουργία αυτή ονομάζεται `Group-Drill-down` και συμμετέχουν περισσότεροι από ένας κόμβοι. Όλα τα δέντρα με την τιμή που ρωτήθηκε στο ℓ_q εκτελούν drill-down στο καινούργιο pivot level. Εάν το καινούργιο pivot level είναι ίσο με το MaxPivotLevel, τότε τα δέντρα που χρησιμοποιούν ήδη το MaxPivotLevel ως pivot level δεν προχωρούν σε κάποια περαιτέρω ενέργεια. Σε περίπτωση που η στατιστική πληροφορία δεν υποδεικνύει ότι πρέπει να γίνει κάποιο re-indexing, η περιγραφόμενη διαδικασία τερματίζει με τη δημιουργία του soft-state δείκτη για αυτή την τιμή.

Κατά τη διάρκεια της λήψης μίας απόφασης και της εκτέλεσης μίας re-indexing λειτουργίας ενεργοποιούνται μηχανισμοί κλειδώματος (lock mechanisms). Ο λόγος για το locking είναι η αποφυγή της διερεύνησης του ενδεχομένου για re-indexing πολλαπλές φορές κατά τη διάρκεια ταυτόχρονων αναζητήσεων στα ίδια δέντρα. Τα locks απενεργοποιούνται μετά την ολοκλήρωση της διαδικασίας ή μετά τη λήξη μίας μικρής χρονικής περιόδου. Τα βήματα των αλγορίθμων 3 και 4 ξεκινάνε αν δεν υπάρχουν ενεργοποιημένα locks, τα οποία ενεργοποιούνται στη συνέχεια για τα εμπλεκόμενα δέντρα. Στην αντίθετη περίπτωση δεν ξεκινάει η διαδικασία λήψης απόφασης για re-indexing και απαντιέται μόνο η ερώτηση.

Ένα παράδειγμα των αποτελεσμάτων των περιγραφόμενων re-indexing λειτουργιών φαίνεται στο Σχήμα 3.7 και αφορά τα δέντρα με την τιμή *'biomed'* στο root level. Τα δύο δέντρα του Σχήματος Figure3.7(a) αποθηκεύονται σε διαφορετικούς κόμβους της DHT επικάλυψης και θεωρούνται ως η αρχική κατάσταση πριν την έναρξη ενός re-indexing. Αρχικά, ας υποθέσουμε ότι ένα ερώτημα για το *'Spain'* είναι η αφορμή για να ξεκινήσει ένα drill-down. Το αποτέλεσμα του drill-down φαίνεται στο Σχήμα Figure 3.7(b). Ένα ερώτημα για το *'biomed'* μπορεί να έχει ως αποτέλεσμα ένα roll-up προς το root level και να σχηματιστεί το δέντρο που φαίνεται στο Σχήμα 3.7(c) ή ένα Group-Drill-down στο επίπεδο του `Region`. Ένα Group-Drill-down θα έχει σαν αποτέλεσμα τα δέντρα που απεικονίζονται στο Σχήμα 3.7(d) και διαφέρει από το απλό drill-down, αφού όλα τα δέντρα εκτελούν drill-down στο επίπεδο ℓ_2.

3.5.1 Ενημερώσεις

Μία *ενημέρωση* (update) καλείται η διαδικασία που ακολουθείται κατά την εισαγωγή μίας πλειάδας, όταν το root keys της είναι ήδη αποθηκευμένο στην επικάλυψη. Η διαδικασία της ενημέρωσης αποτελείται από δύο διαδοχικές φάσεις: αρχικά εισάγεται η πλειάδα και στη συνέχεια ενημερώνονται οι δείκτες των τιμών της που είναι πιθανό να έχουν ήδη δημιουργηθεί. Η



Σχήμα 3.7: *Παραδείγματα λειτουργιών drill-down και roll-up*

φάση της εισαγωγής διαφέρει ελάχιστα σε σύγκριση με τη διαδικασία της εισαγωγής που έχει περιγραφεί.

Οι ενημερώσεις του υπάρχοντος συνόλου δεδομένων είναι μία πιο πολύπλοκη διαδικασία. Κατά την εισαγωγή των καινούργιων πλειάδων, η επιλογή του σωστού pivot level είναι κρίσιμη, ώστε να εξασφαλιστεί η ορθότητα των αναζητήσεων. Η επιλογή του pivot level δεν είναι όμως απλή, δεδομένου ότι τα pivot levels των αποθηκευμένων δέντρων για ένα συγκεκριμένο root key μπορεί να ποικίλουν λόγω των re-indexing λειτουργιών που έχουν ήδη εκτελεστεί.

Για την επιλογή του κατάλληλου pivot level λαμβάνονται υπόψη οι ακόλουθες υποθέσεις:

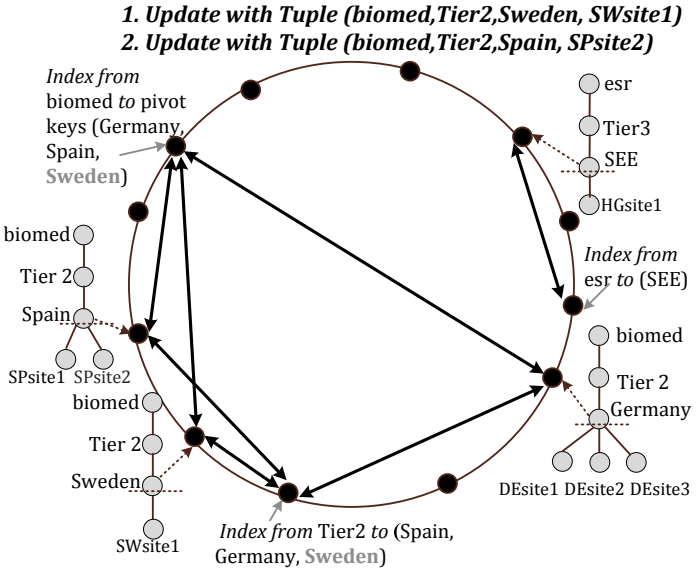
- Εάν μία καινούργια πλειάδα περιέχει ένα pivot key, τότε το συγκεκριμένο κλειδί πρέπει να χρησιμοποιηθεί κατά την εισαγωγή της. Διαφορετικά, οι αναζητήσεις για αυτή την τιμή θα επιστρέψουν μόνο τις τιμές που υπήρχαν πριν από την καινούργια εισαγωγή.

- Ακόμα και αν δεν έχει χρησιμοποιηθεί κάποια από τις τιμές της πλειάδας ως pivot key, μπορεί κάποια από αυτές τις τιμές να είναι ήδη αποθηκευμένη στο δίκτυο. Η επιλογή μίας τέτοιας τιμής ως pivot key θα είχε ως αποτέλεσμα την ανακάλυψη μόνο της καινούργιας πλειάδας κατά τη διάρκεια μίας μελλοντικής αναζήτησης. Για να αποφευχθεί κάτι τέτοιο, το pivot level που θα επιλεχθεί σε αυτήν την περίπτωση πρέπει να είναι ίσο με το MaxPivotLevel που έχει χρησιμοποιηθεί για τις πλειάδες που περιέχουν τις υπάρχοντες τιμές. Οι λειτουργίες re-indexing θα αναλάβουν να προσαρμόσουν κατάλληλα το pivot level για τις καινούργιες πλειάδες.

Η δυσκολία στη διεκπεραίωση των ενημερώσεων αυξάνει λόγω του γεγονότος ότι η πληροφορία για τις αποθηκευμένες τιμές και το MaxPivotLevel είναι κατανεμημένη σε διαφορετικούς κόμβους. Λαμβάνοντας υπόψη αυτό, υλοποιήθηκε ένας κατανεμημένος κατάλογος με τη δημιουργία δεικτών μεταξύ των κόμβων που είναι υπεύθυνοι για το root keys και των κόμβων με τα pivot keys, δηλαδή αποθηκεύονται εγγραφές για κάθε root key και τα αντίστοιχα pivot keys του. Η βελτίωση που επιτεύχθηκε με τη προσθήκη αυτού του καταλόγου δίνει τη δυνατότητα για online ενημερώσεις, ενώ το σύστημα συνεχίζει να εξυπηρετεί αποτελεσματικά τα αιτήματα των χρηστών.

Κατά την εισαγωγή μίας καινούργιας πλειάδας, εκτελείται μία αναζήτηση για το root key. Εάν βρεθεί το root key στον κόμβο που είναι υπεύθυνος για αυτό, τότε διερευνάται εάν κάποια από τις τιμές της πλειάδας αντιστοιχεί σε pivot key. Σε περίπτωση που ισχύει αυτό, τότε η πλειάδα αποθηκεύεται στον κόμβο που είναι υπεύθυνος για το pivot key και οι καινούργιες τιμές της κάτω από το pivot level προστίθενται σε έναν καινούργιο κλάδο του υπάρχοντος δέντρου. Στην αντίθετη περίπτωση, παράγεται το κλειδί που αντιστοιχεί στην τιμή του MaxPivotLevel και χρησιμοποιείται ως pivot key κατά την εισαγωγή της πλειάδας στην επικάλυψη. Η επιλογή του MaxPivotLevel ως pivot level δεν αποκλείει την ύπαρξη των ίδιων τιμών που βρίσκονται

πάνω από το pivot level και σε άλλα δέντρα; συνεπώς μπορεί να έχουν δημιουργηθεί και soft-state δείκτες προς τις τιμές αυτές. Οι δείκτες αυτοί πρέπει να ενημερωθούν έτσι ώστε τα indexed ερωτήματα να επιστρέφουν όλα τα αποτελέσματα. Για την ενημέρωση των δεικτών, ο κόμβος που είναι υπεύθυνος για τη νέα πλειάδα αναζητεί όλες τις τιμές για κάθε επίπεδο ℓ_i που βρίσκεται μεταξύ του root level και του pivot level ($0 < \ell_i < pivotlevel$) και όσοι δείκτες βρεθούν ενημερώνονται κατάλληλα για την καινούργια πλειάδα.



Σχήμα 3.8: *Παραδείγματα ενημέρωσης ενός pivot key που δεν υπάρχει και ενός που υπάρχει ήδη στην επικάλυψη*

Κάποια παραδείγματα για τις πιθανές περιπτώσεις κατά τη διάρκεια μίας ενημέρωσης παρουσιάζονται στο Σχήμα 3.8. Ο κόμβος με το δείκτη για το root key *'biomed'* καταλήγει στο συμπέρασμα ότι καμία από τις τιμές της πλειάδας *'(biomed, Tier2, Sweden, SWsite1)'* δεν αντιστοιχεί σε κάποιο αποθηκευμένο pivot key. Επιπλέον, το root key γνωρίζει ότι το MaxPivotLevel για τα δέντρα του είναι το `Region` και επομένως η πλειάδα εισάγεται με το pivot key που αντιστοιχεί στην τιμή *'Sweden'*. Το καινούργιο pivot key προστίθεται στη λίστα με τα pivot keys για αυτό το root key. Παρόλα αυτά, η τιμή *'Tier2'* έχει ήδη δεικτοδοτηθεί. Κατά την αναζήτηση της τιμής *'Tier2'* σύμφωνα με τη διαδικασία που ακολουθείται για τις ενημερώσεις, ο κόμβος που είναι υπεύθυνος για αυτό το δείκτη ανακαλύπτεται και δημιουργείται ένας καινούργιος δείκτης μεταξύ αυτού του κόμβου και του κόμβου με τα πραγματικά δεδομένα. Κατά την ενημέρωση για την πλειάδα *'(biomed,Tier2,Spain,SPsite2)'*, ο κόμβος που είναι υπεύθυνος για το root key *'biomed'* προχωράει στην εισαγωγή της πλειάδας χρησιμοποιώντας το pivot key που αντιστοιχεί στην τιμή *'Spain'* και έχει ως αποτέλεσμα την προσθήκη ενός νέου κλάδου στον υπάρχον

δέντρο. Η δεικτοδοτημένη τιμή *'Tier2'* δεν επηρεάζεται και για αυτό το λόγο δεν γίνονται περαιτέρω ενέργειες.

3.6 Θέματα προς Συζήτηση

Στην ενότητα αυτήν αναλύονται κάποια θέματα που σχετίζονται με την επιλογή των παραμέτρων που χρησιμοποιούνται στις προτεινόμενες τεχνικές καθώς και βελτιστοποιήσεις που μπορούν να εφαρμοστούν στο σύστημα.

3.6.1 Απαιτήσεις σε μνήμη

Η αρχιτεκτονική του προτεινόμενου συστήματος απαιτεί την αποθήκευση επιπρόσθετης πληροφορίας για την εφαρμογή των προτεινόμενων αλγορίθμων. Αναλυτικότερα, κάθε κόμβος της επικάλυψης είναι πιθανό να αποθηκεύει την ακόλουθη μορφή πληροφορίας ανάλογα με τη συνάρτηση κατακερματισμού που παράγει τα κλειδιά και με το φορτίο ερωτημάτων:

- Δεδομένα: Έχει θεωρηθεί ότι τα δεδομένα που εισάγονται στο σύστημα αποτελούνται από πλειάδες που περιέχουν μία τιμή για κάθε επίπεδο της ιεραρχίας και τα αντίστοιχο fact (ή facts). Όταν μία πλειάδα καταλήγει στον κόμβο που είναι υπεύθυνος για το pivot key της, εισάγεται σε μία δενδρική δομή που περιέχει το συγκεκριμένο pivot key, έτσι ώστε να αποφεύγεται η πολλαπλή αποθήκευση ίδιων τιμών, ειδικά όσον αφορά τις τιμές των επιπέδων που βρίσκονται υψηλότερα στο δέντρο από το pivot level. Επίσης, οι δενδρικές δομές διευκολύνουν την ανακάλυψη των τιμών που εμπεριέχονται στα ερωτήματα κατά τη διάρκεια των αναζητήσεων. Για κάθε αποθηκευμένο δέντρο, διατηρείται τουλάχιστον μία πλειάδα με L αριθμητικές τιμές για στατιστικούς λόγος, όπου η κάθε τιμή αντιστοιχεί σε ένα επίπεδο της ιεραρχίας.

- Root δείκτες (root indices): Κάθε φορά που εισάγεται στην DHT επικάλυψη ένα διαφορετικό root key, δημιουργούνται δείκτες προς τα pivot keys του. Επιπλέον, όταν εισάγεται ένα καινούργιο κλειδί που αντιστοιχεί σε ένα υπάρχον root key, τότε προστίθεται και ο αντίστοιχος δείκτης.

- Soft-state δείκτες (soft-state indices): Τα flooded ερωτήματα έχουν σαν αποτέλεσμα τη δημιουργία soft-state δεικτών, που χρησιμοποιούνται στη συνέχεια για τη γρηγορότερη επίλυση των μελλοντικών ερωτημάτων. Ο μέγιστος αριθμός των επιτρεπόμενων δεικτών ορίζεται με την παράμετρο I_{max} για κάθε κόμβο. Συνεπώς, ο απαιτούμενος αποθηκευτικός χώρος για κάθε δείκτη είναι $O(I_{max})$.

3.6.2 Επιλογή των Παραμέτρων

Η υιοθέτηση διάφορων παραμέτρων στο προτεινόμενο σύστημα επηρεάζει την απόδοσή του ανάλογα με τις τιμές που θα επιλεχθούν για αυτές. Η τιμή του threshold (*thr*) παίζει σημαντικό ρόλο όσον αφορά τον αριθμό των reindexing λειτουργιών που θα εκτελεστούν. Αφενός, η επιλογή μίας υψηλής τιμής ως threshold έχει σαν αποτέλεσμα λιγότερες reindexing λειτουργίες και συνεπώς αυξάνεται η χρήση των soft-state δεικτών. Η περίπτωση αυτή ευνοεί κυρίως φορτία ερωτημάτων που στοχεύουν περισσότερο τα υψηλότερα επίπεδα, ενώ ο αριθμός των διακριτών τιμών στα επίπεδα αυτά είναι περιορισμένος. Συμπερασματικά, η πιθανότητα για πολλαπλά ερωτήματα που αφορούν τις ίδιες τιμές είναι μεγαλύτερη. Αφετέρου, η επιλογή μίας χαμηλότερης τιμής για την παράμετρο *thr* επιφέρει συχνότερες reindexing λειτουργίες που απεικονίζουν τις αλλαγές στις τάσεις των ερωτημάτων. Όμως, το μειονέκτημα της συχνής εκτέλεσης roll-up και drill-down λειτουργιών είναι η αύξηση της κατανάλωσης του εύρους ζώνης για τη μετακίνηση των εμπλεκόμενων πλειάδων και η διαγραφή των υπαρχόντων soft-state δεικτών. Παρόλα αυτά, οι λειτουργίες αυτές είναι ιδιαίτερα αποτελεσματικές όταν είναι δημοφιλείς οι τιμές των χαμηλότερων επιπέδων. Αυτό συμβαίνει, γιατί η συμβολή των δεικτών στην επίλυση των ερωτημάτων δεν είναι επαρκής. Επομένως, η επιλογή χαμηλότερων τιμών για την παράμετρο *thr* ενδείκνυται για φορτία ερωτημάτων που στοχεύουν τα χαμηλότερα επίπεδα των ιεραρχιών και ερωτούν μεγάλο αριθμό διαφορετικών τιμών.

Οι soft-state δείκτες συνεισφέρουν στη βελτίωση της απόδοσης του συστήματος. Όμως η διατήρηση της συνοχής των δεικτών μπορεί να είναι πολύπλοκη και δαπανηρή σε ένα σύστημα που προσαρμόζεται ανάλογα με τις προτιμήσεις των χρηστών. Για αυτό το λόγο, η παράμετρος TTL χρησιμοποιείται: Οι δείκτες που δεν έχουν χρησιμοποιηθεί για μία χρονική περίοδο μεγαλύτερη από το TTL θεωρούνται ότι δεν είναι χρήσιμοι και διαγράφονται. Ο περιορισμός αυτός εξασφαλίζει ότι οι αλλαγές στο σύστημα (π.χ. οι αλλαγές των δεδομένων, οι αφίξεις/αναχωρήσεις των κόμβων, κλπ.) δε θα έχουν ως αποτέλεσμα την ύπαρξη μη έγκυρων δεικτών, οι οποίοι θα επηρεάζουν την ορθότητα των μηχανισμών αναζήτησης. Επίσης, ο αριθμός των δεικτών μπορεί να ρυθμιστεί σύμφωνα με τις δυνατότητες του συστήματος. Αν και η πρόσβαση σε περισσότερη μνήμη γίνεται όλο και πιο εύκολη, η δομή των δεδομένων δεν απαιτεί τη δημιουργία δεικτών 'χωρίς όριο'. Για αυτό το λόγο, όταν ο αριθμός των δεικτών φτάσει το όριο I_{max}, η δημιουργία ενός καινούργιου δείκτη έχει σαν αποτέλεσμα τη διαγραφή του παλαιότερου. Εάν μία δεικτοδοτημένη τιμή απαιτεί περισσότερους από ένας δείκτες, τότε διαγράφονται όλοι τους για λόγους συνοχής. Η ρύθμιση του I_{max} για τη βελτίωση της απόδοσης μπορεί να χρησιμοποιηθεί για την εξαγωγή γνώσης σχετικά με τα αποθηκευμένα δεδομένα (π.χ. πόσο πολωμένη είναι η κάθε ιεραρχία). Συνολικά, το σύστημα τείνει να διατηρεί τους πιο χρήσιμους δείκτες, δηλαδή αυτούς που αναφέρονται στα πιο δημοφιλή δεδομένα.

Η παράμετρος του εύρους παραθύρου W σχετίζεται με τον αριθμό των προηγούμενων στατιστικών που αποθηκεύονται από κάθε κόμβο για τον υπολογισμό των μέσων ρυθμών που αφορούν τα εισερχόμενα ερωτήματα. Μία μεγάλη τιμή για το W συνεπάγεται ότι το σύστημα δεν μπορεί να αντιληφθεί γρήγορα τις μεταβολές στα φορτία ερωτημάτων, ενώ μία μικρή τιμή μπορεί να έχει αρνητική επίπτωση λόγω της μη ύπαρξης επαρκούς πληροφορίας για να γίνουν οι reindexing λειτουργίες ή της λήψης λανθασμένων αποφάσεων. Για την εκτίμηση της τιμής του μπορεί να θεωρηθεί ότι $W = O(1/\lambda)$, δηλαδή να συσχετιστεί το εύρος του παραθύρου με τα ερωτήματα. Όσο πιο συχνά είναι τα ερωτήματα, τόσο μικρότερο μπορεί να είναι το W και αντίστροφα.

3.6.3 Συνοχή των Δεδομένων

Μία άλλη συνιστώσα που πρέπει να ληφθεί υπόψη είναι ποιες στρατηγικές πρέπει να ακολουθηθούν, ώστε να εξασφαλιστεί η συνοχή των δεδομένων (data consistency) κατά τη διάρκεια των διάφορων λειτουργιών. Σε ένα ιδιαίτερα δυναμικό περιβάλλον, όπως αυτό που θεωρείται για τις προτεινόμενες μεθόδους που επιτρέπουν την εκτέλεση των ενημερώσεων και των reindexing λειτουργιών με online τρόπο, πρέπει να δοθεί ιδιαίτερη προσοχή για την εξασφάλιση της συνοχής των δεδομένων, έστω και αν αυξάνεται το κόστος επικοινωνίας και συντήρησης. Ένας σημαντικός παράγοντας αναφορικά με τα θέματα του data consistency είναι η εγκυρότητα των root δεικτών, ώστε να εξασφαλιστεί η σωστή εκτέλεση των ενημερώσεων. Τα root keys πρέπει να γνωρίζουν συνεχώς όλα τα αποθηκευμένα pivot keys και να επιβεβαιώνουν την ύπαρξη τους μετά από κάποιο χρονικό διάστημα. Για αυτό το λόγο, συνιστάται η αποστολή περιοδικών μηνυμάτων από τα root keys για να εξακριβώσουν την εγκυρότητα της πληροφορίας που διατηρούν για τα pivot keys τους.

Επιπλέον, είναι σημαντικό να διασφαλιστεί η συμμετοχή όλων των σχετικών δέντρων στις roll-up και drill-down λειτουργίες που τα αφορούν αλλά και να εξασφαλιστεί η αποτελεσματικότητα των μηχανισμών locking κατά την εκτέλεση των διαδικασιών για re-indexing, ώστε οι αναζητήσεις που θα ακολουθήσουν να ανακτούν ολοκληρωμένα αποτελέσματα. Κατά τη διάρκεια των λειτουργιών αυτών, μπορεί να παρατηρηθεί μία παροδική αστάθεια του συστήματος κατά την επίλυση των ερωτημάτων λόγω της μετακίνησης των δεδομένων. Για την αντιμετώπιση αυτού του φαινομένου, τα δεδομένα αποθηκεύονται προσωρινά (cache of data) στους αρχικούς κόμβους, μέχρι να ολοκληρωθεί η διαδικασία του re-indexing. Μόλις εισαχθούν όλες οι πλειάδες στο καινούργιο pivot level και το root key ολοκληρώσει την ενημέρωση των δεικτών προς τα καινούργια pivot keys, τα παλιά δέντρα διαγράφονται από τους αρχικούς κόμβους μετά τη λήξη του TTL. Η επίλυση των αναζητήσεων για τις τιμές που γίνονται re-indexed καθυστερούν στους καινούργιους κόμβους μέχρι να ολοκληρωθεί η συναλλαγή. Εφόσον το root key γνωρίζει τις λειτουργίες που είναι υπό εκτέλεση στα δεδομένα του, καθίσταται επίσης υπεύθυνο για

την προώθηση των ερωτημάτων που δεν είναι exact-match είτε στους κόμβους που ήταν αρχικά υπεύθυνοι για τα δεδομένα είτε σε αυτούς που έγιναν υπεύθυνοι μετά το re-indexing. Επίσης δεν επιτρέπεται η δημιουργία των soft-state δεικτών για οποιαδήποτε τιμή που ανήκει στα δέντρα που γίνονται re-indexed, μέχρι αυτό να ολοκληρωθεί.

Τέλος, δίνεται ιδιαίτερη προσοχή κατά τη δημιουργία των soft-state δεικτών για να αποφευχθούν μη έγκυροι δείκτες. Εκτός από τη χρήση του TTL, όταν παρατηρείται κάποιο πρόβλημα κατά τη δημιουργία ενός δείκτη, τότε αυτός διαγράφεται.

3.7 Πειραματική Αξιολόγηση

3.7.1 Περιγραφή της Εξομοίωσης

Στη συνέχεια παρουσιάζεται μία ολοκληρωμένη αξιολόγηση του συστήματος που προκύπτει από την εφαρμογή των προτεινόμενων τεχνικών. Η απόδοση του συστήματος βασίζεται σε μία σημαντικά τροποποιημένη έκδοση του εξομοιωτή για τοπολογίες της DHT επικάλυψης που παρέχεται από το FreePastry [Fre], αν και οποιαδήποτε υλοποίηση ενός DHT πρωτοκόλλου θα μπορούσε να χρησιμοποιηθεί. Το μέγεθος του δικτύου που έχει προεπιλεγεί κατά την εκτέλεση των πειραμάτων είναι 256 κόμβοι, οι οποίοι επιλέγονται τυχαία για να εισάγουν πλειάδες και να θέτουν ερωτήματα.

Στο μεγαλύτερο μέρος των πειραμάτων χρησιμοποιούνται “συνθετικά” δεδομένα που παράγονται με τη χρήση γεννήτριας. Τα εισαγόμενα δεδομένα σχηματίζουν δέντρα, όπου η κάθε τιμή έχει το πολύ ένα γονέα. Κάθε διακριτή τιμή στο επίπεδο ℓ_i έχει ένα σταθερό αριθμό παιδιών στο ℓ_{i+1}. Επίσης έχει προεπιλεχθεί το σύνολο των δεδομένων να αποτελείται από 100k πλειάδες, οι οποίες μπορούν να οργανωθούν σε ιεραρχίες που αποτελούνται από τέσσερα επίπεδα (see Figure 3.2) με ένα αριθμητικό fact (π.χ. `CPU_time`). Ο αριθμός των διακριτών τιμών σε κάθε επίπεδο είναι $|\ell_0 = 100|, |\ell_1 = 1000|, |\ell_2 = 10000|$ και $|\ell_3 = 100000|$. Το αρχικό επίπεδο που έχει επιλεχθεί είναι το ℓ_1, εκτός και αν δηλώνεται η επιλογή ενός διαφορετικού επιπέδου.

Όσον αφορά τα φορτία ερωτημάτων που χρησιμοποιούνται στα πειράματα ακολουθείται μία προσέγγιση δύο σταδίων: αρχικά επιλέγεται το επίπεδο που θα στοχεύσει το ερώτημα σύμφωνα με την κατανομή του *levelDist*, ενώ στη συνέχεια προσδιορίζεται η τιμή που θα ερωτηθεί από το επιλεγμένο επίπεδο βάσει της κατανομής του *valueDist*. Στα πειράματα που ακολουθούν, χρησιμοποιείται η κατανομή Zipfian ($p_i \sim 1/i^{\theta}$) για το *levelDist*, ενώ η πόλωση σε κάθε επίπεδο εκφράζεται με τις κατανομές uniform, 80/20, 90/10 and 99/1 για το *valueDist*.

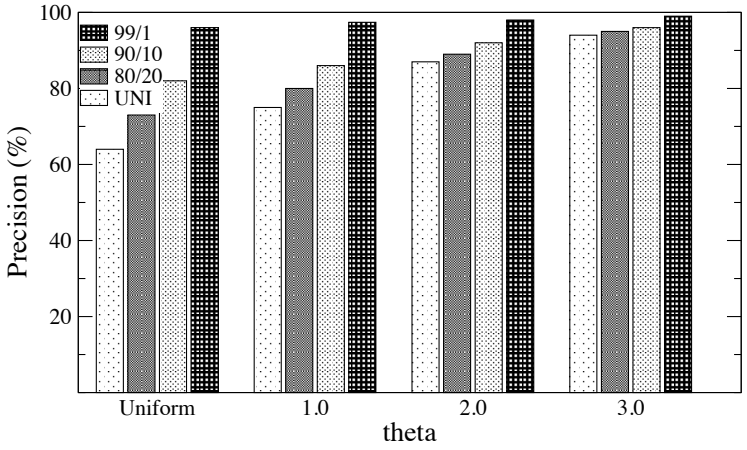
Τα ερωτήματα θέτονται με ένα μέσο ρυθμό $1\frac{query}{time\_unit}$, σε συνολικό χρόνο εξομοίωσης 50k time units. Τα αποτελέσματα που παρουσιάζονται αφορούν ερωτήματα μίας διάστασης που περιγράφεται από μία ιεραρχία με πολλαπλά επίπεδα. Η προεπιλεγμένη τιμή για το *thr* είναι το

0.3, που είναι μία ικανοποιητική τιμή για την αποφυγή συχνών και μη απαραίτητων προσπαθειών για re-indexing. Η εξομοίωση της συμπεριφοράς του συστήματος για διαφορετικές τιμές του *thr* που είναι κοντά στην προεπιλεγμένη τιμή παρουσιάζει μικρές ποιοτικές διαφορές. Η προεπιλεγμένη τιμή για το W, το οποίο σχετίζεται με το πόσο γρήγορα το σύστημα μπορεί να προσαρμοστεί στις αλλαγές, είναι ίσο με 1000 time units. Τέλος, η τιμή για το TTL είναι πολύ μεγάλη (ουσιαστικά οι δείκτες δεν λήγουν).

Σε αυτή την ενότητα, ένας βασικός στόχος είναι η επίδειξη της απόδοσης και της προσαρμοστικότητας του προτεινόμενου συστήματος υπό διάφορες συνθήκες. Ο στόχος είναι να αποδειχθεί ότι το σύστημα είναι ιδιαίτερα αποτελεσματικό για διάφορα σύνολα δεδομένων και φορτία ερωτημάτων και ότι μπορεί να προσαρμοστεί γρήγορα σε ξαφνικές αλλαγές της πόλωσης (skew) χωρίς να απαιτείται η συνεχής τροποποίηση των προεπιλεγμένων τιμών των διάφορων παραμέτρων. Ειδικότερα, το ποσοστό των ερωτημάτων που επιλύονται χωρίς flooding ορίζεται ως precision και είναι αυτό που μετράται κυρίως.

3.7.2 Απόδοση για Πόλωση προς Διαφορετικά Επίπεδα

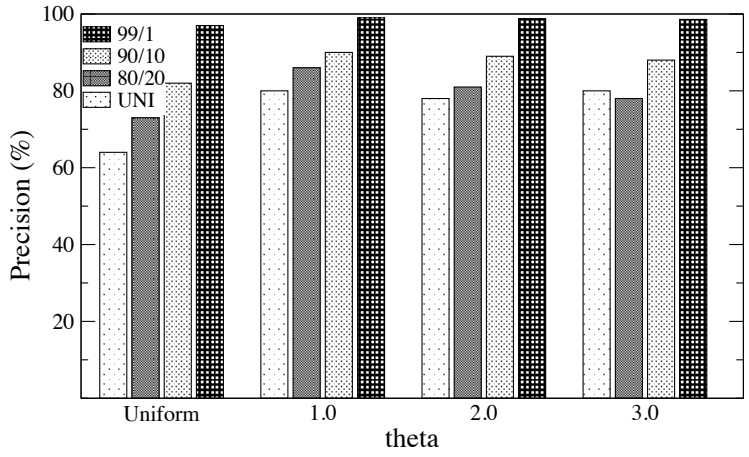
Στην πρώτη ομάδα πειραμάτων αναγνωρίζεται η συμπεριφορά του προτεινόμενου συστήματος για διάφορα φορτία ερωτημάτων. Ειδικότερα, ο αριθμός των ερωτημάτων που απευθύνονται σε κάθε επίπεδο διαφοροποιείται με την αύξηση της παραμέτρου θ της κατανομής που ακολουθεί το *levelDist*. Για κάθε τιμή του θ, οι τιμές μέσα σε κάθε επίπεδο επιλέγονται με τη χρήση των τεσσάρων διαφορετικών κατανομών που έχουν οριστεί για το *valueDist*.



Σχήμα 3.9: *Αποτελέσματα για το precision για διάφορα φορτία ερωτημάτων που είναι πολωμένα προς το* ℓ_0

Στο Σχήμα 3.9 παρουσιάζονται τα αποτελέσματα σχετικά με το precision που καταγράφηκε για φορτία ερωτημάτων που είναι πολωμένα προς τα το γενικότερο επίπεδο των ιεραρχιών

(δηλαδή προς το ℓ_0). Όσο αυξάνει η τιμή της παραμέτρου θ για το *levelDist*, η απόδοση της προτεινόμενης μεθόδου βελτιώνεται: Οι re-indexing διαδικασίες εκτελούνται γρηγορότερα και αυξάνεται και ο αριθμός των exact match ερωτημάτων λόγω του επιλεγμένου επιπέδου. Με την αύξηση της πόλωσης για το *valueDist*, παρατηρείται ιδιαίτερα υψηλό precision επειδή το ποσοστό των δημοφιλών ερωτημάτων και η πυκνότητα των ερωτημάτων για συγκεκριμένες πλειάδες αυξάνονται. Ένας άλλος παράγοντας που παίζει μεγάλο ρόλο είναι ο περιορισμένος αριθμός των διαφορετικών τιμών στο ℓ_0. Προφανώς ο αριθμός αυτών των τιμών είναι αρκετά μικρός σε σύγκριση με αυτόν των χαμηλότερων επιπέδων και επιτρέπει την αύξηση της χρήσης των soft-state δεικτών, καθώς και τη γρηγορότερη εκτέλεση των re-indexing λειτουργιών, όταν αυτό χρειάζεται. Για μία ομάδα τιμών του θ, η προτεινόμενη μέθοδος για re-indexing συμπεριφέρεται αποτελεσματικότερα καθώς η κατανομή γίνεται πιο πολωμένη, δεδομένου ότι περισσότερα ερωτήματα αφορούν λιγότερες διακριτές τιμές. Τέλος, παρατηρείται ότι για μεγαλύτερες τιμές του θ είναι μικρότερη η διαφορά στο precision για διαφορετικές κατανομές του *valueDist*.



Σχήμα 3.10: *Αποτελέσματα για το precision για διάφορα φορτία ερωτημάτων που είναι πολωμένα προς το* ℓ_3

Το Σχήμα 3.10 παρουσιάζει τα αποτελέσματα για φορτία ερωτημάτων που κατευθύνονται προς τα χαμηλότερα επίπεδα της ιεραρχίας. Και σε αυτήν την περίπτωση παρατηρείται μία παρόμοια τάση στην αποτελεσματικότητα του συστήματος όσο αυξάνεται η πόλωση στην κατανομή του *valueDist* και η προτεινόμενη μέθοδος έχει σαν αποτέλεσμα υψηλό precision, αν και κάπως μειωμένο σε σύγκριση με αυτό που επιτεύχθηκε στην προηγούμενη ομάδα πειραμάτων. Επίσης, τα αποτελέσματα δείχνουν ότι το precision για τις ίδιες κατανομές του *valueDist* μειώνεται όσο αυξάνεται το θ: Αυτό συμβαίνει λόγω του γεγονότος ότι το επίπεδο ℓ_3 έχει ένα σημαντικά μεγαλύτερο αριθμό τιμών. Με την αύξηση του αριθμού των ερωτημάτων για αυτές τις τιμές, αυξάνεται και η πιθανότητα (σε σχέση βέβαια με την επιλογή του *valueDist*) των ερωτημάτων

που περιλαμβάνουν μη δεικτοδοτημένες τιμές. Παρόλα αυτά, η μείωση είναι μικρότερη όταν το *valueDist* γίνεται πιο πολωμένο.

3.7.3 Μελέτη των Προτεινόμενων Τεχνικών για Φορτία με Πολλαπλά Πολωμένα Σημεία

Στην επόμενη κατηγορία πειραμάτων, εξετάζεται η απόδοση του συστήματος σε ένα πιο απαιτητικό τύπο φορτίου ερωτημάτων, που αναφέρεται ως *MULTI*: Ενώ διαφορετικά επίπεδα λαμβάνουν τον ίδιο αριθμό ερωτημάτων, ένα διαφορετικό μέρος των συνολικών δεδομένων ζητιέται. Ειδικότερα, όλα τα επίπεδα χωρίζονται σε τέταρτα και τα ερωτήματα κατευθύνονται προς ένα διαφορετικό επίπεδο για κάθε τέταρτο (με τη χρήση διαφορετικών τιμών του *valueDist*), με τέτοιο τρόπο ώστε κανένα τέταρτο να μη σχετίζεται με κάποιο άλλο στο σύνολο των δεδομένων. Αυτό το φορτίο ερωτημάτων είναι αντιπροσωπευτικό για την αποτελεσματικότητα που επιδεικνύουν οι προτεινόμενες μέθοδοι, αφού οι λειτουργίες re-indexing πρέπει να προσαρμόσουν διαφορετικά δεδομένα σε διαφορετικά επίπεδα λεπτομέρειας. Ο Πίνακας 3.1 συνοψίζει τα αποτελέσματα για το συγκεκριμένο πείραμα, όπου εκτός από το precision, καταγράφονται και το κόστος που σχετίζεται με τον αριθμό των πλειάδων που έγιναν re-indexed όπως και ο αριθμός των συνολικών λειτουργιών roll-up και drill-down.

Πίνακας 3.1: *Σύγκριση της απόδοσης του συστήματος για τα φορτία ερωτημάτων MULTI και για διαφορετικές τιμές του valueDist*

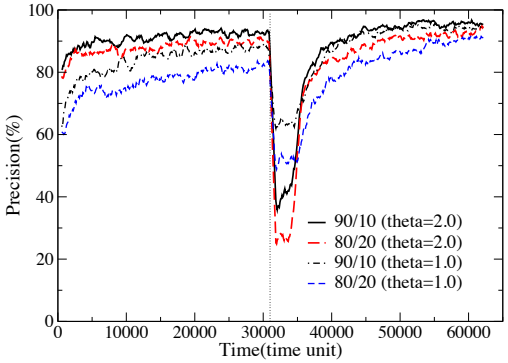
valueDist	**precision (%)**	**#roll-ups**	**#rolled-up trees**	**#drill-downs**	**re-inserts (%)**
uniform	92.0	25	250	500	75
80/20	94.3	25	250	171	42
90/10	95.2	25	250	51	30
99/1	99.5	1	10	6	1.6

Η τεχνική που ακολουθείται αποδεικνύεται ιδιαίτερα αποτελεσματική και στους τέσσερις τύπους φορτίων ερωτημάτων και επιτυγχάνει πολύ υψηλά ποσοστά precision (μεταξύ 92% και 100%) με μικρό κόστος: Ο μεγαλύτερος αριθμός λειτουργιών re-indexing λαμβάνει χώρα όταν οι διαφορετικές τιμές των επιπέδων ρωτιούνται ομοιόμορφα, όπου το 75% των πλειάδων εισάγεται ξανά στην επικάλυψη μετά από τις αντίστοιχες λειτουργίες re-indexing. Καθώς το *valueDist* γίνεται πιο πολωμένο, ο αριθμός των λειτουργιών re-indexing μειώνεται. Αυτό αποδεικνύει με σαφήνεια ότι η προτεινόμενη μέθοδος προσαρμόζει τον αριθμό των λειτουργιών re-indexing ανάλογα με τις ανάγκες: Ο αριθμός των δέντρων που επαναδεικτοδοτούνται είναι ανάλογος

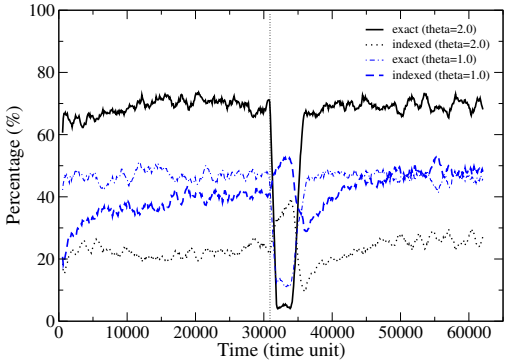
του αριθμού των μοναδικών δέντρων που είναι δημοφιλή. Αυτή η ιδιότητα είναι ιδιαίτερα σημαντική, ειδικά αν ληφθεί υπόψη ότι στην πλειοψηφία των εφαρμογών αναμένονται δυναμικά και ιδιαίτερα πολωμένα φορτία.

3.7.4 Απόδοση σε Δυναμικά Περιβάλλοντα

Η προσαρμοστικότητα και η απόδοση του συστήματος που χρησιμοποιεί τις προτεινόμενες τεχνικές σε δυναμικά περιβάλλοντα εξετάζεται στην επόμενη ομάδα πειραμάτων. Η κατανομή που ακολουθούν τα φορτία των ερωτημάτων χαρακτηρίζεται από μία ξαφνική αλλαγή στην πόλωση από το επίπεδο ℓ_0 στο επίπεδο ℓ_3 και αντίστροφα αφού έχουν τεθεί τα μισά από τα ερωτήματα που εξομοιώνονται.



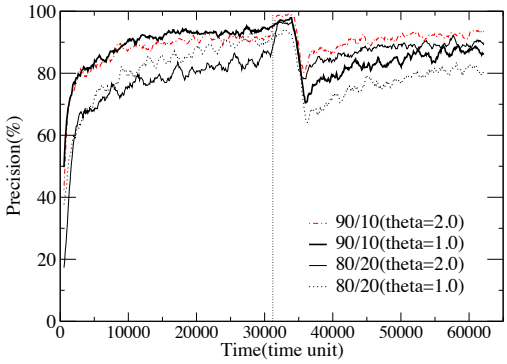
Σχήμα 3.11: *Το precision κατά τη διάρκεια του χρόνου, όταν η πόλωση αλλάζει κατεύθυνση από το ℓ_0 στο ℓ_3*



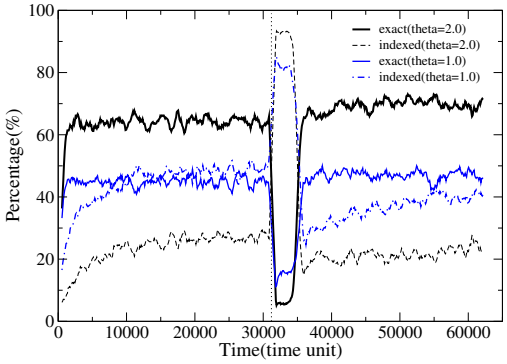
Σχήμα 3.12: *Το precision (χωρισμένο σε exact match και indexed ερωτήματα) κατά τη διάρκεια του χρόνου για valueDist 90/10, όταν η πόλωση αλλάζει κατεύθυνση από το ℓ_0 στο ℓ_3*

Το Σχήμα 3.11 παρουσιάζει τη συμπεριφορά του συστήματος όταν το φορτίο των ερωτημάτων αλλάζει κατεύθυνση, δηλαδή ενώ αρχικά στόχευε τα γενικότερα επίπεδα, στη συνέχεια αρχίζει να στοχεύει τα χαμηλότερα επίπεδα. Τα αποτελέσματα δείχνουν ότι για όλα τα φορτία που εξομοιώθηκαν, το σύστημα αυξάνει το precision του λόγω του συνδυασμού των soft-state δεικτών και των re-indexing λειτουργιών, ενώ η πλειοψηφία των ερωτημάτων απαντιέται ως exact match ερωτήματα. Το precision φτάνει πάνω από 90% για $\theta = 2.0$ και πάνω από 80% για $\theta = 1.0$ πριν την αλλαγή της πόλωσης. Κατά τη διάρκεια της μεταβατικής φάσης, το flooding των ερωτημάτων αυξάνει, αλλά το σύστημα καταφέρνει γρήγορα να ανακάμψει και να ανακτήσει την προηγούμενη απόδοση του (μόνο μετά από το 5% των ερωτημάτων). Η απότομη μείωση του precision παρατηρείται ακριβώς κατά την αλλαγή στις τάσεις των ερωτημάτων: Στο επίπεδο ℓ_3 υπάρχει ένας μεγαλύτερος αριθμός διακριτών τιμών και για αυτό το λόγο η ύπαρξη χρήσιμων δεικτών είναι λιγότερο πιθανή. Η συνεισφορά των soft-state δεικτών δεν αρκεί ώστε να αποφευχθεί το flooding μέχρι να αρχίσουν να εκτελούνται οι λειτουργίες drill-down. Σε αυτό το στάδιο,

όσο μεγαλύτερη είναι η τιμή του θ, τόσο μεγαλύτερη είναι η μείωση του precision, αλλά γίνεται γρηγορότερα η ανάκαμψη: Όπως φαίνεται στο Σχήμα 3.12 για τα φορτία με *valueDist* 90/10, τόσο τα exact match όσο και τα indexed ερωτήματα είναι λιγότερα για $\theta = 2.0$. Αυτό συμβαίνει, γιατί τα ερωτήματα είναι πιο πολωμένα προς το επίπεδο ℓ_3 και επωφελούνται λιγότερο από τα δέντρα που έχουν κάνει ήδη roll-up. Ωστόσο, όσο αυξάνει το θ, οι αποφάσεις για drill-down λαμβάνονται γρηγορότερα και ευνοούν την αύξηση των exact match ερωτημάτων, που αποτελούν την πλειοψηφία.



Σχήμα 3.13: *Το precision κατά τη διάρκεια του χρόνου, όταν η πόλωση αλλάζει κατεύθυνση από το ℓ_3 στο ℓ_0*



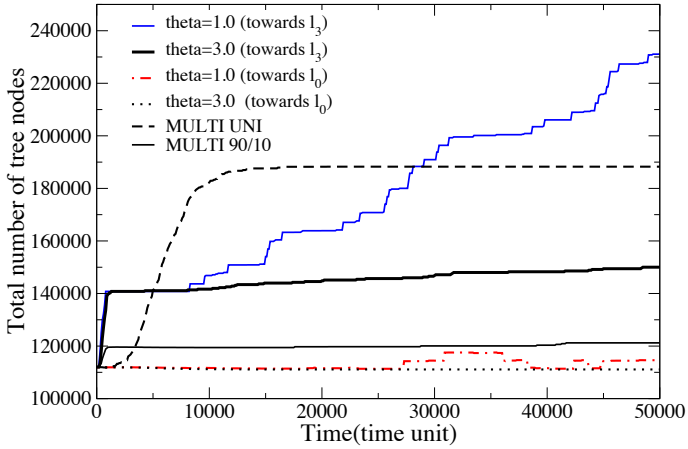
Σχήμα 3.14: *Το precision (χωρισμένο σε exact match και indexed ερωτήματα) κατά τη διάρκεια του χρόνου για valueDist 90/10, όταν η πόλωση αλλάζει κατεύθυνση από το ℓ_3 στο ℓ_0*

Το precision μελετάται και στην περίπτωση της απότομης αλλαγής της κατεύθυνσης του φορτίου ερωτημάτων από το ℓ_3 στο ℓ_0 για διάφορα φορτία ερωτημάτων και απεικονίζεται στο Σχήμα 3.13. Κατά τη διάρκεια των σταθερών φάσεων της εξομοίωσης παρατηρούνται παρόμοιες τάσεις με τις αντίστοιχες που παρατηρήθηκαν στα πολωμένα φορτία ερωτημάτων προς μία κατεύθυνση και το σύστημα διατηρεί το precision υψηλό (πάνω από 80%) για όλα τα φορτία. Καθώς το *valueDist* γίνεται όλο και πιο πολωμένο, καταγράφονται υψηλότερα precision, αφού ο αριθμός των δημοφιλών τιμών συρρικνώνεται και οι λειτουργίες drill-down εκτελούνται γρηγορότερα συμβάλλοντας στην προσαρμοστικότητα του συστήματος. Μετά την αλλαγή στην κατεύθυνσης της πόλωσης, τα flooded ερωτήματα είναι λιγότερα για τα φορτία ερωτημάτων με $\theta = 2.0$ (παρατηρείται αντίθετη συμπεριφορά σε σχέση με αυτή του προηγούμενου πειράματος). Το Σχήμα 3.14 παρουσιάζει μία αναλυτικότερη εικόνα του τρόπου επίλυσης των ερωτημάτων μετά τη ξαφνική αλλαγή στην κατεύθυνση της πόλωσης. Λόγω του μικρότερου αριθμού των διακριτών τιμών στα υψηλότερα επίπεδα, η χρήση των δεικτών είναι αποδοτική. Τα roll-up που διαδραματίζονται καταστρέφουν τους υπάρχοντες δείκτες και αυτό επηρεάζει αρνητικά την απόδοση του συστήματος παροδικά. Μόλις όμως ολοκληρωθούν, το σύστημα επανακτά την αποδοτικότητα του, ενώ σημειώνεται σημαντική αύξηση στα exact match ερωτήματα.

Η σύγκριση των αποτελεσμάτων μεταξύ των δύο ομάδων πειραμάτων αποκαλύπτει ότι οι soft-state δείκτες μπορούν να διατηρήσουν το υψηλό precision του συστήματος όταν η πόλωση κατευθύνεται προς τα υψηλότερα επίπεδα, τα οποία διακρίνονται από μικρότερο αριθμό διακριτών τιμών. Αντιθέτως, η προσαρμοστικότητα του συστήματος εξαρτάται σημαντικά από τις λειτουργίες re-indexing, όταν είναι πιο δημοφιλή τα χαμηλότερα επίπεδα των ιεραρχιών. Παρόλα αυτά, το σύστημα χρειάζεται περιορισμένο χρόνο για να οργανώσει τη δεικτοδότηση του και γίνει ιδιαίτερα αποτελεσματικό και στις δύο περιπτώσεις.

3.7.5 Απαιτήσεις για Αποθηκευτικό Χώρο ως Συνάρτηση του Αριθμού των Κόμβων

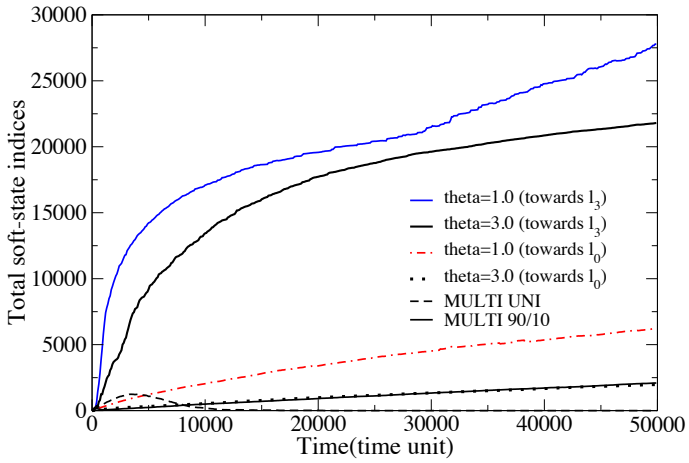
Οι πλειάδες που εισάγονται στο σύστημα αποθηκεύονται σε δενδρικές δομές με το ίδιο pivot key για την αποφυγή της αποθήκευσης πλεονάζουσας πληροφορίας για τις τιμές που βρίσκονται πάνω από το pivot level. Ωστόσο, οι τιμές αυτές μπορεί να βρίσκονται σε περισσότερα δέντρα που είναι αποθηκευμένα σε διαφορετικούς κόμβους της επικάλυψης ανάλογα με το επιλεγμένο pivot level. Ο συνολικός αριθμός των κόμβων των δέντρων (tree nodes) λαμβάνει τη μικρότερη τιμή του όταν το root level επιλέγεται ως *pivotlevel* και τη μέγιστη τιμή του όταν το χαμηλότερο επίπεδο της ιεραρχίας είναι το *pivotlevel* όλων των δέντρων (≈111K και 400K nodes αντίστοιχα για το σύνολο των δεδομένων που εισάγονται). Κάθε κόμβος του δέντρου (που αναφέρεται ως *tree node*) αντιπροσωπεύει την τιμή μίας πλειάδας για αυτό το επίπεδο. Ο συνολικός αριθμός των tree nodes κατά τη διάρκεια του χρόνου εξομοίωσης απεικονίζεται στο Σχήμα 3.15. Η αύξηση του αριθμού των tree nodes υποδηλώνει την εκτέλεση λειτουργιών drill-down, ενώ η μείωση τους συσχετίζεται με τις συνέπειες των λειτουργιών roll-up. Συνεπώς, ο συνολικός αριθμός των tree nodes κατά τη διάρκεια του χρόνου σχετίζεται με τη εξέλιξη των λειτουργιών re-indexing. Τα ερωτήματα που κατευθύνονται προς το ℓ_3 όταν το $\theta = 1.0$ οδηγούν το σύστημα να επανεισάγει τις πλειάδες με πιο αργό ρυθμό σε χαμηλότερα επίπεδα της ιεραρχίας, τα οποία τείνουν να είναι και τα πιο δημοφιλή. Εφόσον όμως η διαφορά στο popularity μεταξύ των επιπέδων δεν είναι σημαντική, οι λειτουργίες drill-down εκτελούνται καθόλη τη διάρκεια της εξομοίωσης. Στην περίπτωση του αντίστοιχου φορτίου ερωτημάτων για $\theta = 3.0$, η αύξηση των tree nodes είναι πιο απότομη και ολοκληρώνεται σε μικρότερο χρονικό διάστημα κατά τα αρχικά στάδια της εξομοίωσης. Τα πιο δημοφιλή δέντρα επαναεισάγουν γρηγορότερα τις πλειάδες τους με χρήση κλειδιών από χαμηλότερα επίπεδα και για αυτό το λόγο δεν παρατηρούνται μεγάλες διακυμάνσεις κατά τη διάρκεια της υπόλοιπης εξομοίωσης. Τα φορτία ερωτημάτων που κατευθύνονται προς το ℓ_0 δεν προκαλούν σημαντικές μεταβολές στο συνολικό αριθμό των tree nodes. Και στις δύο περιπτώσεις του *levelDist* σε αυτά τα φορτία ερωτημάτων, ο αριθμός των tree nodes διατηρείται κοντά στην αρχική τιμή του, δεδομένου ότι ο συνολικός αριθμός των tree nodes όταν χρησιμοποιείται το ℓ_1 ως pivot level δε διαφέρει σημαντικά σε σύγκριση με τον αριθμό των tree nodes



Σχήμα 3.15: *Ο συνολικός αριθμός των κόμβων στα δέντρα (tree nodes) κατά τη διάρκεια του χρόνου για φορτία ερωτημάτων πολωμένα προς το l_0 και προς το l_3 (όπου το* valueDist *είναι 90/10) και για* MULTI *φορτία ερωτημάτων (όπου το* valueDist *είναι ομοιόμορφο και 90/10)*

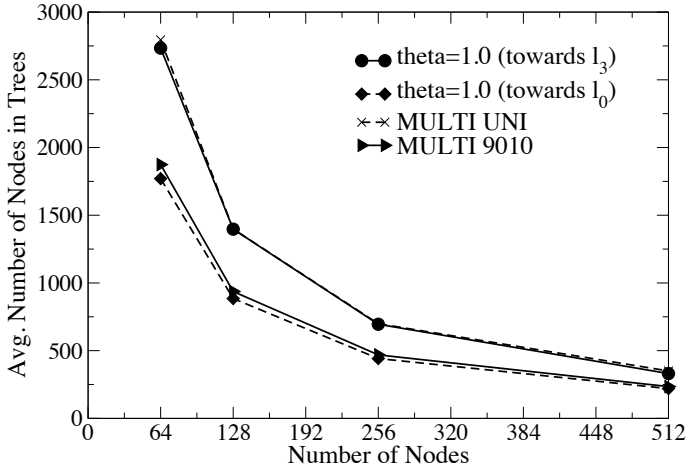
όταν χρησιμοποιείται το ℓ_0 ως pivot level. Τέλος, όσον αφορά τα *MULTI* φορτία ερωτημάτων, οι λειτουργίες roll-up και drill-down συνυπάρχουν. Όταν ο αριθμός των ερωτημάτων στοχεύει ομοιόμορφα τις τιμές του κάθε επιπέδου, η επαναεισαγωγή των πλειάδων συνεχίζεται για ένα μεγαλύτερο χρονικό διάστημα και οι πλειάδες αυτές είναι περισσότερες σε σύγκριση με αυτές για το φορτίο ερωτημάτων με *valueDist* 90/10. Το popularity των επιπέδων αλλάζει για διαφορετικές ομάδες δέντρων στα φορτία δεδομένων *MULTI* και επομένως η εισαγωγή των πλειάδων στο καταλληλότερο επίπεδο εκτελείται ταυτόχρονα για τα διάφορα δέντρα σε σύγκριση με τα φορτία ερωτημάτων που είναι πολωμένα προς το ℓ_3 για $\theta = 1.0$.

Ο συνολικός αριθμός των soft-state δεικτών που υπάρχουν στο σύστημα εμφανίζεται στο Σχήμα 3.16. Η δημιουργία των soft-state δεικτών επηρεάζεται από την εξέλιξη των λειτουργιών re-indexing. Σύμφωνα με την περιγραφόμενη στρατηγική, ένας soft-state δείκτης δημιουργείται μετά το flooding ενός ερωτήματος ως μία συμπληρωματική λύση στη προσαρμογή του pivot level. Η απότομη κλίση στις καμπύλες των φορτίων που κατευθύνονται προς το επίπεδο ℓ_3 δικαιολογείται λαμβάνοντας υπόψη το γεγονός ότι ο αριθμός των διαφορετικών τιμών στα χαμηλότερα επίπεδα είναι μεγαλύτερος σε σχέση με τον αντίστοιχο αριθμό στα υψηλότερα επίπεδα. Η αποδοτικότητα των λειτουργιών re-indexing γίνεται περισσότερο προφανής στην περίπτωση του φορτίου *MULTI* που ακολουθεί ομοιόμορφη κατανομή για την επιλογή τιμών μέσα σε κάθε επίπεδο. Οι καινούργιοι δείκτες δημιουργούνται προσωρινά και διαγράφονται μόλις αρχίσουν να εκτελούνται οι λειτουργίες roll-up και drill-down, που συνεισφέρουν στην προσαρμογή του συστήματος στις τάσεις των εισερχόμενων ερωτημάτων.



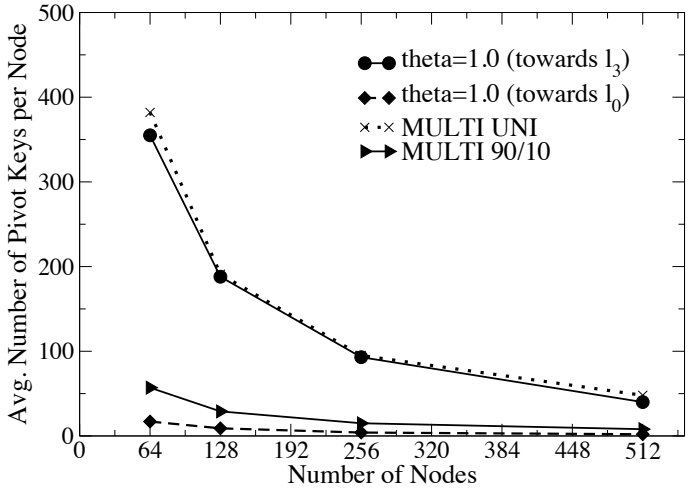
Σχήμα 3.16: *Ο συνολικός αριθμός των soft-state δεικτών κατά τη διάρκεια του χρόνου για φορτία ερωτημάτων πολωμένα προς το ℓ_0 και το ℓ_3 με* valueDist *90/10 και φορτία ερωτημάτων* MULTI *με* valueDist *που ακολουθεί την ομοιόμορφη και τη 90/10 κατανομή*

Τα διαφορετικά αντικείμενα που αποθηκεύονται στους κόμβους της επικάλυψης μπορούν να χωριστούν στις ακόλουθες κατηγορίες: δενδρικές δομές που αποθηκεύουν τις τιμές των ιεραρχιών, root δείκτες που δημιουργούνται κατά την εισαγωγή ενός καινούργιου root value και γνωρίζουν τα pivot keys της, soft-state δείκτες και στατιστική πληροφορία που διατηρείται για τη λήψη των αποφάσεων για re-indexing. Τα πειραματικά αποτελέσματα που απεικονίζονται στα Σχήματα 3.17-3.19 αναφέρονται στο μέσο αριθμό των δεδομένων αυτών ανά κόμβο ή στα αντικείμενα που καθορίζουν τον όγκο τους. Οι τιμές αυτές είναι οι μέσες τιμές κατά τη διάρκεια της συνολικής εξομοίωσης και υπολογίστηκαν από τις τιμές που μετρήθηκαν κατά τακτά χρονικά διαστήματα. Συνεπώς, οι τιμές που καταγράφηκαν ανταποκρίνονται και στις διάφορες αλλαγές που προκλήθηκαν από τις λειτουργίες roll-up και drill-down. Το φορτίο *MULTI* με ομοιόμορφη κατανομή και το πολωμένο φορτίο προς το ℓ_3 με $\theta = 1.0$ έχουν το μέγιστο αριθμό tree nodes, γεγονός που είναι σύμφωνο με τα αποτελέσματα του Σχήματος 3.15, όπου οι καμπύλες αυτών των φορτίων παρουσιάζουν και τις υψηλότερες τιμές. Πρέπει να σημειωθεί ότι οι τιμές που παρουσιάζονται για το φορτίο *MULTI* είναι και οι μεγαλύτερες τιμές που μπορούν να καταγραφούν για το συγκεκριμένο τύπο φορτίου, αφού μελετάται η περίπτωση όπου το *valueDist* είναι ομοιόμορφο. Επομένως, όλα τα δέντρα του συνόλου των δεδομένων επανεισάγουν τις πλειάδες τους στο κατάλληλο επίπεδο. Εάν η κατανομή των ερωτημάτων αλλάξει σε 90/10, τότε ο μέσος αριθμός των tree nodes μειώνεται σημαντικά. Ο μέσος αριθμός των tree nodes για το φορτίο ερωτημάτων που είναι πολωμένο προς το ℓ_0 παραμένει χαμηλός. Ο μέσος αριθμός των pivot keys (όπως φαίνεται στο Σχήμα 3.18) παρουσιάζει μία ανάλογη συμπεριφορά με αυτήν που παρατηρήθηκε για τα tree nodes για όλα τα φορτία δεδομένων. Ο αριθμός των pivot keys συσχετίζεται με τον όγκο της στατιστικής πληροφορίας που διατηρείται και με τον αριθμό των root δεικτών. Για κάθε pivot



Σχήμα 3.17: *Μέσος αριθμός των tree nodes για κάθε κόμβο*

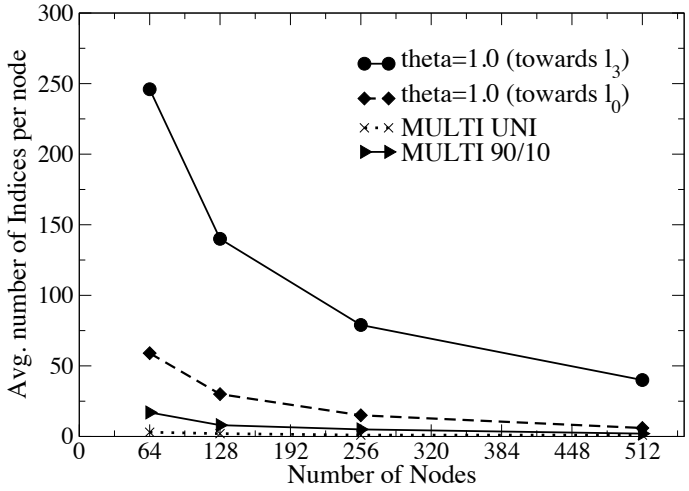
key, διατηρείται μία πλειάδα με L επίπεδα για την αποθήκευση της στατιστικής πληροφορίας. Επιπλέον, εάν ο μέσος ρυθμός των εισερχόμενων ερωτημάτων υπολογίζεται με τη μέθοδο του sliding window με *n* slots, ο αριθμός των αντίστοιχων πλειάδων πρέπει να πολλαπλασιαστεί με αυτόν τον παράγοντα. Επίσης, ο αριθμός των υπαρχόντων pivot keys είναι και αυτός ίσος με τον αριθμό των root indices και αυτό δικαιολογείται αν ληφθεί υπόψη ότι ένας ξεχωριστός δείκτης διατηρείται για κάθε pivot key.



Σχήμα 3.18: *Μέσος αριθμός των pivot keys για κάθε κόμβο*

Η αξιολόγηση του μέσου αριθμού των δεικτών απεικονίζεται στο Σχήμα 3.19 και επιβεβαιώνει το γεγονός ότι όσο πιο αποτελεσματικές είναι οι λειτουργίες re-indexing, τόσο λιγότεροι δείκτες δημιουργούνται. Το φορτίο ερωτημάτων *MULTI* απαιτεί τους λιγότερους δείκτες, γιατί τα

δέντρα προσαρμόζουν τα pivot levels τους κατάλληλα. Ο μέσος αριθμός των soft-state δεικτών είναι μεγαλύτερος για φορτία ερωτημάτων που είναι πολωμένα προς το ℓ_3 και αυτό οφείλεται σε διάφορους παράγοντες. Πολλά δέντρα εκτελούν λειτουργίες drill-down σε χαμηλότερα επίπεδα της ιεραρχίας και οι πλειάδες είναι περισσότερο διάσπαρτες στους κόμβους της επικάλυψης, άρα μία δεικτοδοτημένη τιμή χρησιμοποιεί περισσότερους δείκτες. Επιπλέον, ο αριθμός των διαφορετικών τιμών είναι μεγαλύτερος και το γεγονός αυτό αυξάνει την πιθανότητα να ερωτηθεί μία μη δεικτοδοτημένη τιμή που θα έχει ως επακόλουθο τη δημιουργία ενός νέου δείκτη. Τα αντίθετα συμπεράσματα ισχύουν για τα φορτία ερωτημάτων που κατευθύνονται προς το ℓ_0.

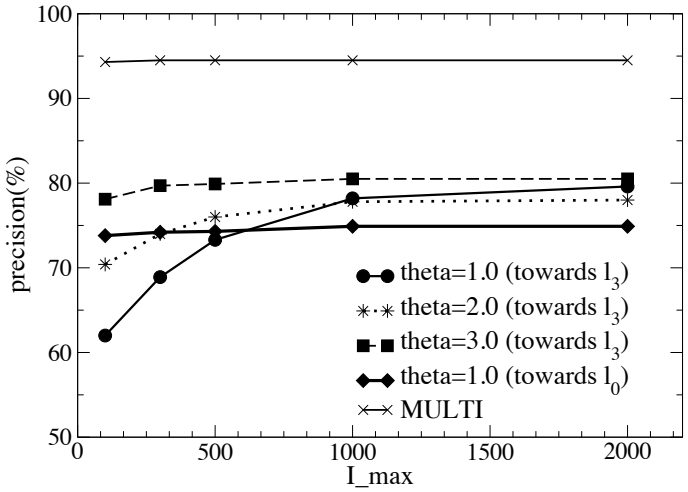


Σχήμα 3.19: *Μέσος αριθμός των soft-state δεικτών ανά κόμβο*

Το precision του συστήματος είναι σταθερό με μία μέση διακύμανση γύρω στο 1% ανεξάρτητα από τον αριθμό των κόμβων που συμμετέχουν στη DHT επικάλυψη. Επιπλέον, ο αριθμός των κόμβων δεν επηρεάζει το συνολικό αριθμό των pivot keys και των tree nodes, δεδομένου ότι οι τιμές αυτές εξαρτώνται κυρίως από την κατανομή του φορτίου ερωτημάτων. Όμως, ο αριθμός των soft-state δεικτών επηρεάζεται από τον αριθμό των κόμβων του συστήματος: όσο λιγότεροι είναι οι κόμβοι, τόσο αυξάνεται η πιθανότητα ο ίδιος κόμβος να αποθηκεύει περισσότερα από ένα δέντρα που περιέχουν την τιμή που δεικτοδοτείται και για αυτό το λόγο δημιουργείται μόνο ένας δείκτης. Η κατανομή των αναφερόμενων αντικειμένων στους κόμβους είναι καλύτερη όταν ο αριθμός των κόμβων αυξάνει. Σύμφωνα με τα πειραματικά αυτά αποτελέσματα, όταν περισσότεροι κόμβοι συμμετέχουν στην επικάλυψη, τότε κάθε κόμβος είναι υπεύθυνος για ένα μικρότερο μέσο αριθμό των υπό συζήτηση δεδομένων.

3.7.6 Μελέτη της Επίδρασης της Παραμέτρου I_{max}

Η τιμή της παραμέτρου I_{max} μπορεί να ρυθμιστεί σε κάθε κόμβο και ορίζει το μέγιστο αριθμό των soft-state δεικτών που μπορεί να αποθηκεύσει ένας κόμβος. Η δημιουργία ενός καινούργιου δείκτη οδηγεί στη διαγραφή του παλαιότερου (δηλαδή του δείκτη που δεν έχει χρησιμοποιηθεί για το μεγαλύτερο χρονικό διάστημα), εάν ο αριθμός των δεικτών υπερβεί το I_{max}. Ο στόχος της συγκεκριμένης στρατηγικής είναι ο εξής: εάν οι δείκτες που πρόκειται να διαγραφούν είναι χρήσιμοι, τότε ο χρόνος τους θα είχε ανανεωθεί. Συνεπώς η διαγραφή τους δεν θα έχει αρνητική επίδραση στην απόδοση του συστήματος.



Σχήμα 3.20: *Το precision για διάφορα UNI φορτία ερωτημάτων*

Η τιμή του I_{max} έχει διαφορετική επίδραση στην αποδοτικότητα του συστήματος, η οποία εξαρτάται από τον τύπο του φορτίου ερωτημάτων. Η χρήση των δεικτών αυξάνει όταν το φορτίο ερωτημάτων δεν υπαγορεύει στο σύστημα μία συγκεκριμένη κατεύθυνση για να εκτελεστούν οι λειτουργίες re-indexing, δηλαδή όταν περισσότερα από ένα επίπεδα είναι εξίσου δημοφιλή. Όσο πιο ομοιόμορφη είναι η κατανομή που ακολουθούν τα ερωτήματα, τόσο περισσότεροι δείκτες δημιουργούνται.

Το Σχήμα 3.20 δείχνει το precision που επιτυγχάνεται για διάφορες τιμές του θ, όταν η πόλωση των ερωτημάτων κατευθύνεται προς τα επίπεδα ℓ_0 και ℓ_3 αντίστοιχα και τα ερωτήματα στοχεύουν ομοιόμορφα τις τιμές σε κάθε επίπεδο. Το φορτίο που ευνοεί τα υψηλότερα επίπεδα δε φαίνεται να επηρεάζεται από τη μεταβολή του I_{max}, αν και το *valueDist* που χρησιμοποιείται ακολουθεί ομοιόμορφη κατανομή, που αποτελεί και την πιο απαιτητική περίπτωση. Αυτό το αποτέλεσμα οφείλεται στο γεγονός ότι σε αυτήν την περίπτωση χρησιμοποιούνται κυρίως οι root δείκτες, οι οποίοι δε συμπεριλαμβάνονται στο κριτήριο του I_{max}. Επιπλέον, οι πιθανές τιμές των

υψηλότερων επιπέδων είναι λιγότερες και για αυτό το λόγο απαιτείται λιγότερος αποθηκευτικός χώρος για τη δημιουργία των soft-state δεικτών. Στην αντίθετη περίπτωση, όπου τα φορτία ερωτημάτων κατευθύνονται προς το ℓ_3, η αύξηση της τιμής του I_{max} μπορεί να επηρεάσει το precision μέχρι 10%, ειδικά στο φορτίο όπου $\theta = 1.0$, σύμφωνα με το οποίο ρωτιούνται τιμές από σχεδόν όλα τα επίπεδα και οι λειτουργίες re-indexing δεν επαρκούν να καλύψουν τη ζήτηση για τις τιμές όλων των επιπέδων. Το φορτίο ερωτημάτων *MULTI* αποτελεί το φορτίο εκείνο για το οποίο οι αλγόριθμοι για το re-indexing ευνοούν περισσότερο την αποδοτικότητα του συστήματος. Σε αυτήν την περίπτωση, οι αποφάσεις για re-indexing είναι σύμφωνες με τις τάσεις στα ερωτήματα και το precision εξαρτάται κυρίως από τη διόρθωση των pivot levels. Συνεπώς, η μεταβολή στις τιμές του I_{max} δεν επηρεάζει το precision που επιτυγχάνεται.

3.7.7 Μελέτη της Επίδρασης του Αριθμού των Επιπέδων των Ιεραρχιών

Ο αριθμός των επιπέδων μπορεί επίσης να επηρεάσει το precision του συστήματος. Ο λόγος που συμβαίνει αυτό είναι διπλός. Η αύξηση των επιπέδων οδηγεί σε περισσότερα υποψήφια επίπεδα προς τα οποία μπορεί να εκτελεστεί κάποιο re-indexing και γίνεται πιθανότερη η επίλυση ενός ερωτήματος με τη χρήση soft-state δεικτών ή με flooding. Σε αυτό το πείραμα χρησιμοποιήθηκαν ιεραρχίες με τέσσερα, έξι και οχτώ επίπεδα αντίστοιχα. Ο αριθμός των εισερχόμενων πλειάδων παρέμεινε 100k και επιλέχθηκε το ℓ_1 ως pivot level κατά τις αρχικές εισαγωγές των πλειάδων. Στοχεύοντας στη διατήρηση ενός σταθερού αριθμού συνολικών πλειάδων, τα πλάτη των δέντρων σε κάθε επίπεδο περιορίστηκαν. Η κατανομή των φορτίων ερωτημάτων, που περιλαμβάνουν περίπου 50k ερωτήματα, παράγεται επιλέγοντας το $\theta = 1.0$ για το *levelDist* και μία πολωμένη κατανομή 90/10 για το *valueDist*. Τα αποτελέσματα που παρουσιάζονται αναφέρονται σε φορτία ερωτημάτων που είναι πολωμένα προς τα υψηλότερα και τα χαμηλότερα επίπεδα.

Πίνακας 3.2: *Το precision για διαφορετικό αριθμό επιπέδων της ιεραρχίας*

Levels	towards l_0 (%)	towards l_3 (%)
4	86	91
6	80	77
8	78	77

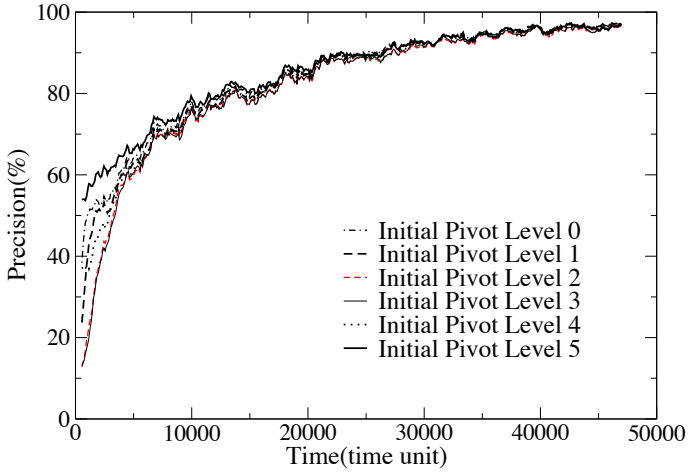
Το precision για τα φορτία ερωτημάτων που περιγράφηκαν καταγράφεται στον Πίνακα 3.2. Η αύξηση του αριθμού των επιπέδων έχει ως αποτέλεσμα τη μείωση του precision και για τις δύο κατηγορίες των φορτίων ερωτημάτων, όπως ήταν αναμενόμενο. Επίσης, παρατηρείται μία σημαντική μείωση του αριθμού των λειτουργιών roll-up στις μετρήσεις για τα φορτία ερωτημάτων που κατευθύνονται προς το ℓ_0 και μία μείωση των αποφάσεων για drill-down για τα φορτία

ερωτημάτων που κατευθύνονται προς το ℓ_3. Επιπλέον, και για τις δύο κατηγορίες εμφανίζεται μία τάση αύξησης των λειτουργιών re-indexing προς την αντίθετη κατεύθυνση σε σύγκριση με την πόλωση του φορτίου. Η συμπεριφορά αυτή μπορεί να δικαιολογηθεί από το γεγονός ότι υπάρχουν περισσότερα επίπεδα και οι αποφάσεις για re-indexing μπορεί να είναι διφορούμενες και να χρειάζονται κάποιες επιπλέον διορθωτικές λειτουργίες re-indexing, ώστε το σύστημα να προσαρμοστεί στα εισερχόμενα ερωτήματα. Επιπλέον, παρατηρείται ότι αυξάνεται ο αριθμός των ερωτημάτων που επιλύονται με τη χρήση soft-state δεικτών, ειδικά για φορτία ερωτημάτων που κατευθύνονται προς το ℓ_0. Η διαφορά στο καταγεγραμμένο precision είναι περισσότερο αξιοσημείωτη για την αντίθετη περίπτωση πόλωσης. Ο περιορισμένος αριθμός των διακριτών τιμών στα υψηλότερα επίπεδα ευνοεί την επαναχρησιμοποίηση των δεικτών. Όπως φαίνεται στον Πίνακα 3.2, η μείωση στο precision είναι μικρότερη όταν τα ερωτήματα είναι πολωμένα προς το ℓ_0.

3.7.8 Μελέτη της Απόδοσης του Συστήματος για Δεδομένα του APB bennchmark

Η προσαρμοστικότητα του συστήματος δοκιμάζεται επίσης και για μία άλλη κατηγορία πραγματικών δεδομένων. Για αυτό το λόγο, χρησιμοποιήθηκαν φορτία ερωτημάτων που προέρχονται από το APB-1 benchmark [apb]. Το APB-1 δημιουργεί μία δομή βάσης δεδομένων με πολλαπλές διαστάσεις και παράγει ένα σύνολο από επιχειρησιακές λειτουργίες που αντιπροσωπεύουν τη λειτουργικότητα των OLAP εφαρμογών. Για τα πειράματα που εκτελέστηκαν, χρησιμοποιείται η διάσταση του `product`, δηλαδή μια ιεραρχία με έξι επίπεδα (το τελευταίο επίπεδο της ιεραρχίας περιέχει το 90% των μελών της). Ειδικότερα, ο αριθμός των διαφορετικών τιμών κάθε επιπέδου είναι $|\ell_0 = 50|, |\ell_1 = 150|, |\ell_2 = 800|, |\ell_3 = 3050|, |\ell_4 = 6950|$ and $|\ell_5 = 93050|$. Ένα άλλο χαρακτηριστικό του συνόλου δεδομένων είναι ότι ο αριθμός των τιμών για κάθε κόμβο στο επόμενο επίπεδο δεν είναι σταθερός, όπως στα προηγούμενα σύνολα δεδομένων που παράχθηκαν με τη χρήση γεννήτριας. Το φορτίο των ερωτημάτων είναι πολωμένο προς τα χαμηλότερα επίπεδα της ιεραρχίας και το 75% των ερωτημάτων είναι για τιμές του ℓ_4 και του ℓ_5.

Τα πειραματικά αποτελέσματα απεικονίζονται στο Σχήμα 3.21, όπου φαίνεται το precision κατά τη διάρκεια του χρόνου για διαφορετικά αρχικά pivot levels. Αξίζει να σημειωθεί ότι το σύστημα προσαρμόζεται στις απαιτήσεις των φορτίων ερωτημάτων και η απόδοση του είναι παρεμφερής ανεξάρτητα από το επίπεδο που επιλέχθηκε να χρησιμοποιείται ως pivot level κατά τις αρχικές εισαγωγές των πλειάδων. Το συμπέρασμα είναι ότι οι λειτουργίες re-indexing, ειδικότερα οι λειτουργίες drill-down προς το ℓ_4 και το ℓ_5, και οι soft-state δείκτες καταφέρνουν ώστε το σύστημα να αυξήσει το precision, το οποίο φτάνει κοντά στο 100%.



Σχήμα 3.21: *Το precision κατά τη διάρκεια του χρόνου για τα APB δεδομένα για διαφορετικά αρχικά pivot levels*

3.7.9 Ενημερώσεις

Για τη μέτρηση του κόστους των σταδιακών ενημερώσεων των δεδομένων που χρησιμοποιήθηκαν στα πειράματα, εφαρμόστηκε το εξής σενάριο: Αρχικά επιλέχθηκε το 90% των πλειάδων και εκτελέστηκαν τα φορτία ερωτημάτων που περιγράφονται στο 3.7.2. Μετά την εκτέλεση των ερωτημάτων, εκτελέστηκαν ενημερώσεις για την εισαγωγή των υπόλοιπων πλειάδων, που αντιστοιχούν στο 10% του συνόλου δεδομένων. Σε αυτό το σημείο πρέπει να επισημανθεί ότι το φορτίο ερωτημάτων παίζει σημαντικό ρόλο για τον υπολογισμό του κόστος των ενημερώσεων, καθώς επηρεάζει το επίπεδο δεικτοδότησης των αποθηκευμένων πλειάδων. Το κόστος σε μηνύματα για την εισαγωγή μίας πλειάδας είναι ένα lookup μήνυμα για την εύρεση του root key και ένα insertion μήνυμα για την αποθήκευση της πλειάδας. Οι αρχικές εισαγωγές των πλειάδων δεν απαιτούν την αποστολή επιπρόσθετων lookup μηνυμάτων για την ανανέωση των δεικτών, δεδομένου ότι δεν υπάρχουν άλλοι δείκτες εκτός από αυτούς μεταξύ των root keys και των αντίστοιχων pivot keys τους. Το επιλεγμένο pivot level για τις αρχικές πλειάδες είναι το ℓ_1. Τα πειράματα που πραγματοποιήθηκαν αφορούν τον υπολογισμό του κόστους των ενημερώσεων όσον αφορά τα επιπρόσθετα lookups για την ενημέρωση των δημιουργημένων δεικτών για καινούργιες πλειάδες.

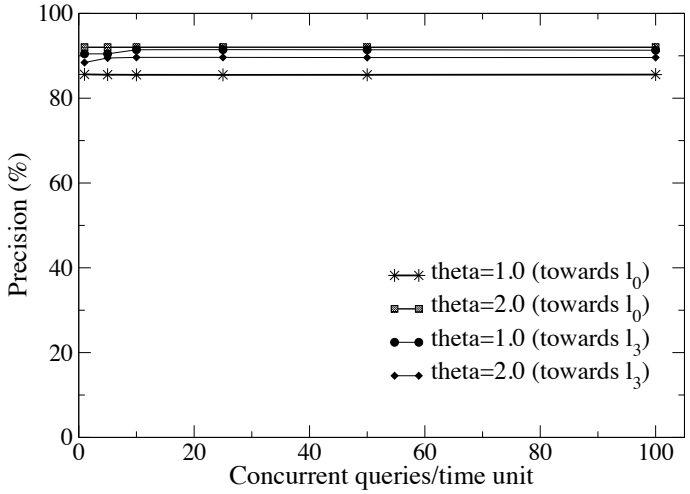
Ο Πίνακας 3.3 περιλαμβάνει το μέσο αριθμό των lookups ανά εισαγωγή για την ενημέρωση των soft-state δεικτών. Το κόστος αυτό μπορεί να θεωρηθεί αμελητέο όταν η πόλωση είναι προς τα υψηλότερα επίπεδα της ιεραρχίας. Το μέγιστο κόστος που ισούται με 2 καταγράφηκε για πολωμένα φορτία προς το ℓ_3 και ομοιόμορφο *valueDist*. Όσο λιγότερο πολωμένη είναι η κατανομή, τόσο αυξάνεται η πιθανότητα της ύπαρξης soft-state δεικτών σε επίπεδα εκτός αυτού που είναι το δημοφιλέστερο. Επίσης, όσο αυξάνεται η πόλωση, αυτό το κόστος μειώνεται.

Πίνακας 3.3: *Μέσος αριθμός lookup μηνυμάτων για την ενημέρωση των δεικτών μετά την εισαγωγή κάθε πλειάδας για διάφορα* valueDist

valueDist	***Skew towards*** ℓ_0		***Skew towards*** ℓ_3	
	$\theta = 1.0$	$\theta = 2.0$	$\theta = 1.0$	$\theta = 2.0$
uniform	≈ 0	≈ 0	1.99	1.98
80/20	0.01	≈ 0	1.8	1.16
90/10	0.14	≈ 0	0.39	0.25
99/1	≈ 0	≈ 0	0.01	0.02

3.7.10 Συμπληρωματικά πειραματικά αποτελέσματα

Στη συνέχεια περιγράφονται εν συντομία κάποια πρόσθετα πειραματικά αποτελέσματα.Ένα από τα πειράματα αυτά αφορά τη συμπεριφορά του συστήματος όταν μεταβάλλεται ο αριθμός των ταυτόχρονων ερωτημάτων ανά time unit για τα φορτία ερωτημάτων που περιγράφηκαν. Σε αυτό το πείραμα, το ερωτήματα των αρχικών φορτίων που παρουσιάστηκαν στην Ενότητα 3.7.2 χωρίζονται σε ομάδες ερωτημάτων. Η κάθε ομάδα ξεκινάει να εκτελείται κατά την έναρξη του time unit. Τα αποτελέσματα που προέκυψαν για φορτία ερωτημάτων που είναι πολωμένα προς το ℓ_0 και το ℓ_3 φαίνονται στο Σχήμα 3.22. Το precision που μετρήθηκε παρουσιάζει αμελητέα απόκλιση και για αυτό το λόγο το σύστημα παραμένει ιδιαίτερα αποδοτικό στην επίλυση των ερωτημάτων, ακόμα και αν εξυπηρετεί έναν μεγάλο αριθμό ταυτόχρονων χρηστών.



Σχήμα 3.22: *Το precision για ταυτόχρονα ερωτήματα όπου το valueDist είναι 90/10*

Επιπλέον, τα πειράματα που εκτελέστηκαν μέχρι και 1k κόμβους έδειξαν μικρές ποιοτικές διαφορές. Τα αποτελέσματα των εξομοιώσεων για διαφορετικές τιμές του threshold έδειξαν ότι

η διακύμανση του precision για $0,2 \leq thr \leq 0,4$ είναι το πολύ 2%. Όταν το $thr \geq 0,5$ και οι κατανομές των φορτίων είναι ομοιόμορφες, η διακύμανση του precision φτάνει μέχρι 10%. Για $thr \leq 0,3$, πρέπει να αυξηθεί ο αρχικός αριθμός των ερωτημάτων που πρέπει να συμπληρωθεί για να επιτραπούν οι λειτουργίες re-indexing, ώστε να μειωθεί η εκτέλεση πλεοναζόντων λειτουργιών. Τα πειράματα για κατανομές δεδομένων που διαφέρουν ως προς τον αριθμό διαφορετικών τιμών ανά επίπεδο έδειξαν επίσης μικρές ποιοτικές διαφορές. Μία άλλη σημαντική παρατήρηση είναι ότι η επιλογή διαφορετικών επιπέδων ως αρχικά pivot levels δεν επηρεάζει την απόδοση του συστήματος κατά τη φάση της σταθερής λειτουργίας του, αφού οι λειτουργίες re-indexing και οι soft-state δείκτες προσαρμόζουν κατάλληλα τις δομές δεικτοδότησης τους συστήματος.

3.8 Ένα Κατανεμημένο Grid Information System

Ένα σενάριο χρήσης που υποστηρίζει τη χρησιμότητα του προτεινόμενου συστήματος μπορεί να θεωρηθεί για τη συλλογή και διαχείριση πληροφορίας που παράγεται από Information Services σε Grid περιβάλλοντα. Οι υπολογιστικοί πόροι και οι υπηρεσίες σε Grid υποδομές διαφημίζουν ένα μεγάλο όγκο δεδομένων, τα οποία χρησιμοποιούνται από διάφορους χρήστες που ανήκουν σε πολλαπλούς διαχειριστικούς τομείς. Η πληροφορία αυτή δεν προορίζεται μόνο για τον χειρισμό γεγονότων (event handling), όπως στα παραδοσιακά συστήματα παρακολούθησης (monitoring systems) στα δίκτυα υπολογιστών και στις υπολογιστικές συστοιχίες. Η πληροφορία που παράγεται από διάφορους μηχανισμούς όπως τα monitoring systems για συστοιχίες υπολογιστών (Ganglia [Gan], Hawkey [Haw]), διάφορες υπηρεσίες (GRAM, RLS [glo]), συστήματα ουρών, κλπ., οργανώνεται και παρέχεται σε διάφορες εφαρμογές, όπως λογιστικά συστήματα (accounting systems), χρονοδρομολογητές, δικτυακές πύλες (portals), κλπ. Για αυτό το λόγο, το βασικό ζήτημα κατά το σχεδιασμό ενός Grid Information Service είναι η αναγνώριση της πληροφορίας που απαιτείται και ο προσδιορισμός του καλύτερου τρόπου για να γίνει διαθέσιμη.

Στις σύγχρονες αρχιτεκτονικές για Grid Monitoring (MDS, R-GMA), τα δεδομένα που αφορούν την κατάσταση της υποδομής συλλέγονται από ένα συνδυασμό monitoring συστημάτων από κάθε πόρο και οργανώνονται από τους "παραγωγούς της πληροφορίας" (*information producers* ή *information providers*). Στις αρχικές γενιές των monitoring αρχιτεκτονικών, οι information producers οργανώνονταν σε ιεραρχικές δομές και δημοσίευαν τα δεδομένα τους, τα οποία αποθηκεύονταν εν τέλει σε βάσεις δεδομένων τύπου LDAP μετά τη συλλογή τους. Στις πιο μοντέρνες εφαρμογές, οι information producers είναι γνωστοί στο σύστημα, αφού εγγράφονται σε ένα *Index service* (MDS) ή ένα *Registry* (R-GMA). Οι καταναλωτές της πληροφορίας ρωτάνε τη δομή αυτή για τον εντοπισμό των producers και επικοινωνούν με όλους αυτούς για να αποκτήσουν τα δεδομένα που χρειάζονται. Επιπροσθέτως, υπάρχουν διάφορες υπηρεσίες σύνοψης (aggregator services) που συλλέγουν πληροφορία από τους information producers με τη χρήση μηχανισμών που προσδιορίζουν τον τύπο των δεδομένων και τη συλλεγόμενη πληροφορία. Για παράδειγμα, τα VOs διατηρούν τέτοιες υπηρεσίες για τη συλλογή πληροφορίας για τους πόρους του VO από τους information producers που υπάρχουν σε διάφορες συστοιχίες (sites). Λόγω του μεγάλου όγκου των δεδομένων και της χρησιμότητας τους σε πολλαπλές υπηρεσίες, θεωρώ πως είναι σημαντική η ύπαρξη μία αποδοτικής λύσης για την αποθήκευση και τη δεικτοδότηση της παραγόμενης πληροφορίας, η οποία καθιστά δυνατή τη δυναμική προσαρμογή του συστήματος στα αιτήματα των χρηστών και των υπηρεσιών και συνεισφέρει στη διαλειτουργικότητα και την αποδοτικότητα της Grid υποδομής.

Στη βιβλιογραφία υπάρχουν διάφορα συστήματα και αρχιτεκτονικές που προτείνονται ως λύσεις για information systems. Η λύση που χρησιμοποιείται ευρέως είναι το *Globus Monitoring*

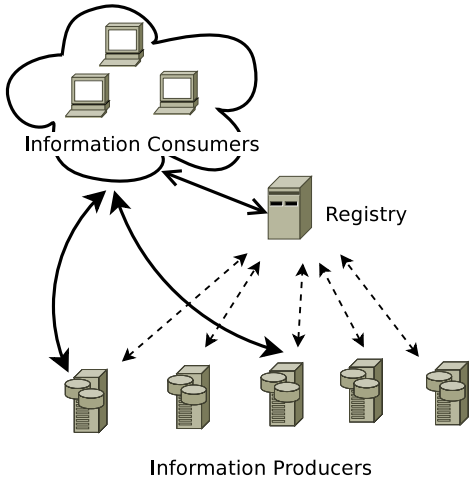
and Discovery Service (MDS) [MDS]. Ένα *Grid Index Information Service (GIIS)* παρέχει ένα συνοπτικό κατάλογο για δεδομένα χαμηλότερου επιπέδου που αποθηκεύονται σε πολλαπλά *Grid Resource Information Services (GRISs)*. Διάφορα GIISs μπορούν να συνδυαστούν σε μία ιεραρχική δομή που επιτρέπει τη συνολική ανάκτηση της πληροφορίας με την επερώτηση του GIIS που βρίσκεται στο υψηλότερο επίπεδο (top level GIIS). Ωστόσο, το MDS δεν αποτελεί μία επεκτάσιμη λύση: Πολλαπλά αιτήματα από τους clients μπορούν να οδηγήσουν στην υπερφόρτωση του top level GIIS [ZFS07]. Ένα άλλο σύστημα που χρησιμοποιείται κυρίως για το accounting και τη δημοσίευση πληροφορίας που αφορά τους χρήστες είναι το *Relational Grid Monitoring and Discovery Service (R-GMA)* [RGM], το οποίο υποστηρίζει πολύπλοκους τύπους ερωτημάτων που συναντώνται σε σχεσιακές βάσεις δεδομένων. Το R-GMA παρουσιάζει την πληροφορία σαν να είναι αποθηκευμένη σε μία εικονική βάση δεδομένων που αποτελείται από ένα σύνολο εικονικών πινάκων. Όμως, μεγάλοι όγκοι δεδομένων πρέπει να μεταφερθούν offline σε μία κεντρική βάση δεδομένων με ότι μειονεκτήματα μπορεί αυτό να επιφέρει στην απόδοση του συστήματος. Το βασικό μειονέκτημα σε όλες αυτές τις μεθόδους είναι ότι καμία από αυτές τις αρχιτεκτονικές δεν είναι επεκτάσιμη όταν αυξάνεται ο αριθμός των data collectors [ZFS07]. Επιπρόσθετα, όλες αυτές οι αρχιτεκτονικές προϋποθέτουν τη μεταφορά των δεδομένων (data migration) με offline τρόπο (ή ανά τακτά χρονικά διαστήματα) σε μία κεντρική τοποθεσία, όπου η συνολική πληροφορία μπορεί να γίνει διαθέσιμη. Σε αντίθεση, η προτεινόμενη λύση αποτελεί ένα εντελώς αποκεντρωμένο σύστημα, όπου όλα τα δεδομένα είναι συνεχώς διαθέσιμα και δεικτοδοτούνται στο επίπεδο λεπτομέρειας που επιτρέπει τη γρήγορη επίλυση των ερωτημάτων.

Γενικώς, οι αρχιτεκτονικές τέτοιων πληροφοριακών συστημάτων μπορούν να χωριστούν στις ακόλουθες δομικές κατηγορίες σύμφωνα με τις υπάρχουσες τεχνολογίες και τα παραδείγματα που προέρχονται από υπάρχουσες υποδομές:

- Κεντρικοποιημένη λύση (Centralized solution): Η παραγόμενη πληροφορία από διάφορα εργαλεία για monitoring δημοσιεύεται σε μία κεντρική βάση δεδομένων. Οι χρήστες θέτουν τα ερωτήματα τους σε αυτή τη βάση δεδομένων για να ανακτήσουν την πληροφορία που τους ενδιαφέρει. Μία συνηθισμένη τεχνική είναι η δημιουργία ακριβών αντιγράφων της βάσης δεδομένων ή η διατήρηση μίας τέτοιας βάσης για κάθε VO, ώστε να αποφευχθούν φαινόμενα ιδιαίτερα υψηλού φορτίου και μεμονωμένα σημεία αποτυχίας (single points of failures).

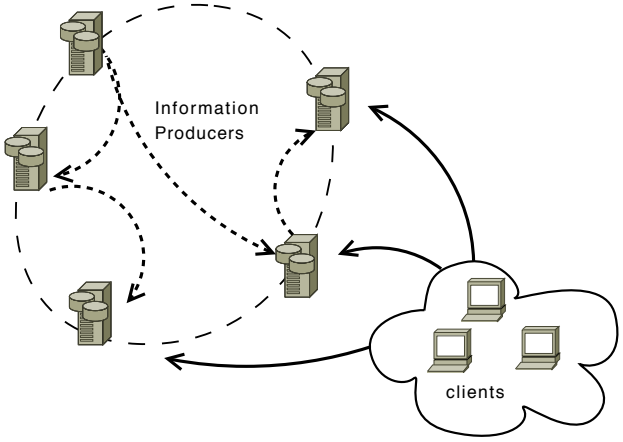
- Κατανεμημένη λύση (Distributed solution): Σε αυτήν την περίπτωση υλοποιείται ένα μοντέλο producer–consumer με κατανεμημένο τρόπο ώστε να δημιουργηθεί μία εικονική βάση δεδομένων. Οι producers εγγράφονται σε ένα Κατάλογο (Registry) και περιγράφουν το τύπο και τη δομή της παραγόμενης πληροφορίας. Οι consumers επικοινωνούν με το Registry για την εύρεση των κατάλληλων producers και στη συνέχεια επικοινωνούν

με αυτούς για την ανάκτηση των σχετικών δεδομένων, όπως φαίνεται στο Σχήμα 3.23 (*distributedCP* model).



Σχήμα 3.23: *Απεικόνιση ενός IS που αντιπροσωπεύει το μοντέλο του distributedCP*

Το περιγραφόμενο σύστημα αποτελεί μία ολοκληρωμένη λύση για την οργάνωση και την αποθήκευση τέτοιου είδους πληροφορίας. Η προτεινόμενη διάταξη μπορεί να ενσωματωθεί σε ένα Grid περιβάλλον και να οδηγήσει σε μία πλήρως κατανεμημένη λύση. Σε αυτήν την αρχιτεκτονική, οι κόμβοι με τους information producers και αυτοί που σχετίζονται με τη διαχείριση της πληροφορίας οργανώνονται σε μία DHT επικάλυψη, όπως φαίνεται και στο Σχήμα 3.24. Η δρομολόγηση των μηνυμάτων στους κόμβους της επικάλυψης διεκπεραιώνεται σύμφωνα με το DHT πρωτόκολλο που χρησιμοποιείται και δεν απαιτείται κάποια κεντρική δομή. Η χρήση της P2P επικάλυψης οδηγεί σε μία επεκτάσιμη λύση, ενώ επιτυγχάνεται εξισορρόπηση του φόρτου δεδομένων και ερωτημάτων. Για τη συγκεκριμένη εφαρμογή ορίζεται και μία εννοιολογική ιεραρχία με διάφορα επίπεδα σύνοψης που χαρακτηρίζει τα παραγόμενα δεδομένα. Έτσι, κάθε numerical fact περιγράφεται από την εννοιολογική ιεραρχία του. Επίσης, δεν απαιτείται offline συλλογή και επεξεργασία των δεδομένων, αφού υποστηρίζονται οι διαδικασίες των online ενημερώσεων. Ο μηχανισμός re-indexing επιτρέπει τη σύνοψη της πληροφορίας σε διαφορετικά επίπεδα λεπτομέρειας ανάλογα με τις τάσεις στα εισερχόμενα ερωτήματα. Επιπλέον, το Registry ή το Index Service που περιέχει τις τοποθεσίες των υποστάσεων που εκτελούν τις αντίστοιχες υπηρεσίες υλοποιείται με κατανεμημένο τρόπο Η σχεδίαση αυτή προσδίδει την ιδιότητα της αυτοοργάνωσης στο σύστημα και διαφοροποιείται από τις υπάρχουσες κατανεμημένες προσεγγίσεις, όπου ένας consumer πρέπει να απευθυνθεί σε έναν κεντρικό κατάλογο για να ανακαλύψει τους producers. Τα δεδομένα συνοψίζονται δυναμικά και αποθηκεύονται με κατανεμημένο τρόπο σύμφωνα με τις προτιμήσεις των χρηστών και με τον τρόπο αυτό επιτυγχάνεται η μείωση του κόστους επεξεργασίας και επικοινωνίας.



Σχήμα 3.24: *Η προτεινόμενη αρχιτεκτονική για ένα IS σύστημα*

Ένα παράδειγμα χρήσης της προτεινόμενης αρχιτεκτονικής για το IS μπορεί να αποτελέσει η υιοθέτηση της για την υπηρεσία που παρέχει πληροφορία για accounting σκοπούς στο EGEE Accounting Portal [ege]. Η υπηρεσία αυτή χρησιμοποιεί το [et.05], που είναι μία εφαρμογή για την επεξεργασία των δεδομένων που παράγονται σε κάθε site. Στη συνέχεια, οι R-GMA producers συλλέγουν τα δεδομένα και τα υποβάλλουν σε μία κεντρική βάση. Η βάση αυτή συλλέγει εκατομμύρια εγγραφών και τα αποθηκεύει offline. Λόγω του τεράστιου όγκου των δεδομένων, μόνο κάποιες συνοπτικές όψεις των διάφορων μετρικών, όπως Number of jobs, Normalized CPU usage, SumCPU, CPU efficiency υπολογίζονται offline και γίνονται διαθέσιμες στους χρήστες. Ο χειρισμός αυτός τονίζει το γεγονός ότι στις κεντρικοποιημένες προσεγγίσεις πρέπει να λαμβάνεται υπόψη η ποσότητα της πληροφορίας που φυλάσσεται αλλά και το κόστος που απαιτείται για τη φύλαξη της. Η απλή εννοιολογική ιεραρχία για το VO που χρησιμοποιήθηκε στα προηγούμενα παραδείγματα θα μπορούσε να χρησιμοποιηθεί και για τη συγκεκριμένη εφαρμογή. Όμως, στην Ενότητα 3.9, περιγράφεται μία πιο λεπτομερής εννοιολογική ιεραρχία που χρησιμοποιείται και στα πειράματα για το συγκεκριμένο σενάριο χρήσης, βάσει της οποίας γίνεται η κατανομή των δεδομένων στους κόμβους της επικάλυψης.

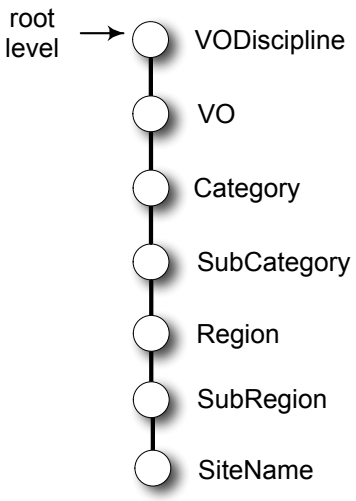
3.9 Αξιολόγηση του Προτεινόμενου Grid Information System

Το προτεινόμενο σύστημα δοκιμάστηκε για το σενάριο χρήσης του ως ένα Grid IS και συγκρίθηκε με κάποιες υπάρχουσες λύσεις. Στο ακόλουθο πείραμα θεωρήθηκε ότι η οργάνωση των δεδομένων και η λειτουργικότητα που παρέχεται από το EGEE Accounting Portal που αναπτύχθηκε από το CESGA [ege], ενώ χρησιμοποιήθηκαν δεδομένα και ερωτήματα από την πραγματική χρήση του συγκεκριμένου συστήματος. Το συγκεκριμένο portal συγκεντρώνει δεδομένα

από όλα τα sites που συμμετέχουν στις υποδομές του EGEE και του WLCG αλλά και από άλλα sites που ανήκουν σε Grid οργανισμούς που συνεργάζονται με την υποδομή του EGEE. Τα δεδομένα αναλύονται περαιτέρω για τη δημιουργία στατιστικών περιλήψεων που παρουσιάζονται στους χρήστες. Τα numerical facts που ενδιαφέρουν τους χρήστες είναι διάφορες μετρικές όπως Normalized CPU time, Number of Jobs, κλπ. και οι τιμές για αυτά συλλέγονται από διάφορα monitoring εργαλεία από τους grid πόρους. Η μέθοδος που ακολουθείται στο συγκεκριμένο portal είναι ότι εκτελείται περαιτέρω επεξεργασία των συγκεντρωμένων δεδομένων και στη συνέχεια εισάγονται σε μία offline βάση δεδομένων. Ο μεγάλος όγκος των τιμών που συλλέγονται καθιστά μη εφικτή τη διαθεσιμότητα τους στους χρήστες. Για αυτό το λόγο, δημιουργούνται συνοπτικές όψεις και αποθηκεύονται σε μία κεντρική βάση ώστε να είναι εφικτή η παρουσίαση τους μέσω του portal στους χρήστες. Στην προτεινόμενη προσέγγιση υιοθετείται μία παρεμφερής αντίληψη. Το monitoring των πόρων και η συλλογή των τιμών δεν διερευνάται λεπτομερώς και θεωρείται ότι χρησιμοποιούνται κάποια "παραδοσιακά" grid εργαλεία (π.χ. APEL). Επίσης, γίνεται η υπόθεση ότι η επεξεργασία και δημοσίευση των numerical facts διάφορων μετρικών γίνεται από μία τοπική υπηρεσία του κάθε κόμβο. Στη συνέχεια, η πληροφορία αυτή μπορεί να αποθηκευτεί στους αντίστοιχους Information Servers που μπορεί να υπάρχουν σε κάθε site. Ο στόχος του συγκεκριμένου πειράματος είναι να δείξει ότι η πληροφορία που συλλέχθηκε σχετικά με τους grid πόρους μπορεί να οργανωθεί δυναμικά έτσι ώστε να προσαρμοστεί στο φορτίο των ερωτημάτων ευνοώντας της εξισορρόπηση του φόρτου εργασίας ανάμεσα στους κόμβους. Ο φόρτος των ερωτημάτων (query load) για κάθε Information Server συγκρίνεται με τη centralized και τη distributedCP λύση.

Η περιγραφή των αριθμητικών τιμών των μετρικών (numerical facts) ακολουθεί μία ιεραρχία εφτά επιπέδων που εξάχθηκε από τη δομή που χρησιμοποιείται για την παρουσίαση των δεδομένων στη συγκεκριμένη εφαρμογή. Τα επίπεδα της ιεραρχίας απεικονίζονται στο Σχήμα 3.25.

Η συγκεκριμένη ιεραρχία έχει χρησιμοποιηθεί για όλες τις κατηγορίες και τα προγράμματα, ώστε να διατηρηθεί μία κοινή και γενική περιγραφή. Για να εξασφαλιστεί ότι μία τιμή δεν ανήκει σε πολλαπλά δέντρα με διαφορετικές τιμές στο root level, έχει προηγηθεί επεξεργασία των δεδομένων και έχουν αντιστοιχηθεί με μοναδικό τρόπο σε ακέραιες τιμές λαμβάνοντας υπόψη τις τιμές όλων των επιπέδων ανάμεσα στο VO επίπεδό και στο υπό εξέταση επίπεδο. Η συγκεκριμένη υπόθεση μπορεί να αιτιολογηθεί, για παράδειγμα το Number of Jobs διαφέρει όταν τα EGEE sites που βρίσκονται στην Ελλάδα ρωτιούνται για δύο διαφορετικές τιμές. Επιπλέον, στις υπάρχουσες grid υποδομές αποτελεί κοινή πρακτική να συμπεριλαμβάνεται η τιμή για την ιδιότητα του VO στα ερωτήματα προς το Information Service. Σύμφωνα με την πληροφορία που παρέχεται από το accounting portal, ο συνολικός αριθμός των sites που συμμετέχουν σε διάφορες κατηγορίες και προγράμματα της υποδομής κυμαίνεται γύρω στα 700. Κάθε site θεωρείται ως ένας ξεχωριστός κόμβος στην εξομοίωση του προτεινόμενου συστήματος. Το μέγεθος του



Σχήμα 3.25: *Τα επίπεδα της εννοιλογικής ιεραρχίας που ορίστηκε για το Grid Information System.*

συνόλου των δεδομένων που εισήχθηκε στο σύστημα είναι 5798 και μόνο τα υποστηριζόμενα VOs από κάθε site λήφθηκαν υπόψη. Για παράδειγμα, οι πιθανές τιμές για ένα site που ανήκει στο *EGEE* μπορεί να είναι:

[High-Energy Physics, atlas, EGEE, Production, SouthEastern, Greece, HG-01-GRNET].

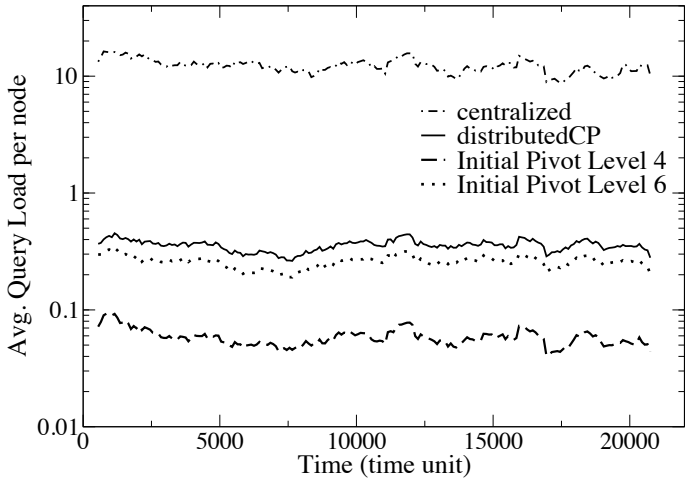
Παρόλα αυτά, κάποια από τα sites που ανήκουν σε άλλες κατηγορίες μπορεί να μην έχουν δηλωθεί με τρόπο τέτοιο ώστε να προσδιορίζονται οι τιμές για κάθε επίπεδο της ιεραρχίας. Σε αυτήν την περίπτωση, το αντίστοιχο επίπεδο λαμβάνει μία "ψευδοτιμή", η οποία δεν περιλαμβάνεται ποτέ σε κάποια ερώτηση. Για παράδειγμα, μία πλειάδα για ένα τέτοιο site που ανήκει στο *Tier2* έχει την αντίστοιχη μορφή:

[Infrastructure, dteam, Tier2,`Tier2Sub`*, UK, UK-London-Tier2, UKI-LT2-HEP].*

Το φορτίο ερωτημάτων δημιουργήθηκε βάσει πραγματικών ερωτημάτων που έχουν τεθεί στο EGEE accounting portal κατά τη διάρκεια δύο μηνών. Δεδομένου ότι αυτή η εφαρμογή επεξεργάζεται περαιτέρω τα δεδομένα που συλλέγονται και δημιουργεί συνόψεις αυτών, προσφέρει τη δυνατότητα ένα ερώτημα να αναφέρεται σε μία ομάδα από VOs. Στα αποτελέσματα που παρουσιάζονται κάθε ερώτημα μεταφράζεται σε ισοδύναμα ερωτήματα, όπου κάθε ερώτημα αναφέρεται σε ένα VO από αυτά της συνολικής ομάδας. Κάθε ομάδα ερωτημάτων υποβάλλεται στο σύστημα μία φορά, ενώ θεωρείται ένας σταθερός ρυθμός εκτέλεσης για τις ομάδες των ερωτημάτων. Η ίδια σύμβαση ακολουθείται και στην περίπτωση του κεντρικοποιημένου και του *distributedCP* μοντέλου. Εξ όσων είναι γνωστά, είναι κοινή πρακτική να προσδιορίζεται ένα συγκεκριμένο VO όταν ερωτούνται τα διάφορα εργαλεία των Information services. Η απόδοση της προτεινόμενης αρχιτεκτονικής συγκρίνεται με το κεντρικοποιημένο μοντέλο, όπου μία κεντρική βάση δεδομένων αποθηκεύει όλη τη σχετική πληροφορία και απαντάει τα ερωτήματα και το *distributedCP* μοντέλο όπου ένας κεντρικός κατάλογος προωθεί τα ερωτήματα σε όλους τους

κόμβους με σχετικά δεδομένα. Το φορτίο ερωτημάτων περιέχει περίπου 20k ομάδες ερωτημάτων, που έχουν μεταφραστεί σε 250k ερωτήματα σύμφωνα με τον περιορισμό για τη δημιουργία ξεχωριστού ερωτήματος για κάθε VO. Η πλειοψηφία των ερωτημάτων στοχεύουν τα ℓ_2, ℓ_3 και ℓ_4 γύρω στ 85% του χρόνου (19%,36% και 32% αντιστοίχως).

Τα πειράματα επαναλήφθηκαν με τη χρήση κάθε πιθανού επιπέδου ως αρχικό επίπεδο εισαγωγής. Το precision του προτεινόμενου συστήματος παραμένει υψηλό και μόνο το πολύ 3% των ερωτημάτων είναι flood ερωτήματα. Η δομή των δεδομένων και η πόλωση του φορτίου ερωτημάτων ευνοούν την επίλυση των ερωτημάτων και δεν απαιτείται flooding. Η διαφορά έγκειται στο ποσοστό των exact match ερωτημάτων και των indexed ερωτημάτων. Το precision των exact match ερωτημάτων φτάνει γύρω στο 65%, όταν τα επίπεδα ℓ_2, ℓ_3 και ℓ_4 επιλέγονται ως το αρχικό *pivotlevel*. Επίσης, η επιλογή των επιπέδων αυτών ως pivot levels αποτελεί μία καλή λύση και για την κατανομή των δεδομένων στους κόμβους.



Σχήμα 3.26: *Μέσος φόρτος ερωτημάτων ανά κόμβο*

Το Σχήμα 3.26 απεικονίζει τις μεταβολές του μέσου αριθμού ερωτημάτων που επιλύονται ανά κόμβο κατά τη διάρκεια της εξομοίωσης. Κατά την έναρξη κάθε time unit, μία ομάδα ερωτημάτων τίθεται στο σύστημα και συνεπώς ο μέσος φόρτος της κεντρικοποιημένης λύσης είναι ίσος με τον αριθμό των ερωτημάτων που περιλαμβάνονται σε αυτήν την ομάδα. Η καμπύλη που αφορά το *distributedCP* μοντέλο αφορά μόνο το μέσο αριθμό ερωτημάτων σε ένα producer. Το φορτίο του κόμβου που διατηρεί τον κατάλογο είναι ίσο με αυτό του κόμβου του ή με ένα μέρος αυτού εάν υπάρχει ένας κόμβος για τον κατάλογο για κάθε VO. Η δομή δεικτοδότησης του προτεινόμενου συστήματος έχει ως αποτέλεσμα υψηλό precision, ενώ η επεξεργασία κάθε ερωτήματος γίνεται από λιγότερους κόμβους και επομένως ο φόρτος ερωτημάτων ανά κόμβο είναι σημαντικά μικρότερος σε σύγκριση με αυτόν στο μοντέλο *distributedCP* . Ο μέσος φόρτος ανά κόμβο του προτεινόμενου Grid Information System φτάνει το αντίστοιχο φόρτο ενός κόμβου

στο *distributedCP* μοντέλο, εάν το l_6 επιλεχθεί ως το αρχικό *pivotlevel*. Σε αυτήν την περίπτωση, οι δείκτες προωθούν τα ερωτήματα στον ίδιο σχεδόν αριθμό κόμβων όπως στο *distributedCP* μοντέλο.

ΚΕΦΑΛΑΙΟ 4

Ένα Κατανεμημένο Σύστημα για τη Διασύνδεση Πολυδιάστατων Δεδομένων

Στο Κεφάλαιο 4 παρουσιάζεται ένα κατανεμημένο σύστημα για την αποδοτική κατανομή και επεξεργασία πολυδιάστατων ιεραρχικών δεδομένων. Το συγκεκριμένο σύστημα ονομάζεται LinkedPeers και στοχεύει στην υποστήριξη δύο βασικών χαρακτηριστικών: υποστήριξη ευρείας κλίμακας για μερικώς δομημένα δεδομένα και αποδοτική, κατανεμημένη επεξεργασία ερωτημάτων. Οι προτεινόμενες στρατηγικές αναζήτησης επικεντρώνονται κυρίως στην επίλυση ερωτημάτων συνάθροισης (aggregate queries) σε πολλαπλές διαστάσεις. Για να επιτευχθεί η επίλυση τέτοιου είδους ερωτημάτων, το σύστημα LinkedPeers χρησιμοποιεί μία "εννοιολογική" αλυσίδα από DHT δακτυλίδια για την αποθήκευση της πληροφορίας με τρόπο τέτοιο ώστε να διατηρείται η ιεραρχική δομή. Επιπλέον, υλοποιούνται προσαρμοστικοί μηχανισμοί τόσο για τη ρύθμιση του επιπέδου λεπτομέρειας της δεικτοδότησης όσο και για τον "προ-υπολογισμό" πιθανών, μελλοντικών ερωτημάτων σύμφωνα με τα εισερχόμενα ερωτήματα. Η αποτελεσματικότητα του LinkedPeers αξιολογείται πειραματικά για διαφορετικά δεδομένα και φορτία ερωτημάτων αποδεικνύοντας ότι το σύστημα επιτυγχάνει υψηλό precision ενώ ταυτόχρονα μειώνει το κόστος επικοινωνίας και προσαρμόζει τις δομές δεικτοδότησης του ανάλογα με τα ερωτήματα που λαμβάνει.

Οι μέθοδοι που προτάθηκαν στο LinkedPeers προσαρμόζονται και επεκτείνονται κατά τέτοιο τρόπο, ώστε να χρησιμοποιηθούν σε ένα πραγματικό σενάριο χρήσης που υπαγορεύεται από τη διαχείριση διασυνδεδεμένων δεδομένων (Linked Data). Το πρόσφατο παράδειγμα της πρωτοβουλίας των Linked Open Data (LOD) έχει ως στόχο την ενοποίηση μεγάλου όγκου δεδομένων από

διαφορετικές πηγές προέλευσης στο Διαδίκτυο παρέχοντας μηχανισμούς ευρείας κλίμακας για reasoning. Ο υπολογισμός των ερωτημάτων στα αναφερόμενα περιβάλλοντα γίνεται κυρίως με τα εξής συστήματα: Κεντρικοποιημένα συστήματα και ενοποιημένα συστήματα (federation systems). Από τη μία πλευρά, η πρώτη κατηγορία συστημάτων αντιμετωπίζει περιορισμούς όσον αφορά την επεκτασιμότητα τους και τον όγκο των δεδομένων που μπορούν να διαχειριστούν. Από την άλλη πλευρά, το federation από διαφορετικές RDF αποθήκες (RDF stores) μπορεί να μειώσει την απόδοση του συστήματος. Λαμβάνοντας υπόψη τη δομή των διασυνδεδεμένων δεδομένων και τις απαιτήσεις για την αποτελεσματική διαχείριση τέτοιου είδους δεδομένων, προτείνεται το PI4LD: ένα σύστημα δεικτοδότησης που βασίζεται σε P2P τεχνολογίες για τη διαχείριση διασυνδεδεμένων δεδομένων. Σε αυτό το σύστημα, οι αποθηκευμένες οντότητες και κλάσεις οργανώνονται σε ιεραρχίες σύμφωνα με τις οντολογίες, τις ταξινομίες και τα λεξιλόγια που χρησιμοποιούνται. Οι σχέσεις μεταξύ των οντοτήτων διατηρούνται με ένα κατανεμημένο τρόπο, ενώ τα δεδομένα που παράγονται από διαφορετικές πηγές μπορούν να εισαχθούν στο σύστημα. Το προκύπτον σύστημα μπορεί να χρησιμοποιηθεί είτε ως μία αυτόνομη λύση για τη φιλοξενία διασυνδεδεμένων δεδομένων ή ως ένα ενδιάμεσο επίπεδο πάνω από federated πηγές. Το PI4LD υποστηρίζει διάφορους τύπους από ερωτήματα, όπως απλά ερωτήματα για την ανακάλυψη των οντοτήτων και ταυτόχρονα πιο πολύπλοκους τύπους ερωτημάτων. Η αναλυτική αξιολόγηση του συστήματος τόσο για συνθετικά όσο και για πραγματικά δεδομένα και η σύγκριση του με μία δημοφιλή RDF αποθήκη δείχνει ότι το PI4LD μπορεί να πετύχει μειωμένους χρόνους για την επεξεργασία των ερωτημάτων.

4.1 Ένα Κατανεμημένο Σύστημα για τη Διασύνδεση Πολυδιάστατων Δεδομένων

Τα δεδομένα σε ποικίλες εφαρμογές περιγράφονται από πολλαπλές διαστάσεις (π.χ. διάσταση χρόνος, διάσταση πελάτης, διάσταση τοποθεσία, κ.α.). Το πολυδιάστατο μοντέλο δεδομένων έχει υιοθετηθεί ευρέως από τις αποθήκες δεδομένων και τις OLAP εφαρμογές, δεδομένου ότι επιτρέπει την αναπαράσταση των δεδομένων με τη μορφή *κύβων δεδομένων*. Σε αυτή τη μορφή αναπαράστασης, οι διαστάσεις του κύβου αντιστοιχούν σε οντότητες για τις οποίες διατηρούνται εγγραφές. Για την αναπαράσταση των τιμών των διαστάσεων σε διαφορετικά επίπεδα λεπτομέρειας μπορούν να χρησιμοποιηθούν εννοιολογικές ιεραρχίες.

Πολυδιάστατα δεδομένα δεν χρησιμοποιούνται μόνο στις εφαρμογές αναλυτικής επεξεργασίας, αλλά αντίστοιχες απαιτήσεις για τη διαχείριση τέτοιου είδους δεδομένων τίθενται και από μία πληθώρα διαδικτυακών εφαρμογών. Σε τέτοιου είδους εφαρμογές, ένα άλλο βασικό χαρακτηριστικό είναι η έλλειψη κεντρικοποιημένων και αυστηρώς ορισμένων σχημάτων, όπως

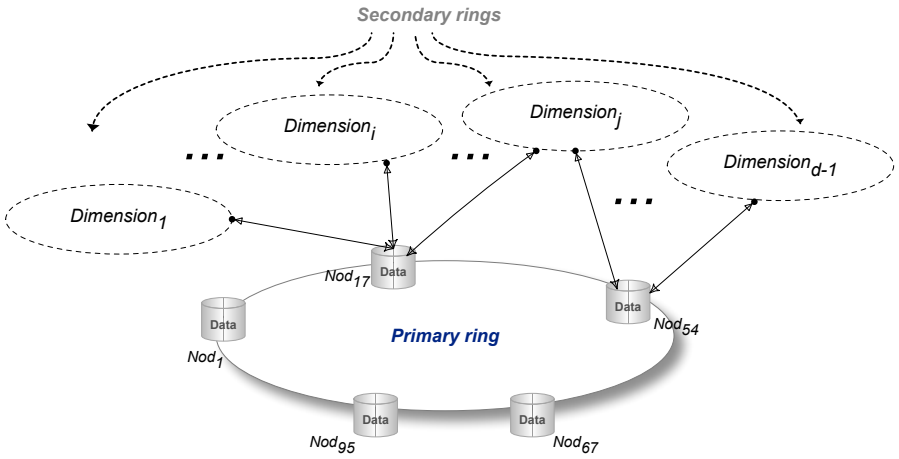
αυτά που χρησιμοποιούνται στα "παραδοσιακά" συστήματα βάσεων δεδομένων. Συνήθως, τέτοιου είδους εφαρμογές εκτελούνται σε κατανεμημένες πλατφόρμες, όπου διαφορετικές πηγές παράγουν δεδομένα [Data] και για αυτό το λόγο είναι δύσκολο να γίνει η επεξεργασία τους από κεντρικοποιημένες λύσεις. Συνεπώς, τέτοιου είδους πλατφόρμες πρέπει να έχουν τη δυνατότητα να επεξεργάζονται πολυδιάστατα δεδομένα που είναι αποθηκευμένα κατανεμημένα με αποτελεσματικό και online τρόπο.

Το σύστημα *LinkedPeers* που προτείνεται σε αυτήν τη διατριβή αποτελεί ένα σύστημα αποθήκευσης και επεξεργασίας δεδομένων πολλαπλών διαστάσεων με αποτελεσματικό τρόπο. Κάθε διάσταση οργανώνεται περαιτέρω με τη χρήση εννοιολογικών ιεραρχιών. Η οργάνωση οποιαδήποτε αριθμού διαθέσιμων πόρων για την αποθήκευση των δεδομένων επιτυγχάνεται με τη χρήση ενός *DHT*. Οι πόροι που παράγουν τα δεδομένα μπορούν να τα εισάγουν στο σύστημα αυτόνομα και να τα ενημερώσουν με νέες εγγραφές οποιαδήποτε στιγμή. Για την περιγραφή των δεδομένων δε χρειάζεται ο αριθμός των διαστάσεων να είναι σταθερός, ενώ κάθε διάσταση περιγράφεται από μία εννοιολογική ιεραρχία. Η επεξεργασία των ερωτημάτων γίνεται με έναν πλήρως κατανεμημένο τρόπο και μπορεί να ενεργοποιήσει μηχανισμούς προσαρμογής της δεικτοδότησης βάσει των ερωτημάτων και προ-υπολογισμού ερωτημάτων για την ελαχιστοποίηση του κόστους επικοινωνίας.

Το κίνητρο για το σχεδιασμό του συγκεκριμένου συστήματος είναι η ανάπτυξη μίας κατανεμημένης υποδομής ευρείας κλίμακας για τη διαχείριση μερικώς δομημένων δεδομένων. Το σύστημα αυτό διαφοροποιείται από υπάρχουσες προσεγγίσεις στις οποίες τα σχήματα και οι σχέσεις μεταξύ των δεδομένων καθορίζονται κατά την αρχική εισαγωγή των δεδομένων στο σύστημα. Το *LinkedPeers* στοχεύει στην παροχή μεγαλύτερου βαθμού ελευθερίας στη δομή των δεδομένων: Τα δεδομένα περιγράφονται από d-διαστάσεις, καθεμία εκ των οποίων χρησιμοποιεί την αντίστοιχη ιεραρχία που έχει οριστεί για αυτή. Παρόλα αυτά, δεν απαιτείται η περιγραφή του αντικειμένου που εισάγεται στο σύστημα από όλες τις διαστάσεις. Εν αντιθέσει, ο στόχος είναι να υποστηριχθεί το συγκεκριμένο χαρακτηριστικό και να δημιουργηθεί ένα σύστημα για την αποτελεσματική επεξεργασία τέτοιου είδους δεδομένων.

Το *LinkedPeers* καταφέρνει να οργανώσει τα υπό μελέτη δεδομένα με τέτοιο τρόπο ώστε να διατηρείται όλη η πληροφορία που εμπεριέχεται στην ιεραρχία. Για αυτό το λόγο χρησιμοποιούνται δενδρικές δομές δεδομένων, μία για κάθε διάσταση. Οι δενδρικές δομές διαφορετικών διαστάσεων ανατίθενται σε διαφορετικούς κόμβους του δικτύου, αλλά ταυτόχρονα διατηρούνται σύνδεσμοι μεταξύ των δέντρων που συνδέονται με κάποια σχέση. Η οργάνωση του DHT επιπέδου γίνεται σύμφωνα με τη διάταξη των διαστάσεων που μπορεί να προκύψει από τη σημαντικότητα της κάθε διάστασης, την πόλωση των ερωτημάτων, κλπ.: Το *LinkedPeers* αποτελείται από πολλαπλές "εικονικές" επικαλύψεις, μία για κάθε διάσταση. Στην πραγματικότητα τα κλειδιά για όλες τις διαστάσεις μπορούν να μοιράζονται τον ίδιο χώρο αναγνωριστικών (*identifier space*) και να χρησιμοποιείται μία DHT επικάλυψη για όλες τις διαστάσεις αντί της δημιουργίας

μίας διαφορετικής για κάθε διάσταση. Σε αυτήν την περίπτωση, ο διαχωρισμός των δακτυλιδιών της κάθε διάστασης είναι εικονικός. Μία αφηρημένη απεικόνιση των εικονικών δακτυλιδιών της αρχιτεκτονικής που έχει υιοθετηθεί στο *LinkedPeers* για *d*-διαστάσεις φαίνεται στην Εικόνα 4.1. Η στρατηγική για τη δημιουργία ενός διαφορετικού δακτυλιδιού για κάθε διάσταση έχει ως συνέπεια τη διάσπαση της συνολικής πλειάδας σε επιμέρους τμήματα που περιέχουν τις τιμές της κάθε διάστασης και την εισαγωγή της κάθε "υπο-πλειάδας" στο πρωτεύον (primary) και στα δευτερεύοντα (secondary) δακτυλίδια αντίστοιχα. Οι δενδρικές δομές στα δευτερεύοντα δακτυλίδια διατηρούν πληροφορία σχετικά με τα δέντρα που συσχετίζονται στο πρωτεύον δακτυλίδι.



Σχήμα 4.1: *Γενική απεικόνιση της αρχιτεκτονικής που έχει υιοθετηθεί στο LinkedPeers*

Η ιδέα πίσω από το σχεδιασμό του *LinkedPeers* είναι η δημιουργία ενός αυτόνομου, "πρωτεύοντος" δακτυλιδιού για την αποθήκευση των δεδομένων, ενώ τα δευτερεύοντα δακτυλίδια συμβάλουν στην αποτελεσματική δημιουργία των απαιτούμενων δεικτών, επιτρέποντας στο σύστημα να επιστρέφει γρήγορα συνοπτικά αποτελέσματα για τις ερωτηθέντες τιμές ελαχιστοποιώντας το κόστος επικοινωνίας. Με τη δημιουργία επιπλέον δεικτών και τον προ-υπολογισμό ερωτημάτων, η αποτελεσματικότητα του συστήματος εμπλουτίζεται περαιτέρω.

Το προτεινόμενο σύστημα επιτρέπει την επεξεργασία πολύπλοκων aggregate ερωτημάτων για οποιοδήποτε επίπεδο της κάθε διάστασης, όπως τα ακόλουθα παραδείγματα: "*Ποια μέλη του επιπέδου* `City` *ανήκουν στο* `Country` *'Greece'?*" ή "*Ποιο είναι το* `population` *που αντιστοιχεί στο* `Country` *'Greece' ?*" ή "*Ποια* `Cities` *του* `Country` *'Greece' έχουν* `population` *πάνω από 1 εκατομμύριο κατά το διάστημα που αντιστοιχεί στο* `Year` *'2000'?*", όπου θεωρείται ότι τα διάφορα επίπεδα αντιστοιχούν στις ιεραρχίες `Location` (όπως απεικονίζεται στο Σχήμα 3.1) και `Time`(όπως απεικονίζεται στο 3.4) και περιγράφουν ένα fact για τον πληθυσμό (`population`). Η εφαρμογή της προτεινόμενης δεικτοδότησης επιτρέπει την εύρεση οποιασδήποτε τιμής κάθε ιεραρχίας χωρίς

να απαιτούνται επιπρόσθετες πληροφορίες, ενώ οι συναρτήσεις σύνοψης μπορούν να υπολογιστούν στους κόμβους που καταλήγει ένα ερώτημα.

Συμπερασματικά, το σύστημα *LinkedPeers* χαρακτηρίζεται από τα ακόλουθα καινοτόμα χαρακτηριστικά:

- Αποτελεί μία ολοκληρωμένη λύση για την αποθήκευση, δεικτοδότηση και επεξεργασία ερωτημάτων για δεδομένα που περιγράφονται από ένα μεταβλητό αριθμό διαστάσεων, κάθε μια από τις οποίες δομείται περαιτέρω από εννοιολογικές ιεραρχίες. Το *LinkedPeers* μπορεί να εκτελέσει online ενημερώσεις και να διαχειριστεί τα δεδομένα με πλήρως κατανεμημένο και ανεκτικό σε σφάλματα τρόπο.
- Παρέχει ένα μηχανισμό υλοποίησης (materialization) που προϋπολογίζει τις σχετικές όψεις για μελλοντική χρήση που αφορούν τιμές των ερωτημάτων που απαντιούνται.
- Προσαρμόζει τις δομές δεικτοδότησης σύμφωνα με τα ερωτήματα που θέτονται στο σύστημα.

Η ανάλυση του συστήματος υποστηρίζεται από μία αναλυτική πειραματική αξιολόγηση της απόδοσης του, έτσι ώστε να αναγνωριστεί η συμπεριφορά του για διάφορα φορτία δεδομένων και ερωτημάτων.

4.2 Συμβολισμός και Ορισμοί

Τα δεδομένα που εισάγονται στο σύστημα αποτελούνται από πλειάδες που περιέχουν τιμές από ένα χώρο δεδομένων D. Οι πλειάδες αυτές ορίζονται από ένα σύνολο διαστάσεων $\{d_0, .., d_{d-1}\}$, καθώς και από τα πραγματικά facts. Κάθε διάσταση d_i συσχετίζεται με μία εννοιολογική ιεραρχία που οργανώνεται σε L_i επίπεδα σύνοψης ℓ_{ij}, όπου το $(j \in [0, L_i - 1])$ αναπαριστά το *j-οστό* επίπεδο της *i-τής* διάστασης. Επίσης, ορίζεται ότι το ℓ_{ik} βρίσκεται *υψηλότερα* (*higher*) (ή χαμηλότερα (*lower*)) από το επίπεδο ℓ_{il} και συμβολίζεται ως $\ell_{ik} < \ell_{il}$ (ή $\ell_{ik} > \ell_{il}$ αντίστοιχα) *iff* $k < l$ $(k > l)$, εάν το ℓ_{ik} αντιστοιχεί σε ένα λιγότερο (περισσότερο) λεπτομερές επίπεδο από το ℓ_{il} (π.χ., $Month < Day$). Οι πλειάδες αναπαριστάνονται με την εξής μορφή:

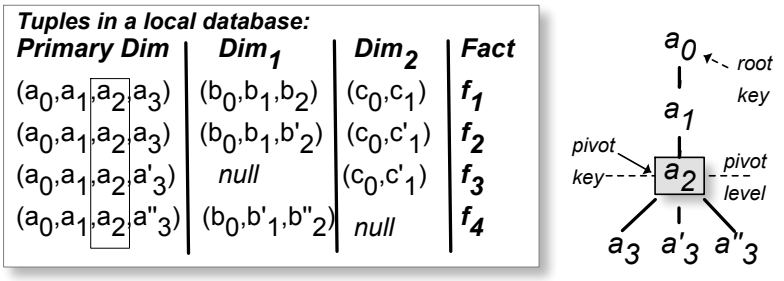
$$\langle v_{0,0}, \ldots, v_{0,L_0-1}, \ldots, v_{d-1,0}, \ldots, v_{d-1,L_{d-1}-1}, f_0, \ldots \rangle$$

όπου το $v_{i,j}$ αναπαριστά την τιμή του *j-οστού* επιπέδου της διάστασης i. Επισημαίνεται επίσης ότι οι τιμές που αντιστοιχούν στο σύνολο τιμών (*value-set* $v_{i,0}, \ldots, v_{i,L_i-1}$) κάποιας διάστασης μπορούν να απουσιάζουν από κάποια πλειάδα και ότι το *fact* (e.g., f_0) μπορεί να είναι οποιοδήποτε τύπου (π.χ., αριθμητικό, κείμενο, διανυσματικό, κ.α.), αλλά κάποια διάσταση πρέπει να θεωρηθεί ως primary. Το επίπεδο ℓ_{i0} ονομάζεται *root level* της i-της διάστασης και η hashed τιμή

που αντιστοιχεί στο v_{i0} ονομάζεται *root key*. Οι τιμές του χαμηλότερου επιπέδου των ιεραρχιών ($v_{i,(L_i-1)}$) αναφέρονται ως *τιμές-φύλλα* (*leaf values*).

Οι τιμές των επιπέδων της ιεραρχίας σε κάθε διάσταση οργανώνονται σε δενδρικές δομές. Χωρίς απώλεια της γενικότητας, θεωρείται ότι κάθε τιμή του επιπέδου ℓ_{ij} έχει ένα μοναδικό πρόγονο στο αμέσως υψηλότερο επίπεδο (δηλαδή στο επίπεδο $\ell_{i(j-1)}$). Για την εισαγωγή των πλειάδων στα πολλαπλά δακτυλίδια, ένα επίπεδο της κάθε διάστασης επιλέγεται; η hashed τιμή του εξυπηρετεί ως *key* στο υποκείμενη DHT επικάλυψη. Κάθε αναφορά σε αυτό το επίπεδο γίνεται ως *pivot key*. Το pivot key που αντιστοιχεί στη primary διάσταση (ή στο primary δακτυλίδι) ονομάζεται *primary key*. Τα pivot levels που αντιστοιχούν στο υψηλότερο και στο χαμηλότερο επίπεδο κάθε ιεραρχίας για ένα συγκεκριμένο root key ονομάζονται *MinPivotLevel* και *MaxPivotLevel* αντίστοιχα.

Οι τιμές του value-set μίας διάστασης μαζί με το αντίστοιχο aggregated fact οργανώνονται ως κόμβοι μίας δενδρικής δομής (*tree structure*), η οποία συνεισφέρει στη διατήρηση των σημασιολογικών σχέσεων και στην αναζήτηση των τιμών. Το Σχήμα 4.2 περιγράφει το παράδειγμα αναφοράς για τις πλειάδες που θεωρούνται και στα επόμενα παραδείγματα που παρουσιάζουν τις διάφορες λειτουργίες του συστήματος. Οι πλειάδες αυτές ακολουθούν ένα σχήμα τριών διαστάσεων. Η primary διάσταση περιγράφεται από μία ιεραρχία τεσσάρων επιπέδων, ενώ οι υπόλοιπες δύο περιγράφονται από δύο ιεραρχίες τριών και δύο επιπέδων αντίστοιχα. Επισημαίνεται ότι στο συγκεκριμένο παράδειγμα οι δύο τελευταίες πλειάδες δεν περιέχουν τιμές για τις διαστάσεις d_1 και d_2 αντίστοιχα. Το επιλεγμένο ως αρχικό pivot level για τη primary διάσταση είναι το ℓ_{02} και συνεπώς όλες οι πλειάδες που φαίνονται έχουν το ίδιο pivot key στην primary διάσταση. Όλα τα value-sets σε κάθε διάσταση οργανώνονται σε δενδρικές δομές που έχουν τα ίδια root keys σε κάθε διάσταση.



Σχήμα 4.2: *Τέσσερις διαφορετικές πλειάδες με διάφορους συνδυασμούς τιμών ανάμεσα στις διαστάσεις και η δενδρική δομή που προκύπτει για την primary διάσταση.*

Ο βασικός τύπος ερωτήματος που υποστηρίζεται στο *LinkedPeers* είναι της μορφής:

$$q = (q_{0k}, ..., q_{ij}, ..., q_{(d-1)m})$$

και αφορά τη συνάθροιση των fact(s) με τη χρήση της κατάλληλης συνάρτησης συνάθροισης (aggregation function). Το q_{ij} συμβολίζει την τιμή του *j-οστού* επιπέδου της ιεραρχίας το οποίο μπορεί επίσης να πάρει την τιμή *'*'* (or *ALL*). Όταν αυτή η τιμή προσδιορίζεται για κάποια διάσταση, τότε οποιαδήποτε τιμή της συγκεκριμένης διάστασης θεωρείται αποδεκτή για την απάντηση του συγκεκριμένου ερωτήματος.

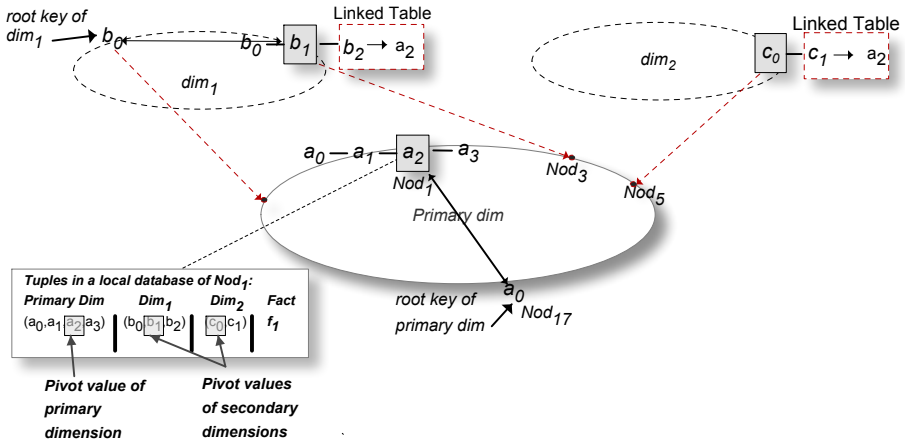
4.3 Εισαγωγή των πολυδιάστατων δεδομένων

Το προτεινόμενο σύστημα χειρίζεται τόσο μαζικές εισαγωγές (bulk insertions) όσο και σταδιακές ενημερώσεις (incremental updates) με έναν ενιαίο τρόπο. Δεδομένου ότι ο σχεδιασμός του συστήματος συνεπάγεται μία εικονική επικάλυψη για κάθε διάσταση, δημιουργείται ένα key (για παράδειγμα με τη χρήση της SHA1 συνάρτησης κατακερματισμού) για το επιλεγμένο pivot value της κάθε διάστασης.

Κατά την εισαγωγή των δεδομένων, η πληροφορία για το pivot value είναι ιδιαίτερα σημαντική (μόνο κατά τις αρχικές εισαγωγές επιλέγεται το pivot level σύμφωνα με τις ανάγκες των εφαρμογών). Παρόλα αυτά, ο σχεδιασμός που ακολουθήθηκε στο *LinkedPeers* προϋποθέτει ότι εάν επιλεχθεί μία τιμή v_{ij} ως pivot key κατά την εισαγωγή μίας πλειάδας, τότε κάθε πλειάδα που περιέχει τη συγκεκριμένη τιμή v_{ij} πρέπει να τη χρησιμοποιήσει ως pivot key στην *i-τή* διάσταση. Για να τηρηθεί η συγκεκριμένη υπόθεση, ένας κόμβος πρέπει να γνωρίζει τα pivot keys που υπάρχουν κατά την εισαγωγή μιας καινούργιας πλειάδας. Για αυτό το λόγο, στο *LinkedPeers* υλοποιείται ένας πλήρως κατανεμημένος κατάλογος που αποθηκεύει πληροφορία σχετικά με τα root keys και τα αντίστοιχα pivot keys τους στο δίκτυο. Κάθε root key αποθηκεύεται στον κόμβο που είναι υπεύθυνος για αυτό. Κάθε φορά που ένα καινούργιο pivot key του συγκεκριμένου root key εισάγεται στο σύστημα, τότε ο κόμβος που είναι υπεύθυνος για το root key ενημερώνεται για αυτό και το προσθέτει σε μία λίστα με τα γνωστά pivot keys που διατηρείται για αυτό το root key. Το root key επίσης είναι ενημερωμένο και για το MaxPivotLevel που έχει χρησιμοποιηθεί για τις εισαγωγές των τιμών του στη συγκεκριμένη διάσταση.

Η διαδικασία για την εισαγωγή των τιμών μίας πλειάδας καταλλήλως σε όλες τις διαστάσεις αποτελείται από τα ακόλουθα βασικά βήματα:

- Το root key της κάθε διάστασης ενημερώνεται για το αντίστοιχο value-set $(v_{i,0}, \ldots, v_{i,L_i-1})$ της πλειάδας, έτσι ώστε να αποφασιστεί το pivot level που ενδείκνυται να χρησιμοποιηθεί.
- Κάθε value-set $(v_{i,0}, \ldots, v_{i,L_i-1})$ εισάγεται στο αντίστοιχο *i-οστό* δακτυλίδι.
- Δημιουργούνται ή ενημερώνονται οι σύνδεσμοι (links) μεταξύ των δέντρων των secondary διαστάσεων προς την primary διάσταση.

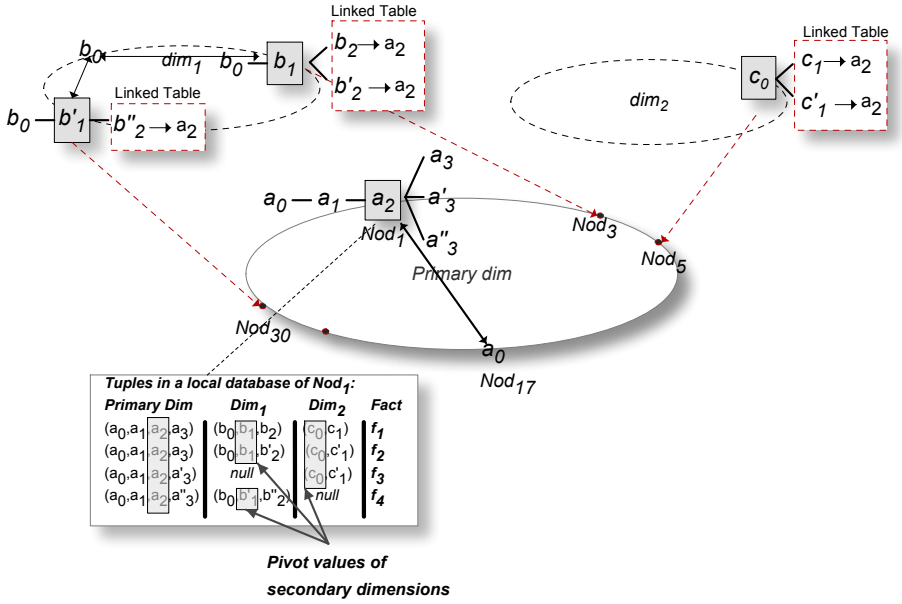


Σχήμα 4.3: *Οι δομές για τα δεδομένα που δημιουργήθηκαν μετά την εισαγωγή της πρώτης πλειάδας που εμφανίζεται στο Σχήμα 4.2*

Αρχικά, ο κόμβος που ξεκινάει τη διαδικασία (κόμβος initiator) επικοινωνεί με το root key του value-set της primary διάστασης. Το root key της primary διάστασης ενημερώνεται για την καινούργια πλειάδα που πρόκειται να εισαχθεί στο σύστημα και υποδεικνύει το κατάλληλο pivot level: Εάν το ίδιο pivot key υπάρχει ήδη, τότε το pivot level επιλέγεται, ενώ στην αντίθετη περίπτωση επιλέγεται το MaxPivotLevel. Όταν το root key δεν υπάρχει ήδη στο σύστημα, τότε αποθηκεύεται στον κόμβο που είναι υπεύθυνος για αυτό και το pivot level του επιλέγεται είτε με τυχαίο τρόπο ή παίρνει την τιμή ενός προεπιλεγμένου pivot level για όλο το σύστημα. Στη συνέχεια εκκινείται η DHT λειτουργία για την εισαγωγή της πλειάδας στην primary διάσταση και αυτό έχει ως αποτέλεσμα την κατάληξη της πλειάδας στον κόμβο που είναι υπεύθυνος για το pivot key που αποφασίστηκε. Ο κόμβος που είναι υπεύθυνος για το pivot key της primary διάστασης αποθηκεύει το αντίστοιχο value-set σε μία δενδρική τιμή και ολόκληρη την πλειάδα σε μία αποθήκη, η οποία αναφέρεται από εδώ και στο εξής ως *τοπική βάση δεδομένων* (*local database*). Επιπλέον, αποθηκεύει το aggregated αποτέλεσμα (ή αποτελέσματα) για όλες τις πλειάδες που έχουν την ίδια τιμή σε κάθε επίπεδο (δηλαδή το αποτέλεσμα των αντίστοιχων ερωτημάτων (v_{0j}, $*, \ldots, *$), όπου $j \in [0, L_i - 1]$). Το Σχήμα 4.3 παρουσιάζει την εισαγωγή του value-set (a_0, a_1, a_2, a_3) στο primary δακτυλίδι σε μία επικάλυψη που αποτελείται από κόμβους που συμβολίζονται ως Nod_i. Το root key a_0 δεν υπάρχει στην επικάλυψη και το ℓ_{02} επιλέγεται με τυχαίο τρόπο ως pivot level. Έτσι, δημιουργείται το root index από το a_0 προς το a_2 και η πλειάδα εισάγεται στον κόμβο Nod_1, ο οποίος είναι υπεύθυνος για το pivot key της τιμής a_2 σύμφωνα με το DHT πρωτόκολλο. Ο κόμβος Nod_1 εισάγει επίσης όλες τις τιμές της πλειάδες στο local database του.

Το επόμενο βήμα είναι η αποθήκευση των value-sets για τις υπόλοιπες διαστάσεις στα αντίστοιχα δακτυλίδια. Ο κόμβος που είναι υπεύθυνος για το primary key επικοινωνεί με κάθε κόμβο

που είναι υπεύθυνος για κάθε κλειδί και ενημερώνεται για το κατάλληλο pivot level στη διάσταση d_i. Αφού προσδιοριστούν τα pivot levels για τις secondary διαστάσεις, το value-set της κάθε διάστασης αποθηκεύεται στον κόμβο που είναι υπεύθυνος για το pivot key του. Ξανά, τα αντίστοιχα συνοπτικά αποτελέσματα διατηρούνται στους κόμβους των δενδρικών δομών. Οι τιμές των secondary διαστάσεων συσχετίζονται με την primary διάσταση μέσω του primary key. Κάθε leaf value ενός δέντρου σε ένα secondary ring διατηρεί μία λίστα από primary keys με τα οποία συνδέεται. Η δομή που αποθηκεύει τις αντιστοιχήσεις μεταξύ των leaf values και των primary keys αναφέρεται ως *Linked Table*. Επίσης, ο κόμβος που είναι υπεύθυνος για το primary key αποθηκεύει τα pivot levels των value-sets των secondary διαστάσεων στην τοπική βάση του μαζί με την ολόκληρη πλειάδα. Μία άλλη παρατήρηση είναι ότι εάν η εισαγωγή μίας πλειάδας δεν εκτελείται κατά την αρχική φόρτωση των δεδομένων στο σύστημα και ένα root key υπάρχει ήδη, τότε οι υπάρχοντες soft-state δείκτες πρέπει και αυτοί να ενημερωθούν για την καινούργια πλειάδα. Η διαδικασία που περιγράφηκε στην Ενότητα 3.5.1 μπορεί να ακολουθηθεί για το value-set για το οποίο υπάρχει το root key του. Σε αυτήν την περίπτωση, δεδομένου ότι οι soft-state δείκτες μπορεί να αποθηκεύουν τα aggregated facts για τις δεικτοδοτημένες τιμές, οι soft-state δείκτες πρέπει να ενημερωθούν όχι μόνο για τις τοποθεσίες των καινούργιων δέντρων, αλλά και για τα καινούργια facts. Εάν ένα δέντρο υπάρχει ήδη, τότε οι τιμές που έχουν σημειωθεί ως δεικτοδοτημένες πρέπει να μάθουν για την καινούργια πλειάδα.



Σχήμα 4.4: *Τελική τοποθέτηση και δεικτοδότηση των πλειάδων, που εμφανίζονται στο Σχήμα 4.2, στο LinkedPeers*

Στο Σχήμα 4.3 απεικονίζονται και οι δομές για τα δεδομένα των secondary διαστάσεων, όπου τα δέντρα αποτελούντα από μόνο ένα κλάδο. Κατά την εισαγωγή του value-set (b_0, b_1, b_2), ο

root index b_0 δημιουργείται (το pivot level ισούται με το root level και δεν είναι αναγκαία η δημιουργία επιπρόσθετης δεικτοδότησης στην περίπτωση που αφορά την τιμή c_0). Το Σχήμα 4.4 απεικονίζει την τελική τοποθέτηση των τιμών των πλειάδων του Σχήματος 4.2 στους κόμβους της επικάλυψης. Όταν εισάγεται η δεύτερη πλειάδα στην επικάλυψη, το root index για την τιμή a_0 καθορίζει ότι η τιμή a_2 υπάρχει ήδη στο δίκτυο και αντιστοιχεί σε pivot key. Για αυτό το λόγο, η πλειάδα αυτή πρέπει να αποθηκευτεί στον κόμβο Nod_1 και ένας νέος κλάδος εισάγεται στο υπάρχον δέντρο. Οι τιμές της διάστασης d_1 δεν έχουν οριστεί στην τρίτη πλειάδα, αλλά το γεγονός αυτό δεν επηρεάζει τη διαδικασία της εισαγωγής. Η εισαγωγή των τιμών για τη διάσταση d_1 έχει σαν αποτέλεσμα τη δημιουργία ενός καινούργιου δέντρου για το pivot value b'_1. Εφόσον το primary key για όλες τις πλειάδες είναι το ίδιο, το local database του κόμβου Nod_1 περιέχει όλες τις γραμμές που φαίνονται στο σχήμα 4.2.

4.4 Επεξεργασία Ερωτημάτων

Τα ερωτήματα που θέτονται στο σύστημα αποτελούνται από συνδυασμούς τιμών. Όταν ένα ερώτημα περιλαμβάνει μία τιμή που αντιστοιχεί σε ένα pivot level, τότε ο κόμβος που είναι υπεύθυνος για τη συγκεκριμένη τιμή μπορεί να εντοπιστεί με μία απλή DHT αναζήτηση. Διαφορετικά, ο μηχανισμός εντοπισμού τιμών του DHT πρωτοκόλλου δεν επαρκεί για την αναζήτηση των υπόλοιπων αποθηκευμένων τιμών. Οι προτεινόμενες τεχνικές μπορούν να χρησιμοποιηθούν και για τον εντοπισμό των υπόλοιπων αποθηκευμένων τιμών.

Η λογική της προσέγγισης που ακολουθήθηκε για την εισαγωγή των πλειάδων στη DHT επικάλυψη στοχεύει στη διατήρηση των διασυνδέσεων μεταξύ των πολλαπλών διαστάσεων με έναν κατανεμημένο τρόπο, ώστε να δίνεται η δυνατότητα αναζήτησης των τιμών των διαστάσεων είτε μεμονωμένα είτε με συνδυασμούς πολλαπλών τιμών. Όταν ένα ερώτημα δεν προσδιορίζει μία συγκεκριμένη τιμή για κάποια διάσταση (δηλαδή χρησιμοποιείται η '*'-τιμή για αυτή τη διάσταση), τότε κάθε πιθανή τιμή είναι αποδεκτή για την επίλυση του ερωτήματος. Θεωρείται ότι ένα ερώτημα μπορεί να περιέχει μέχρι *d-1* '*' τιμές για τις *d διαστάσεις*.

Δεδομένου ότι το *LinkedPeers* επιτρέπει την προσαρμοστική μεταβολή των pivot levels σύμφωνα με το φορτίο των ερωτημάτων, ένας κόμβος που ξεκινάει μία αναζήτηση δε γνωρίζει εκ των προτέρων εάν κάποια από τις τιμές που περιέχεται στο ερώτημα αντιστοιχεί σε pivot value. Για αυτό το λόγο, θέτει διαδοχικά lookups για κάθε τιμή που περιέχεται στο ερώτημα ανάλογα με την προτεραιότητα των διαστάσεων, μέχρι να επιστραφεί κάποιο αποτέλεσμα. Αρχικά, χρησιμοποιείται το *lookup* του DHT για τον εντοπισμό της τιμής με την υψηλότερη προτεραιότητα. Εάν ο κόμβος που αποθηκεύει τη συγκεκριμένη τιμή δε μπορεί να βρεθεί, τότε ακολουθείται η ίδια διαδικασία για την επόμενη τιμή που δεν είναι '*'. Εάν δεν εντοπιστεί καμία από τις τιμές που εμπεριέχεται στο ερώτημα, τότε η συγκεκριμένη ερώτηση αποστέλλεται σε όλους τους κόμβους του συστήματος.

4.4.1 Exact Match Ερωτήματα

Τα ερωτήματα που περιέχουν τουλάχιστον μία τιμή που αντιστοιχεί σε pivot level αναφέρονται ως *exact match* ερωτήματα και μπορούν να επιλυθούν με τη χρήση του lookup μηχανισμού του DHT. Διακρίνονται δύο κατηγορίες για τα exact match ερωτήματα:

Κατηγορία 1: Το ερώτημα είναι της μορφής $q = (q_{0pivotlevel}, \ldots)$, δηλαδή στο ερώτημα περιλαμβάνεται ένα pivot value της primary διάστασης. Στις υπόλοιπες διαστάσεις μπορεί να περιλαμβάνονται και άλλες τιμές. Σε αυτήν την περίπτωση, το DHT lookup καταλήγει στον κόμβο που είναι υπεύθυνος για το pivot key της primary διάστασης. Εάν το ερώτημα περιλαμβάνει μόνο αυτήν την τιμή, τότε η αντίστοιχη δενδρική δομή εξερευνάται για το aggregate fact. Διαφορετικά, το local database ερωτάται και τα αποτελέσματα επιλέγονται ανάλογα με τις υπολειπόμενος τιμές τοπικά.

Κατηγορία 2: Το ερώτημα είναι της μορφής $q = (q_{0j}, \ldots, q_{ipivotlevel}, \ldots)$, όπου το q_{0j} δεν αντιστοιχεί σε κάποιο pivot value. Όταν συμβαίνει αυτό, κάποια τιμή του ερωτήματος σε μία secondary διάσταση αντιστοιχεί σε ένα pivot value. Η στρατηγική που ακολουθείται για την επίλυση αυτού του ερωτήματος είναι η αποστολή διαδοχικών ερωτημάτων για τις τιμές του ερωτήματος μέχρι να βρεθεί ο κόμβος που είναι υπεύθυνος για την τιμή που αντιστοιχεί σε pivot level, δηλαδή για το $q_{ipivotlevel}$. Εάν το ερώτημα δεν περιλαμβάνει άλλες τιμές, τότε η δενδρική τιμή που βρέθηκε αποθηκεύει επαρκή πληροφορία για την επίλυση του ερωτήματος, διαφορετικά το ερώτημα προωθείται σε όλους τους κόμβους της primary διάστασης που αποθηκεύουν πλειάδες με την τιμή $q_{ipivotlevel}$. Οι κόμβοι αυτοί ψάχνουν τα local databases τους για την ανάκτηση των σχετικών πλειάδων και επιστρέφουν τα αποτελέσματα στον κόμβο initiator. Εάν περισσότερα pivot values περιλαμβάνονται στο ερώτημα, τότε το ερώτημα επιλύεται από τη διάσταση με την υψηλότερη προτεραιότητα. Στο παράδειγμα του Σχήματος 4.4, ένα ερώτημα για την τιμή b_1 μπορεί να απαντηθεί με το aggregated fact που αποθηκεύεται στον κόμβο Nod_3. Ωστόσο, ένα ερώτημα για το συνδυασμό τιμών $(a_3, b_1, *)$ φτάνει στον κόμβο Nod_3, που δε διαθέτει τοπικά την απαραίτητη πληροφορία για να το απαντήσει και (χρησιμοποιώντας το Linked Table του) προωθεί το ερώτημα στον κόμβο Nod_1, ο οποίος θέτει τελικά το ερώτημα στο local database του.

4.4.2 Flood Ερωτήματα

Τα ερωτήματα που δεν περιέχουν κάποια τιμή που ανήκει σε κάποιο pivot level δεν μπορούν να επιλυθούν από τον εγγενή μηχανισμό του DHT. Η μόνη εναλλακτική είναι να σταλεί το ερώτημα σε όλους τους κόμβους του συστήματος, οι οποίοι το επεξεργάζονται ατομικά. Σε περίπτωση που το ερώτημα περιέχει μόνο μία τιμή, τότε οι δενδρικές δομές του κάθε κόμβου διερευνώνται για την τιμή αυτή. Διαφορετικά, ο κόμβος απευθύνει το ερώτημα στο local database και στέλνει τα αποτελέσματα που βρέθηκαν πίσω στον κόμβο initiator.

Στοχεύοντας στην ελαχιστοποίηση του κόστους επικοινωνίας και επεξεργασίας, η διαδικασία επίλυσης ενός flood ερωτήματος εμπλουτίζεται περαιτέρω. Για να επιτευχθεί αυτό αξιοποιούνται οι DHT μηχανισμοί και οι ιδιότητες της δομής των δεδομένων, ώστε να αποφευχθεί η επίσκεψη του ίδιου κόμβου πολλαπλές φορές και να θεσπιστεί μία σειρά στον τρόπο με τον οποίο προωθείται ένα ερώτημα στους κόμβους της επικάλυψης. Η ιεραρχική δομή των δεδομένων και οι δομές δεικτοδότησης που χρησιμοποιούνται επιτρέπουν τη δημιουργία μίας ελεγχόμενης στρατηγικής flooding που μειώνει σημαντικά το κόστος επικοινωνίας.

Αρχικά, ένα flood ερώτημα προωθείται από ένα κόμβο στον "πλησιέστερο" γείτονα του στη DHT επικάλυψη. Κάθε κόμβος που λαμβάνει το ερώτημα ψάχνει τις δενδρικές δομές του για κάθε τιμή του ερωτήματος. Επίσης, κάθε ξεχωριστή τιμή καθώς και ο συνδυασμός των τιμών που εμπεριέχονται στο ερώτημα αναζητούνται στο local database του. Εάν δε βρεθεί τίποτα σχετικό στον κόμβο αυτό, τότε ο παρόν κόμβος καταγράφει το εύρος (ή εύρη) των IDs για τα οποία είναι υπεύθυνος στο flood μήνυμα και προωθεί το ερώτημα στον πλησιέστερο γείτονα του. Η στρατηγική εφαρμόζεται ώστε να αποφευχθεί η επίσκεψη αυτού του κόμβου ξανά σε κάποια μελλοντική φάση της επίλυσης αυτού του ερωτήματος. Η λογική αυτής της στρατηγικής στηρίζεται στην παραδοχή ότι αν ένας κόμβος έχει ήδη ερωτηθεί και δεν περιέχει σχετικές πλειάδες, τότε δεν υπάρχει κάποιο κέρδος από την προώθηση του ερωτήματος σε αυτόν τον κόμβο ξανά, ακόμα και αν υποδειχθεί ως ένας υποψήφιος κόμβος, ο οποίος είναι πιθανός να αποθηκεύει πλειάδες σχετικές με το ερώτημα που επιλύεται.

Εάν βρεθεί σχετική πληροφορία στον κόμβο που έχει φτάσει το ερώτημα, τότε η προώθηση του ερωτήματος σταματά. Σε περίπτωση που η τιμή βρεθεί σε μία δενδρική δομή, τότε ο κόμβος αυτός γίνεται ο κόμβος-συντονιστής (κόμβος coordinator) της flood διαδικασίας. Εάν περισσότερες από μια τιμές βρεθούν στον ίδιο κόμβο, τότε το ερώτημα επιλύεται στο "εικονικό" δακτυλίδι της διάστασης με την υψηλότερη προτεραιότητα. Οι πιθανές περιπτώσεις για κάποια τιμή που βρέθηκε είναι δύο: είτε ανήκει σε κάποιο επίπεδο πάνω από το pivot level είτε σε κάποιο επίπεδο κάτω από το pivot level. Ο αναφερόμενος κόμβος δε θα καταστεί κόμβος coordinator, όταν μια τιμή του ερωτήματος βρεθεί σε μία ή περισσότερες πλειάδες του local database. Ωστόσο, σε αυτόν τον κόμβο έχει βρεθεί αρκετή πληροφορία στις αποθηκευμένες πλειάδες, ώστε να προωθηθεί το ερώτημα είτε στο root key αυτής της τιμής είτε στο pivot key του αντίστοιχα. Οι πλειάδες που βρέθηκαν και απαντούν το ερώτημα (ή τα aggregated facts) συμπεριλαμβάνονται και αυτές στο μήνυμα κατά τη προώθηση του ερωτήματος. Αν εξαιρεθεί αυτό το επιπρόσθετο βήμα, η διαδικασία για την επίλυση του ερωτήματος συνεχίζει σύμφωνα με την διαδικασία που περιγράφεται στη συνέχεια χωρίς περαιτέρω αλλαγές.

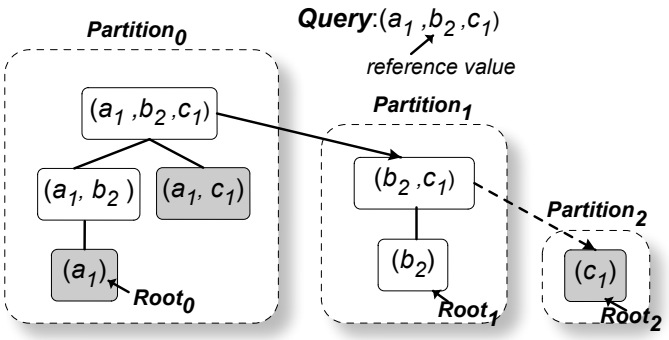
Αν θεωρηθεί ότι η τιμή που βρέθηκε ανήκει σε ένα επίπεδο κάτω από το pivot level, τότε δεν υπάρχουν άλλα δέντρα με τη συγκεκριμένη τιμή. Ο κόμβος είτε στέλνει το αποτέλεσμα στον κόμβο initiator του ερωτήματος (εάν το ερώτημα περιλαμβάνει μόνο μία τιμή ή η ευρισκόμενη τιμή ανήκει στην primary διάσταση) ή προωθεί το ερώτημα κατά μήκος των συνδέσμων του

(links) στους κόμβους της primary διάστασης αποκλείοντας αυτούς που έχουν ήδη ερωτηθεί. Η περιγραφόμενη στρατηγική για τη δεύτερη κατηγορία των exact match ερωτημάτων ακολουθείται. Οι κόμβοι με τα primary keys απαντάνε με τις σχετικές πλειάδες ή το aggregated fact. Τα αποτελέσματα αυτά συλλέγονται από τον κόμβο coordinator και αποστέλλονται στον κόμβο που έθεσε το ερώτημα.

Στην περίπτωση όπου η τιμή που βρέθηκε ανήκει σε ένα επίπεδο πάνω από το pivot level, τότε μπορεί να υπάρχουν και άλλα δέντρα με την ίδια τιμή, ακόμα και αν βρέθηκε ένα τέτοιο δέντρο. Για παράδειγμα, ένα flood μήνυμα για την τιμή a_1 του Σχήματος 4.4 φτάνει στον κόμβο Nod_1, αλλά μπορεί να υπάρχουν και άλλοι κόμβοι με την τιμή a_1 και διαφορετικά pivot keys. Εντούτοις, είναι βέβαιο ότι η συγκεκριμένη τιμή δεν αποθηκεύεται σε κάποιο δέντρο με διαφορετικό root key. Λαμβάνοντας υπόψη αυτές τις παρατηρήσεις, το flood μήνυμα προωθείται στον κόμβο με το αντίστοιχο root key, ο οποίος γίνεται ο κόμβος coordinator της διαδικασίας από εδώ και στο εξής. Ο κόμβος αυτός προωθεί το flood ερώτημα στους κόμβους που γνωρίζει ότι αποθηκεύουν τα pivot keys του εξαιρώντας τους κόμβους που έχουν ήδη ερωτηθεί. Εάν η τιμή που βρέθηκε ανήκει στην primary διάσταση ή το ερώτημα δεν περιλαμβάνει άλλες τιμές, τότε οι κόμβοι απαντούν με τις σχετικές πλειάδες ή το aggregated fact αντίστοιχα. Διαφορετικά, κάθε κόμβος συμπεριλαμβάνει στην απάντηση του κάθε σχετικό αποτέλεσμα που μπορεί να βρέθηκε στο local database του και ένα σύνολο από υποψήφια links που διατηρούν τα pivot key(s) της αναφερόμενης τιμής προς την primary διάσταση. Μόλις ληφθούν όλα τα αποτελέσματα, ο κόμβος coordinator συγχωνεύει τα links και εξαιρεί τους κόμβους που έχουν ήδη ερωτηθεί. Τελικά, τα local databases από τους εναπομείναντες κόμβους ρωτιούνται, επιστρέφονται τα αποτελέσματα, συγχωνεύονται με τα ήδη συλλεγμένα και αποστέλλονται στον κόμβο initiator.

4.4.3 Μία Προσέγγιση που Διαμορφώνεται από τα Ερωτήματα για τον Προ-υπολογισμό τους

Σε πολλά αποθηκευτικά συστήματα πολλαπλών διαστάσεων είναι σύνηθης πρακτική να προ-υπολογίζονται διαφορετικές όψεις (views - GROUP-BYs) για τη βελτίωση των χρόνων απόκρισης. Για ένα σύνολο δεδομένων R που περιγράφεται από d διαστάσεις με ιεραρχίες ενός επιπέδου, μία όψη (view) κατασκευάζεται από μία συνάθροιση στο R για ένα υποσύνολο των δεδομένων σύμφωνα με κάποιες ιδιότητες με αποτέλεσμα τη δημιουργία 2^d διαφορετικών, πιθανών όψεων (δηλαδή προκύπτει εκθετική χρονική και χωρική πολυπλοκότητα). Ο αριθμός των επιπέδων σε κάθε διάσταση επιφέρει ακόμα μεγαλύτερη πολυπλοκότητα. Στο *LinkedPeers*, μία προσέγγιση που διαμορφώνεται από τα ερωτήματα (query-driven approach) θεωρείται για την αντιμετώπιση του προβλήματος των όψεων: Η επιλογή των views που θα προ-υπολογιστούν καθοδηγείται από τα ερωτήματα, καθώς αξιοποιείται η διαδικασία της επίλυσης των ερωτημάτων



Σχήμα 4.5: *Όλοι τα πιθανά view identifiers για ένα ερώτημα που αφορά τιμές από 3 διαστάσεις*

για τον υπολογισμό μερικών αποτελεσμάτων από διάφορες όψεις που θεωρούνται πιθανές να χρειαστούν στο μέλλον και διατηρούνται "partial materialized views" με κατανεμημένο τρόπο.

Το Σχήμα 4.5 παριστάνει όλους τις πιθανούς συνδυασμούς τιμών του ερωτήματος (a_1, b_2, c_1) που σχετίζεται με τα δεδομένα που απεικονίζονται στο Σχήμα 4.4. Οι ιδιότητες που συμμετέχουν ανήκουν στα επίπεδα $\{\ell_{01}, \ell_{12}, \ell_{31}\}$ αντίστοιχα. Κάθε συνδυασμός των τιμών αποτελείται από ένα υποσύνολο των ιδιοτήτων των διαστάσεων σε φθίνουσα σειρά. Ένας πιθανός συνδυασμός τιμών που μπορεί να ερωτηθεί αντιστοιχίζεται σε ένα "view identifier" που αποτελείται από τις αντίστοιχες τιμές. Όταν ένα view identifier (ή συνδυασμός αντίστοιχα) "υλοποιείται", τότε το αποτέλεσμα για αυτόν το συνδυασμό των επερωτούμενων τιμών υπολογίζεται και αποθηκεύεται για μελλοντική χρήση. Για παράδειγμα, το view identifier (a_1, c_1) στο Σχήμα 4.5 αποθηκεύει το αποτέλεσμα του ερωτήματος $(a_1, *, c_1)$. Επιπλέον, κάθε view identifier στο i-το επίπεδο της δενδρικής δομής του Σχήματος 4.5 εξάγεται από τον πρόγονο του στο *(i-1)*-το επίπεδο με την παράλειψη της συμμετοχής μίας διάστασης κάθε φορά. Όταν η τιμή μίας διάστασης παραλείπεται σε ένα view identifier, τότε θεωρείται ότι η τιμή της είναι η '*'-τιμή. Οι identifiers που έχουν καταγραφεί ήδη στο αριστερό μέρος του δέντρου παραλείπονται.

Έστω ότι $S_i \subset S$ είναι το υποσύνολο των view identifiers που αρχίζουν με την τιμή της ιδιότητας που ορίζεται στη διάσταση d_i. Το υποσύνολο των συγκεκριμένων view identifiers καλείται $Partition_{d_i}$ και η διάσταση που συμμετέχει σε όλους τους identifiers σε αυτό το υποσύνολο καλείται $Root_{d_i}$. Στο Σχήμα 4.5, το $Partition_0$ αποτελείται από όλες τα view identifiers που περιέχουν το a_1, το οποίο είναι το $Root_0$, ενώ το a_1 δεν εμφανίζεται σε κανένα identifier των υπόλοιπων partitions.

Σύμφωνα με τη στρατηγική που ακολουθείται κατά τη διάρκεια ενός flooding είναι βέβαιο ότι έχουν λάβει το ερώτημα όλοι οι κόμβοι με δέντρα που περιέχουν την τιμή που βρέθηκε (στη συνέχεια αναφέρεται ως *reference value*) και βάσει της οποίας έγινε η επίλυση του ερωτήματος. Συνεπώς, μπορεί να εξαχθεί με βεβαιότητα το συμπέρασμα ότι δεν υπάρχουν επιπλέον κόμβοι

με πλειάδες που περιέχουν το reference value. Η υπόθεση αυτή δεν είναι έγκυρη για τις υπόλοιπες τιμές που περιλαμβάνονται στο ερώτημα. Η παρατήρηση αυτή είναι σημαντική για τον προσδιορισμό των συνδυασμών που μπορούν να γίνουν materialized και να αποθηκευτούν με κατανεμημένο τρόπο.

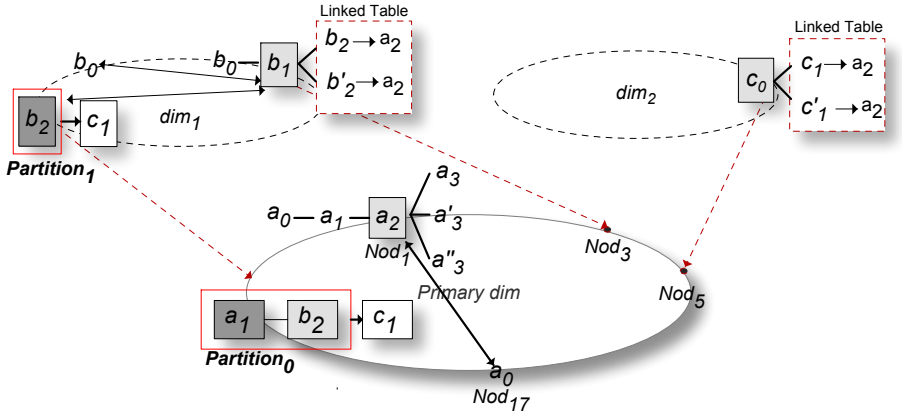
Έστω ότι S είναι το σύνολο όλων των 2^d identifiers. Μπορεί να προκύψει το συμπέρασμα ότι μόνο ένα υποσύνολο $S_{partial} \subset S$ από view identifiers μπορεί να γίνει πλήρως materialized, δηλαδή μόνο τα identifiers των συνδυασμών που περιλαμβάνουν το reference value. Λαμβάνοντας υπόψη το Σχήμα 4.5, θεωρείται ότι το flood ερώτημα για το συνδυασμό (a_1, b_2, c_1) φτάνει στον κόμβο Nod_3 και ότι το reference value είναι το b_2. Το ερώτημα θα προωθηθεί στον κόμβο Nod_1 για να επιλυθεί τελικώς. Όμως δεν εξασφαλίζεται ότι δεν υπάρχουν άλλοι κόμβοι που αποθηκεύουν πλειάδες με τις τιμές a_1 ή c_1. Συμπερασματικά, το $S_{partial}$ αποτελείται μόνο από τα view identifiers στα *μη-γκρι* κουτιά, τα οποία μπορούν να γίνουν materialized.

Αναλυτικότερα, ο υπολογισμός των μερικών όψεων εκτελείται στους κόμβους του *LinkedPeers* με τον ακόλουθο τρόπο: Κάθε κόμβος που επιστρέφει ένα aggregated fact για το flood ερώτημα, υπολογίζει επίσης τους διαθέσιμους view identifiers του $S_{partial}$ που αποθηκεύονται στο local database του. Λόγω της flooding στρατηγικής θα βρεθεί σίγουρα κάθε κόμβος με δέντρα που περιέχουν το reference value. Σύμφωνα με αυτή τη διαδικασία, τα ακόλουθα συμπεράσματα μπορούν να εξαχθούν:

- Το $S_{partial}$ μπορεί να αποτελείται μόνο από view identifiers που ανήκουν στα $Partition_{d_0}$, $Partition_{d_1}, .., Partition_{d_{ref}}$, όπου το $Root_{d_{ref}}$ του $Partition_{d_{ref}}$ είναι το reference value που χρησιμοποιήθηκε για την επίλυση του flood query.

- Εάν το ερώτημα πλημμυριστεί σε όλους τους κόμβους του δικτύου, τότε όλοι οι συνδυασμοί των τιμών που ερωτήθηκαν μπορούν να υπολογιστούν και συνεπώς προκύπτουν $2^d - 1$ συνδυασμοί (το 'ALL' δε γίνεται materialized), εάν το ερώτημα δεν περιέχει την '*'-τιμή σε καμία διάσταση. Στην περίπτωση που υπάρχουν '*'-τιμές, ο αριθμός των view identifiers είναι $2^{d-n} - 1$, όπου n είναι ο αριθμός των '*'-τιμών. Εάν εφαρμόζεται η περιγραφόμενη στρατηγική για την ελαχιστοποίηση των κόμβων που προωθείται το ερώτημα για την επίλυση του, τότε μόνο οι συνδυασμοί που περιέχουν το reference value μπορούν να υπολογιστούν. Παρόλα αυτά, λαμβάνοντας υπόψη τον τύπο του σύνολου των δεδομένων (αριθμός διαστάσεων, αριθμός tuples, κλπ.), τον τύπο του φορτίου των ερωτημάτων (μέσος αριθμός των '*'-τιμών ανά ερώτημα) και τις προδιαγραφές του συστήματος (όπως την επιθυμητή κατανάλωση του εύρους ζώνης, το χώρο αποθήκευσης, κλπ.) διάφορες πολιτικές μπορούν να οριστούν για τον περιορισμό των aggregated αποτελεσμάτων που θα υπολογιστούν.

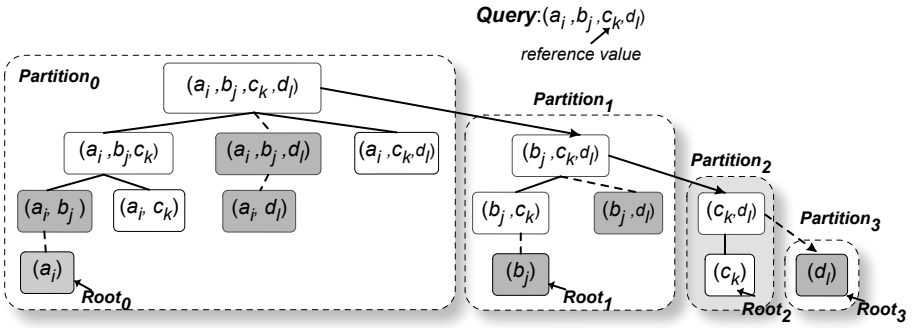
Μόλις παραληφθούν όλα τα αποτελέσματα, ο κόμβος coordinator συγχωνεύει τα aggregated facts για κάθε view identifier. Στη συνέχεια, υπολογίζει το hash value του κάθε $Root_{d_j}$ και εισάγει κάθε $Partition_{d_j}$ ($j \in [0, d_{ref}]$) στην επικάλυψη. Ο κόμβος που είναι υπεύθυνος για το $Root_{d_{ref}}$ δημιουργεί επίσης *δείκτες* προς τις τοποθεσίες των δενδρικών δομών για την προώθηση ενός ερωτήματος που δεν μπορεί να επιλυθεί από τα αποθηκευμένα materialized view identifiers. Η ιδέα για το διαχωρισμό των partitions βασίζεται στο γεγονός ότι οι αποθηκευμένοι συνδυασμοί πρέπει να εντοπίζονται με το ελάχιστο κόστος σε μηνύματα, δηλαδή με το DHT lookup. Δεδομένου ότι ένα ερώτημα αποσυναρμολογείτε στις επιμέρους τιμές του και θέτονται διαδοχικά ερωτήματα σύμφωνα με την προτεραιότητα των διαστάσεων, κάθε identifier αποθηκεύεται στη διάσταση με την υψηλότερη προτεραιότητα για τις τιμές τους.



Σχήμα 4.6: *Κατανομή των view identifiers που έχουν γίνει materialized στους κόμβους του LinkedPeers*

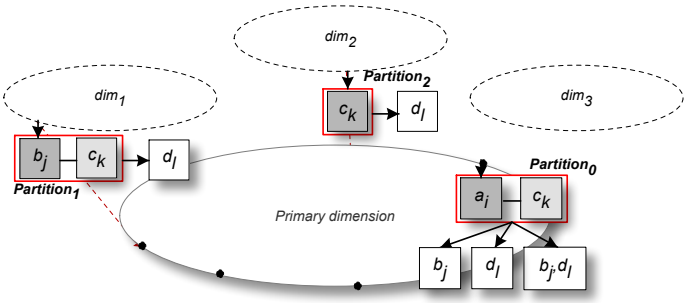
Αν και θα μπορούσε να χρησιμοποιηθεί οποιαδήποτε προσέγγιση που ακολουθείται στα σχεσιακά σχήματα για την αποθήκευση των aggregated facts, απλές 'linked-listed' δομές διατηρούνται στο *LinkedPeers* για την αποθήκευση των διαφορετικών view identifiers με τα αντίστοιχα facts. Όπως φαίνεται στο Σχήμα 4.6, η αποθήκευση των view identifiers του Σχήματος 4.5 γίνεται στους κόμβους που είναι υπεύθυνοι για τις τιμές που εμφανίζονται στα "σκουρόχρωμα γκρι" πλαίσια. Όλα τα ερωτήματα φτάνουν στον κόμβο που είναι υπεύθυνος για το $Root_0$ (δηλαδή για το a_1) πρέπει να περιλαμβάνουν επισης και το $Root_{d_{ref}}$, που είναι το b_2. Ο συνδυασμός των τιμών που ένα ερώτημα πρέπει να περιέχει τουλάχιστον, ώστε να επιλυθεί από κάποιο view identifier ενός partition σημειώνεται με κόκκινο πλαίσιο. Το ερώτημα μπορεί να περιέχει μία ή περισσότερες τιμές που περιέχονται στα άσπρα κουτιά. Για παράδειγμα, τα ερωτήματα $(a_1, b_2, *)$ και (a_1, b_2, c_1) μπορούν να απαντηθούν απευθείας από τα υπολογισμένα αποτελέσματα που έχουν αποθηκευτεί στον κόμβο που είναι υπεύθυνος για το $Partition_0$, σύμφωνα και με το Σχήμα 4.5. Το κλειδί που χρησιμοποιείται για την ανάθεση των δύο αυτών view identifiers στον κατάλληλο κόμβο είναι το hashed value του a_1. Τα Σχήματα 4.7 και 4.8 παρουσιάζουν ένα διαφορετικό παράδειγμα, όπου τα δεδομένα του συστήματος περιγράφονται από τέσσερις διαστάσεις. Το

ερώτημα που θέτεται είναι το (a_i,b_j,c_k,d_l), όπου καμία από αυτές τιμές δεν ανταποκρίνεται σε κάποιο pivot value. Οι πιθανοί συνδυασμοί και αυτοί που τελικά θα υπολογιστούν (δηλαδοί αυτοί που περιέχονται σε "μη-γκρι" κουτιά) φαίνονται στο Σχήμα 4.7, αφού το ερώτημα επιλύθηκε με χρήση της τιμής c_k ως reference value. Μία ενδεικτική απεικόνιση της κατανομής των view identifiers που έχουν γίνει materialized παρουσιάζεται στο Σχήμα 4.8.



Σχήμα 4.7: *Όλοι οι πιθανοί view identifiers για ένα ερώτημα που περιέχει τιμές από τέσσερις διαστάσεις.*

Οι δείκτες που δημιουργούνται και τα view identifiers είναι soft-state έτσι ώστε να ελαχιστοποιείται η πλεονάζουσα πληροφορία. Αυτό επιτυγχάνεται με τη λήξη τους μετά από μία προκαθορισμένη χρονική περίοδο (Time-to-Live or *TTL*). Κάθε φορά που ένας δείκτης χρησιμοποιείται, το TTL του ανανεώνεται. Ο συγκεκριμένος περιορισμός εξασφαλίζει ότι οι αλλαγές στο σύστημα (π.χ. αλλαγές στις τοποθεσίες των δεδομένων, οι αφίξεις/αναχωρήσεις κόμβων, κλπ.) δεν θα έχουν ως αποτέλεσμα "λανθασμένους" δείκτες (stale indices) επιφέροντας αρνητικές επιπτώσεις στην αποτελεσματικότητα του lookup μηχανισμού. Επιπλέον, όταν ο αριθμός των δεικτών φτάνει ένα όριο I_{max}, τότε η δημιουργία ενός καινούργιου δείκτη έχει ως αποτέλεσμα τη διαγραφή ενός ή περισσοτέρων παλαιότερων δεικτών. Γενικώς, το σύστημα τείνει να διατηρεί του πιο "χρήσιμους" δείκτες, δηλαδή αυτούς που αναφέρονται στα δεδομένα που αναζητούνται συχνότερα.



Σχήμα 4.8: *Κατανομή των view identifiers που έχουν γίνει materialized στους κόμβους του LinkedPeers για το παράδειγμα των τεσσάρων διαστάσεων*

Κατά τη διαδικασία των ενημερώσεων των δεδομένων, υπάρχουν διαφορετικές εναλλακτικές που μπορούν να θεωρηθούν για την ενημέρωση των συνδυασμών που έχουν γίνει materialized, λόγω του γεγονότος ότι μπορεί να αλλάξουν τα aggregated facts. Αφενός, μπορεί να θεωρηθεί ότι αφού τα μερικώς materialized views είναι soft-state, μπορεί να αποφευχθεί η εξαντλητική ενημέρωση των τιμών τους, αναμένοντας την ανανέωση των τιμών τους μετά τη λήξη τους. Από την άλλη πλευρά, αν τα επιστρεφόμενα αποτελέσματα πρέπει να είναι ακριβή και υπάρχουν αυστηροί περιορισμοί για το θέμα αυτό, τότε μία στρατηγική μπορεί να εφαρμοστεί, όπου θα αναζητούνται οι τιμές των καινούργιων πλειάδων και θα ενημερώνονται οι συνδυασμοί που έχουν γίνει materialized καταλλήλως.

4.4.4 Indexed Ερωτήματα

Όταν ένα ερώτημα φτάνει σε έναν κόμβο που είναι υπεύθυνος για κάποιο δείκτη, τότε αναζητούνται τα αποθηκευμένα view identifiers (εάν υπάρχουν) για το συνδυασμό των τιμών που ρωτάται. Εάν ο συνδυασμός αυτός βρεθεί, τότε το aggregated fact επιστρέφεται στον κόμβο initiator. Σε περίπτωση που δεν έχει υπολογιστεί ο συγκεκριμένος συνδυασμός, αλλά ο δείκτης γνωρίζει του κόμβους που αποθηκεύουν τα δέντρα με τη δεικτοδοτημένη τιμή, τότε το ερώτημα προωθείται στα αντίστοιχα pivot keys. Εάν το ερώτημα είναι "απλό" ή βρέθηκε τιμή που ανήκει στην primary διάσταση, τότε το aggregated fact επιστρέφεται. Διαφορετικά, οι κόμβοι αυτοί επιστρέφουν τις τοποθεσίες στο primary dimension που σχετίζονται με τη δεικτοδοτημένη τιμή. Το ερώτημα προωθείται σε αυτούς τους κόμβους που επερωτούν τα local database τους. Μετά την επίλυση ενός indexed ερώτηματος με τη χρήση ενός αποθηκευμένου view identifier, εκτελείται η διαδικασία για να γίνουν materialized όλα τα δυνατά view identifiers όπως περιγράφηκε στην προηγούμενη Ενότητα.

Οι κόμβοι με τις πλειάδες που περιέχουν τη δεικτοδοτημένη τιμή πρέπει να ξέρουν την ύπαρξη ενός δείκτη. Η διπλή κατεύθυνση (bidirectionallity) των δεικτών προτείνεται έτσι ώστε να εξασφαλιστεί η συνοχή των δεδομένων, αν και οι δείκτες είναι προσωρινοί. Κατά τη διάρκεια μίας λειτουργίας προσαρμογής της δεικτοδότησης, οι τοποθεσίες των αποθηκευμένων πλειάδων αλλάζουν και οι δείκτες που συσχετίζονται με αυτές τις πλειάδες πρέπει είτε να ενημερωθούν είτε να διαγραφούν αποτρέποντας με αυτόν τον τρόπο τη δημιουργία "λανθασμένων" δεικτών (stale indices). Η επιλογή που ακολουθείται είναι να σβήνονται εντελώς οι προσωρινοί δείκτες, ώστε να αποφευχθεί η αύξηση της πολυπλοκότητας του συστήματος. Επιπλέον, η ύπαρξη λεπτομερούς πληροφορίας για έναν υπάρχοντα δείκτη δεν είναι καθοριστικής σημασίας για τον κόμβο που αποθηκεύει τις πλειάδες της δεικτοδοτημένης τιμής. Αρκεί μία απλή σήμανση των τιμών που είναι δεικτοδοτημένες έτσι ώστε να διαγραφούν οι αντίστοιχοι δείκτες αν χρειαστεί. Όμως σε αυτήν την περίπτωση μπορεί να γίνουν κάποιες πλεονάζουσες ενέργειες για τη διαγραφή μη

έγκυρων δεικτών που έχουν λήξει. Αν όμως δεν υπάρχουν περιορισμοί στο διαθέσιμο αποθηκευτικό χώρο και είναι προτιμότερη η επεξεργασία στο επίπεδο του κόμβου από την κατανάλωση εύρους ζώνης (bandwidth), τότε οι δεικτοδοτημένες τιμές μπορούν να συνοδεύονται από ένα timestamp. Σε αυτήν την περίπτωση, κάθε lookup για μία δεικτοδοτημένη τιμή ανανεώνει το TTL και στις δύο πλευρές του δείκτη και μόνο οι έγκυροι δείκτες διαγράφονται όταν λαμβάνει χώρα μία λειτουργία προσαρμογής της δεικτοδότησης. Οι όψεις που έχουν δημιουργηθεί δεν αποθηκεύουν πληροφορία σχετικά με τις τοποθεσίες των δέντρων και για αυτό το λόγο οι όψεις είναι μόνο soft-state.

4.5 Προσαρμογή της Δεικτοδότησης σύμφωνα με τα Ερωτήματα

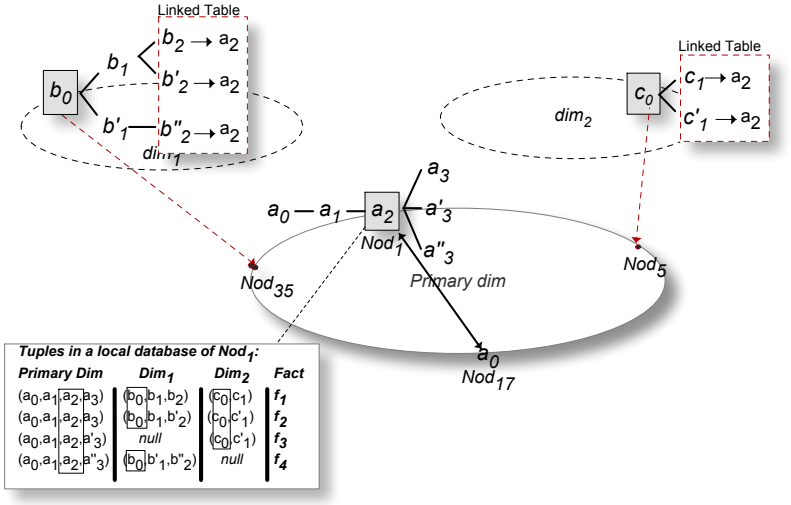
Ένα σημαντικό χαρακτηριστικό του συστήματος είναι ότι προσαρμόζει δυναμικά το επίπεδο δεικτοδότησης "στο επίπεδο του κόμβου" βάσει των εισερχόμενων ερωτημάτων. Για να επιτευχθεί αυτό, χρησιμοποιούνται και πάλι οι δύο διαδικασίες προσαρμογής της δεικτοδότησης που αφορούν την επιλογή του pivot level: *Roll-up* προς γενικότερα επίπεδα της ιεραρχίας και *drill-down* προς χαμηλότερα επίπεδα σε σχέση με το pivot level.

Η βασική ιδέα πίσω από τη διαδικασία για τη λήψη της απόφασης για την προσαρμογή της δεικτοδότησης βασίζεται στο γεγονός ότι ένας κόμβος είναι ικανότερος να ανιχνεύσει εάν η ζήτηση για τις τιμές ενός επιπέδου $\ell_{ij} > pivotlevel$ (όπου το *pivotlevel* συμβολίζει το pivot level της συγκεκριμένης ιεραρχίας για την i-τη διάσταση) είναι μεγαλύτερη και συνεπώς να προχωρήσει σε μία re-indexing λειτουργία προς αυτό το επίπεδο. Αντιθέτως, ο κόμβος αυτός πρέπει να συνεργαστεί με τους υπόλοιπους κόμβους που αποθηκεύουν μία τιμή για ένα επίπεδο $\ell_{ij} < pivotlevel$ ώστε να αποκτήσει μία ευρύτερη εικόνα και να αποφασίσει αν θα είναι επικερδής μία re-indexing λειτουργία προς αυτό το επίπεδο στα εμπλεκόμενα δέντρα. Ως εκ τούτου, ένας κόμβος έχει επαρκή πληροφορία για να αποφασίσει εάν μία λειτουργία drill-down θα ευνοήσει τις τιμές ενός δέντρου. Όμως, σε μία διαδικασία roll-up προς ένα επίπεδο $\ell_{ij} < pivotlevel$ εμπλέκονται όλοι οι κόμβοι που αποθηκεύουν δέντρα με τη συγκεκριμένη τιμή σε αυτό το επίπεδο. Η απόφαση για μία πιθανή re-indexing λειτουργία λαμβάνεται σύμφωνα με στατιστικά που συλλέγονται από τα εισερχόμενα ερωτήματα στα δέντρα που είναι υπεύθυνα για την τιμή βάσει της οποίας εξελίχθηκε η διαδικασία για την επίλυση ενός ερωτήματος, σύμφωνα με τη διαδικασία που αναλύθηκε στην Ενότητα 3.5. Εφόσον μερικά ερωτήματα επιλύονται κατευθείαν από τον κόμβο που διατηρεί ένα δείκτη και δεν προωθούνται περαιτέρω στον κόμβο (κόμβους) που αποθηκεύουν τις πλειάδες, μερική στατιστική πληροφορία κρατείται προσωρινά και στους δείκτες. Η πληροφορία αυτή ωθείται στους κόμβους με τις πλειάδες και συγχωνεύεται με την πληροφορία που διατηρούν αυτοί μόλις μία επόμενη indexed ερώτηση δε μπορεί να επιλυθεί τοπικά και χρειαστεί να προωθηθεί σε αυτούς. Τα ερωτήματα που απαντιούνται με τη χρήση των view identifiers δε συνυπολογίζονται στις αποφάσεις για τις re-indexing λειτουργίες, δεδομένου ότι

επιλύονται κατευθείαν από τον κόμβο που αποθηκεύει τα view identifiers χωρίς να επιβαρύνουν το κόστος επεξεργασίας. Ο κύριος στόχος προς επίτευξη είναι η αύξηση των ερωτημάτων που επιλύονται ως exact matches σε κάθε διάταση.

Κάθε φορά που αναζητείται η τιμή μιας δενδρικής δομής, η αντίστοιχη στατιστική πληροφορία ενημερώνεται, όπως περιγράφηκε στην Ενότητα 3.5. Η διαδικασία για την απόφαση μίας πιθανής re-indexing λειτουργίας μπορεί να ενεργοποιηθεί μετά από ένα indexed ερώτημα που επιλύθηκε χωρίς τη χρήση ενός view που έχει γίνει materialized ή ενός flood ερωτήματος και μόνο για το reference value. Όσον αφορά τα indexed ερωτήματα, ένας κόμβος που αποθηκεύει το δέντρο με τη δεικτοδοτημένη τιμή ελέγχει για την αναγκαιότητα μίας re-indexing λειτουργίας όταν λάβει έναν προκαθορισμένο αριθμό από indexed ερωτήματα. Όταν βρεθεί μία σχετική δενδρική δομή κατά την επίλυση ενός flood ή indexed ερωτήματος, τότε η αντίστοιχη στατιστική πληροφορία της ελέγχεται για να βρεθεί εάν συνιστάται μία re-indexing λειτουργία. Εάν αποφασιστεί ότι απαιτείται ένα drill-down ή φαίνεται πιθανή η ανάγκη για ένα roll-up, τότε αυτό το ενδεχόμενο περιλαμβάνεται στην απάντηση του κόμβου. Λαμβάνοντας υπόψη ότι όλες οι δενδρικές δομές για το reference value εντοπίζονται, μία re-indexing απόφαση εξετάζεται μόνο για αυτήν την τιμή και όχι για όλες τις τιμές που εμπεριέχονται στο ερώτημα. Η διαδικασία για να αποφασιστεί εάν συνιστάται μία re-indexing λειτουργία εκτελείται όπως ορίζουν οι αλγόριθμοι που προτάθηκαν στην Ενότητα 3.5. Παρόλα αυτά, σημαντικές αλλαγές έγιναν για την προσαρμογή των re-indexing λειτουργιών για τα δεδομένα πολλαπλών διαστάσεων λόγω των νέων απαιτήσεων που δημιουργήθηκαν από την ύπαρξη των links ανάμεσα στις διαστάσεις. Εάν μία re-indexing λειτουργία δεν απαιτείται για ένα flood ερώτημα, τότε η μοναδική ενέργεια που πραγματοποιείται είναι η δημιουργία των soft-state δεικτών και το materialization των view identifiers.

Roll-up: Σε γενικές γραμμές, εάν ένας κόμβος εντοπίσει αυξημένη ζήτηση για κάποια τιμή που αντιστοιχεί σε κάποιο επίπεδο πάνω από το pivot level, ξεκινάει τη διαδικασία για να αποφασίσει εάν ένα roll-up προς αυτό το επίπεδο θα είναι ευεργετικό (επικοινωνώντας με τους υπόλοιπους κόμβους που έχουν τη συγκεκριμένη τιμή). Για αυτόν το λόγο, ο κόμβος επισημαίνει στον κόμβο που συλλέγει τα αποτελέσματα ότι χρειάζεται να εξεταστεί το ενδεχόμενο μίας πιθανής roll-up λειτουργίας προς το επίπεδο που ρωτάται. Όταν ο κόμβος coordinator (εάν επιλύεται ένα flood ερώτημα) ή ο κόμβος που είναι υπεύθυνος για κάποιο δείκτη ενημερώνεται για την ανάγκη ενός πιθανού roll-up, τότε ξεκινάει τη συλλογή στατιστικής πληροφορίας από όλα τα δέντρα που περιέχουν την ερωτηθείσα τιμή. Κατόπιν, αποφασίζει εάν χρειάζεται να εκτελεστεί roll-up και μία θετική απόφαση οδηγεί στην επανεισαγωγή όλων των δέντρων με το καινούργιο hash value στην επικάλυψη, ενώ τα δέντρα με τα παλιά pivot values διαγράφονται. Κατά τη διάρκεια ενός roll-up, ένας ή περισσότεροι κόμβοι εισάγουν ξανά τα δέντρα τους, το οποία καταλήγουν στον κόμβο που είναι υπεύθυνος για το καινούργιο pivot key. Εάν η τιμή με την οποία

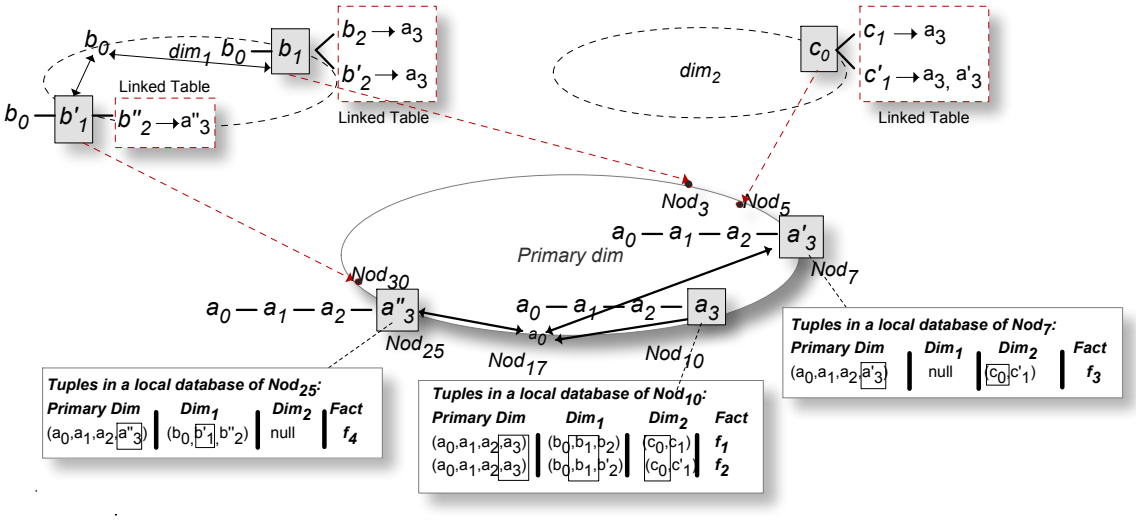


Σχήμα 4.9: *Αναδιοργάνωση των εμφανιζόμενων δενδρικών δομών της διάστασης dim_1 μετά από ένα roll-up προς το επίπεδο ℓ_0*

σχετίζεται το roll-up ανήκει στην primary διάσταση, τότε όλες οι σχετικές πλειάδες που βρίσκονται στο local database μεταφέρονται στο νέο κόμβο. Επίσης, κάθε κόμβος ενημερώνει το root key για τα pivot keys που πρέπει να διαγραφούν και να αντικατασταθούν με το καινούργιο. Επιπλέον, διαγράφονται όλοι οι soft-state δείκτες προς οποιαδήποτε τιμή που περιέχεται σε αυτά τα δέντρα. Το root key περιμένει να λάβει μήνυμα για την ενημέρωση της λίστας των pivot keys από όλους τους κόμβους που συμμετέχουν στη roll-up λειτουργία και μετά αντικαθιστά τα παλαιά pivot keys και τους κόμβους με τα καινούργια δεδομένα. Εν τω μεταξύ, τα ερωτήματα που αφορούν οποιαδήποτε τιμή των δέντρων που συμμετέχουν στη roll-up λειτουγία απαντώνται από τους κόμβους που είναι υπεύθυνοι για τα παλαιά pivot keys. Τα υπάρχοντα view identifiers που περιέχουν κάποια από αυτές τις τιμές δεν επηρεάζονται, δεδομένου ότι η μετεγκατάσταση των δέντρων δεν επηρεάζει τα υπολογισμένα aggregated facts. Το τελικό βήμα είναι η ενημέρωση των links μεταξύ του primary και των secondary δακτυλιδιών, δεδομένου ότι τα links πρέπει να είναι έγκυρα για την επίλυση μελλοντικών ερωτημάτων. Εάν μία roll-up λειτουργία εκτελεστεί στο primary δακτυλίδι, τότε οι εγγραφές στα *Linked Tables* που περιέχουν τα παλαιά pivot keys πρέπει να ενημερωθούν. Κάθε πλειάδα στα local databases αποθηκεύει τα pivot levels στις secondary διαστάσεις. Συνεπώς, ο κόμβος που είναι υπεύθυνος για το καινούργιο pivot key βρίσκει τα leaf values και τα ενημερώνει για το καινούργιο pivot key του, έτσι ώστε να ενημερωθούν τα links προς τη primary διάσταση. Όταν εκτελείται ένα roll-up σε ένα secondary δακτυλίδι, τότε τα αντίστοιχα pivot levels των πλειάδων που περιέχουν το καινούργιο pivot key πρέπει και αυτά να ενημερωθούν. Για αυτόν το λόγο, ο κόμβος που είναι υπεύθυνος για τα νέα δέντρα στέλνει το pivot level σε όλα τα links προς τη primary διάσταση.

Στο Σχήμα 4.9, το αποτέλεσμα μίας roll-up λειτουργίας προς το ℓ_0 στο secondary δακτυλίδι της dim_1 αναπαριστάται. Τα εμπλεκόμενα δέντρα που αποθηκεύουν την τιμή b_0 πριν την εκτέλεση του roll-up φαίνονται στο Σχήμα 4.4. Έστω ότι ένα indexed ερώτημα για την τιμή b_0 δίνει το έναυσμα για μία roll-up λειτουργία που έχει ως αποτέλεσμα την επανεισαγωγή των δέντρων με pivot keys b_1 και b'_1 αντιστοίχως. Ο κόμβος που είναι υπεύθυνος για το νέο pivot key b_0 ενημερώνει επίσης τον κόμβο που είναι υπεύθυνος για το primary key a_2 ότι το καινούργιο pivot level για όλες τις πλειάδες που περιέχουν την τιμή b_0 στη διάσταση dim_1 είναι το ℓ_0.

Drill-down: Η διαδικασία του drill-down είναι λιγότερο πολύπλοκη, λόγω του γεγονότος ότι μόνο ένας κόμβος αποθηκεύει το δέντρο με τις συγκεκριμένες τιμές για αυτό το επίπεδο. Συμπερασματικά, ο κόμβος που απαντάει το ερώτημα μπορεί να αποφασίσει τοπικά εάν χρειάζεται κάποιο drill-down και να προχωρήσει στην εκτέλεση των απαιτούμενων ενεργειών. Στη συνέχεια, μετατρέπει το δέντρο σε πλειάδες που ομαδοποιούνται βάσει των καινούργιων pivot keys και τα εισάγει ξανά στο *LinkedPeers*. Σε περίπτωση που η τιμή βάσει της οποίας γίνεται το drill-down ανήκει στην primary διάσταση, οι πλειάδες από το local database μεταφέρονται και αυτές στους κόμβους που είναι υπεύθυνοι για τα νέα pivot keys. Μόλις ολοκληρωθεί η εισαγωγή των σχετικών δέντρων, ο κόμβος που είναι υπεύθυνος για το παλαιό pivot key ενημερώνει επίσης το root key σχετικά με τα καινούργια pivot keys και τις νέες τοποθεσίες των δέντρων. Οι υπάρχοντες δείκτες προς αυτά τα δέντρα σβήνονται. Τέλος, ο κόμβος που αποφάσισε το drill-down ενημερώνει τα links με τις άλλες διαστάσεις όπως περιγράφηκε και για τη roll-up διαδικασία.



Σχήμα 4.10: *Αναδιοργάνωση των εμφανιζόμενων δενδρικών δομών και των local databases της primary διάστασης μετά από ένα drill-down προς το επίπεδο ℓ_3*

Το Σχήμα 4.10 παρουσιάζει την τοποθέτηση των πλειάδων στην primary διάσταση μετά από μία λειτουργία drill-down προς το επίπεδο ℓ_3 του δέντρου με το pivot key a_2 που φαίνεται στο Σχήμα 4.4. Ένα flood ερώτημα για την τιμή a_3 θα καταλήξει στον κόμβο που είναι υπεύθυνος για

το pivot key a_2 και θα αποτελέσει το έναυσμα για να αποφασιστεί εάν χρειάζεται ένα drill-down προς το επίπεδο ℓ_3. Εάν αποφασιστεί το drill-down, ο κόμβος αυτός εισάγει ξανά τις πλειάδες του συγκεκριμένου δέντρου με τα καινούργια pivot keys a_3, a'_3 και a''_3 αντιστοίχως. Ο συγκεκριμένος κόμβος ενημερώνει επίσης το root key a_0 για τα καινούργια pivot keys και τα pivot keys των secondary διαστάσεων με τα οποίο διασυνδέεται. Για παράδειγμα, η εγγραφή $b_2 \rightarrow a_2$ στο Linked Table του pivot key b_1 αντικαθίσταται τώρα με την εγγραφή $b_2 \rightarrow a_3$.

Group-Drill-down: Όταν εξετάζεται το ενδεχόμενο ενός roll-up για ένα επίπεδο πάνω από το pivot level, τότε διερευνάται εάν ένα drill-down των εμπλεκόμενων δέντρων στο επίπεδο $l_{ij} \geq$ *MaxPivotLevel* ενδείκνυται. Είναι πιθανό να διαπιστωθεί ότι ένα επίπεδο $\ell_{ij} \geq$ *MaxPivotLevel* είναι το πιο δημοφιλές, ενώ αυτή η τάση δεν είχε διαπιστωθεί στις μερικές όψεις των εμπλεκόμενων δέντρων και το γεγονός αυτό τα απέτρεψε να εκτελέσουν ένα drill-down προς αυτό το επίπεδο. Η διαδικασία αυτή καλείται `Group-Drill-down`, δεδομένου ότι πολλαπλοί κόμβοι συμμετέχουν στο drill-down. Όλα τα δέντρα με την τιμή που ρωτήθηκε εκτελούν drill-down προς το καινούργιο pivot level. Εάν το καινούργιο pivot level είναι ίσο με το MaxPivotLevel, τα συγκεκριμένα δέντρα παραμένουν σε αυτό το επίπεδο.

4.6 Πειραματική Αξιολόγηση

4.6.1 Περιγραφή του Συστήματος

Το προτεινόμενο σύστημα αξιολογήθηκε πειραματικά υπό διάφορες συνθήκες. Η απόδοση του συστήματος μελετήθηκε σε μία υλοποίηση του συστήματος, όπου η τοπολογία και η δικτυακή επικοινωνία ανάμεσα στους κόμβους έγινε με τη χρήση του εξομοιωτή που παρέχεται από το FreePastry [Fre], αν και οποιαδήποτε DHT υλοποίηση μπορεί να χρησιμοποιηθεί. Το δίκτυο αναφοράς αποτελείται από 256 κόμβους, όπου οποιοσδήποτε από αυτούς μπορεί να επιλεγεί τυχαία για την έναρξη κάποιας ερώτησης.

Τα συνθετικά δεδομένα που χρησιμοποιούνται είναι δέντρα (διαφορετικά για κάθε διάσταση), όπου κάθε τιμή έχει ένα μοναδικό πρόγονο στο προηγούμενο επίπεδο και έναν σταθερό αριθμό από *mul* απογόνους στο αμέσως χαμηλότερο επίπεδο. Οι εισαγόμενες πλειάδες με τιμές για όλες τις διαστάσεις δημιουργούνται απο συνδυασμούς των leaf values των δέντρων της κάθε διάστασης και περιέχουν και ένα numerical fact. Εξ ορισμού, τα δεδομένα που εισάγονται στο σύστημα αποτελούνται από 1M πλειάδες που περιλαμβάνουν τιμές για τέσσερις διαστάσεις με ιεραρχίες των τριών επιπέδων. Ο αριθμός των διακριτών τιμών στο υψηλότερο επίπεδο είναι base $= 100$ με *mul*=10. Το επίπεδο που χρησιμοποιείται αρχικά ως pivot level είναι το ℓ_1 σε όλες τις διαστάσεις. Για τα φορτία ερωτημάτων ακολουθείται μια προσέγγιση τριών βημάτων: Αρχικά επιλέγεται το μέρος της αρχικής βάσης που θα στοχεύσει το ερώτημα (*TupleDist*). Στη συνέχεια, η πιθανότητα για μία διάσταση να μην έχει τιμή (δηλαδή να έχει την '*'-τιμή στο αντίστοιχο

ερώτημα) είναι P_{d*}. Τέλος, για τις διαστάσεις που ρωτιούνται επιλέγεται το επίπεδο που θα στοχεύσει η ερώτηση σύμφωνα με τη *levelDist* κατανομή. Στα πειράματα που παρουσιάζονται, η πόλωση των φορτίων ακολουθεί ομοιόμορφη, 80/20 και 90/10 κατανομή για το *TupleDist* και το *levelDist*, ενώ το P_{d*} αυξάνεται βαθμιαία από 0.1 στην primary διάσταση σε 0.8 στην τελευταία διάσταση. Τα ερωτήματα παράγονται με μέσο ρυθμό $1\frac{query}{time\_unit}$ σε συνολικό χρόνο εξομοίωσης που ισούται με 50k time units.

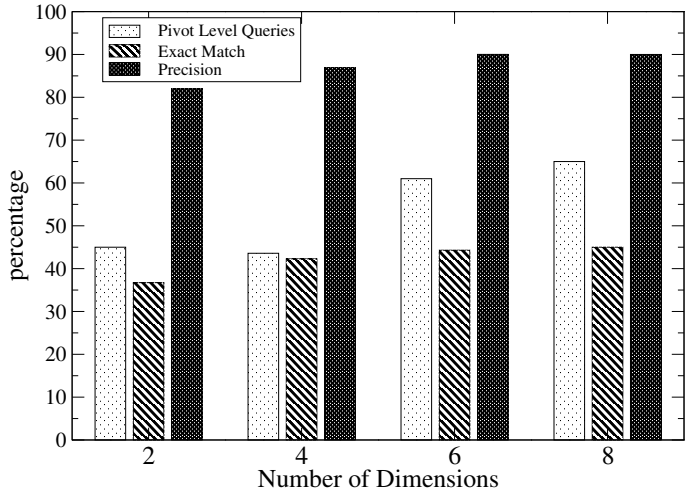
Ο βασικός στόχος της συγκεκριμένης ενότητας είναι να μελετηθεί η απόδοση του συστήματος για διαφορετικούς τύπους δεδομένων και φορτίων ερωτημάτων. Τα πειραματικά αποτελέσματα επικεντρώνονται στο precision που επιτυγχάνεται (δηλαδή στο ποσοστό των ερωτημάτων που επιλύονται χωρίς να γίνουν flood) και στο κόστος επίλυσης τους ως προς τα μηνύματα ανά ερώτημα.

4.6.2 Μελέτη της Απόδοσης του Συστήματος για Διαφορετικό Αριθμό Διαστάσεων και Επιπέδων

Στην πρώτη ομάδα πειραμάτων μελετάται η συμπεριφορά του συστήματος για δεδομένα που αποτελούνται από διαφορετικό αριθμό διαστάσεων ή για δεδομένα που ακολουθούν ιεραρχίες με διαφορετικό αριθμό επιπέδων. Τα ερωτήματα στοχεύουν ομοιόμορφα οποιαδήποτε πλειάδα επί του συνόλου των δεδομένων και οποιοδήποτε επίπεδο των ιεραρχιών σε κάθε διάσταση.

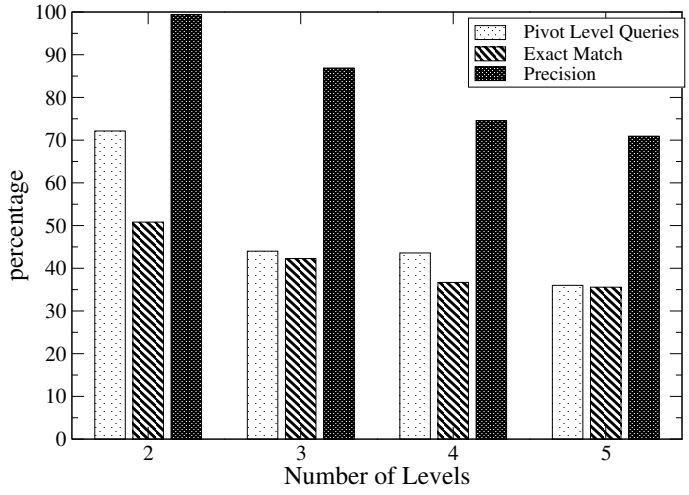
Κατά την εκτέλεση της πρώτης ομάδας πειραμάτων, ο αριθμός των διαστάσεων μεταβάλλεται, ενώ κάθε διάσταση αποτελείται από μία ιεραρχία με τρία επίπεδα. Το Σχήμα 4.11 επιδεικνύει το ποσοστό των ερωτημάτων του συνολικού φορτίου που περιλαμβάνουν τουλάχιστον ένα pivot value (που συμβολίζεται ως `Pivot Level Queries`) και το ποσοστό των ερωτημάτων που επιλύθηκαν ως exact match (που συμβολίζεται ως `Exact Match`). Επίσης επιδεικνύεται και το precision που καταγράφηκε κατά την εκτέλεση των φορτίων ερωτημάτων και το οποίο παραμένει πάνω από 85% για όλους τους τύπους των δεδομένων ανεξάρτητα από τον αριθμό των διαστάσεων. Τα ερωτήματα που δεν κατευθύνονται σε τιμές των pivot levels επιλύονται με τη χρήση δεικτών ή υλοποιημένων όψεων διατηρώντας το ποσοστό του precision υψηλό. Η διαφορά μεταξύ των exact match ερωτημάτων και των ερωτημάτων που περιέχουν τουλάχιστον μία τιμή που ανήκει σε pivot level οφείλεται στο γεγονός ότι σύμφωνα με τη στρατηγική που ακολουθείται για την επίλυση των ερωτημάτων είναι προτιμότερο να χρησιμοποιηθεί ένας δείκτης μίας διάστασης με υψηλότερη προτεραιότητα παρά να συνεχιστεί η προσπάθεια εντοπισμού ενός pivot value στις διαστάσεις με χαμηλότερες προτεραιότητες.

Στο Σχήμα 4.12 παρουσιάζονται τα αποτελέσματα για δεδομένα με τέσσερις διαστάσεις και μεταβλητό αριθμό επιπέδων για τις ιεραρχίες των διαστάσεων. Η μείωση στο precision που παρατηρείται (από 99% εώς 70%) οφείλεται στο γεγονός ότι η αύξηση των επιπέδων έχει αρνητικό αντίκτυπο στην πιθανότητα να τεθεί ένα ερώτημα που περιλαμβάνει μία τιμή που αντιστοιχεί



Σχήμα 4.11: *Ποσοστά σχετικά με τον τρόπο επίλυσης των ερωτημάτων για διαφορετικό αριθμό διαστάσεων, ενώ για κάθε διάσταση θεωρείται μια ιεραρχία τριών επιπέδων*

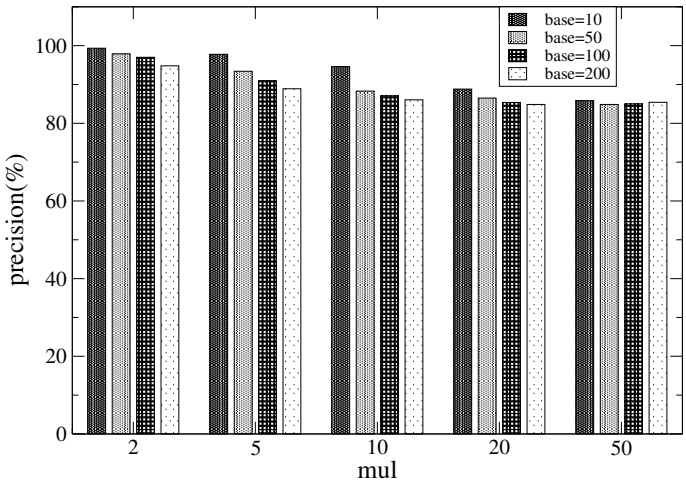
στο pivot level κάποιας διάστασης. Για αυτόν το λόγο, μειώνεται το ποσοστό των `Pivot Level Queries` και αντίστοιχα και το ποσοστό των `Exact Match` ερωτημάτων, όπως φαίνεται και στο Σχήμα 4.12. Η αύξηση των επιπέδων επηρεάζει επίσης και την επερώτηση τιμών που είναι ήδη δεικτοδοτημένες. Για αυτό το λόγο, η απόκλιση μεταξύ των `Pivot Level Queries` και των `Exact Match` ερωτημάτων είναι μεγαλύτερη, όταν οι ιεραρχίες έχουν δύο επίπεδα, όπου όλα τα ερωτήματα που περιλαμβάνουν τιμή στην primary διάσταση επιλύονται είτε με τη χρήση του pivot key είτε με τη χρήση του root index όπως υπαγορεύει η στρατηγική για την επίλυση των ερωτημάτων.



Σχήμα 4.12: *Ποσοστά σχετικά με τον τρόπο επίλυσης των ερωτημάτων ενός φορτίου δεδομένων τεσσάρων διαστάσεων, ενώ ο αριθμός των επιπέδων για τις ιεραρχίες της κάθε διάστασης μεταβάλλεται*

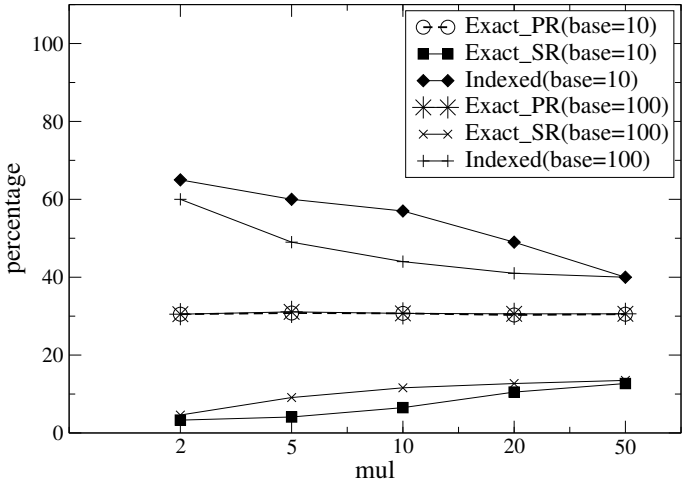
4.6.3 Μελέτη του Τρόπου Επίλυσης Ερωτημάτων για Διαφορετικούς Τύπους Συνόλων Δεδομένων

Στο πείραμα του Σχήματος 4.13 παρουσιάζεται το precision που επιτυγχάνεται για διάφορους τύπους συνόλων δεδομένων. Ο αριθμός των διακριτών τιμών στο υψηλότερο επίπεδο των δέντρων `base` και ο αριθμός των απογόνων του αμέσως χαμηλότερου επιπέδου `mul` μεταβάλλεται, ώστε να δημιουργούνται σύνολα δεδομένων με διαφορετική πυκνότητα. Τα `base` και `mul` επηρεάζουν τις συνδέσεις μεταξύ της primary και των secondary διαστάσεων, τον αριθμό των διακριτών τιμών σε κάθε επίπεδο και συνεπώς την πυκνότητα του συνόλου δεδομένων. Όπως φαίνεται στο Σχήμα 4.13, η αύξηση του `mul` επιφέρει μία μείωση του precision. Μία παρόμοια τάση παρατηρείται και για τα δεδομένα που έχουν παραχθεί με την ίδια τιμή για την παράμετρο `mul`, αλλά διαφορετικές τιμές για το `base`. Παρόλα αυτά, το *LinkedPeers* καταφέρνει να επιλύσει την πλειοψηφία των ερωτημάτων χωρίς να χρειαστεί να γίνει flooding.



Σχήμα 4.13: *Επίδραση των παραμέτρων `mul` και `base` στο precision που επιτυγχάνεται*

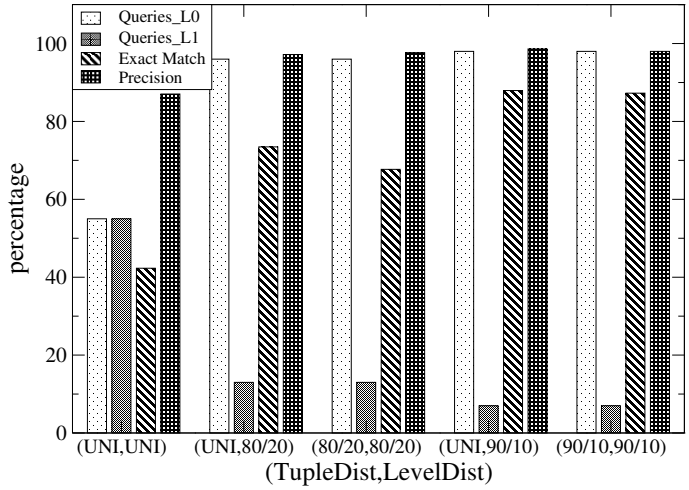
Το ποσοστό των exact match ερωτημάτων στην primary διάσταση (`Exact_PR`) παραμένει σταθερό για όλα τα σύνολα δεδομένων όπως φαίνεται στο Σχήμα 4.14, δεδομένου ότι εξαρτάται από το φορτίο ερωτημάτων. Ωστόσο, τα exact match ερωτήματα στις secondary διαστάσεις (`Exact_SR`) αυξάνουν όσο τα indexed ερωτήματα μειώνονται. Αυτό οφείλεται στο γεγονός ότι οι δείκτες της primary διάστασης αξιοποιούνται λιγότερο και περισσότερα ερωτήματα επιλύονται στα secondary δακτυλίδια.



Σχήμα 4.14: *Ποσοστό που αντιστοιχεί στους τρόπους επίλυσης των ερωτημάτων για διαφορετικά δεδομένα*

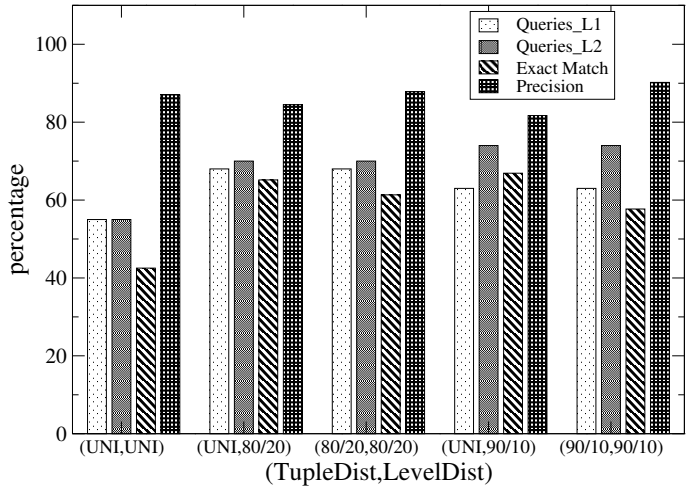
4.6.4 Μελέτη της Απόδοσης τους Συστήματος για Πολωμένα Φορτία Ερωτημάτων

Η προσαρμοστική συμπεριφορά του *LinkedPeers* μελετάται στα ακόλουθα πειράματα που εκτελέστηκαν για διάφορα φορτία ερωτημάτων που ακολουθούν είτε ομοιόμορφες είτε πολωμένες κατανομές. Πιο συγκεκριμένα, η πρώτη ομάδα πειραμάτων αφορά φορτία ερωτημάτων πολωμένα προς τα υψηλότερα επίπεδα των ιεραρχιών για διάφορα *TupleDist*. Ο αριθμός των ερωτημάτων που κατευθύνεται σε κάθε επίπεδο εξαρτάται από την τιμή του *levelDist*. Τα αποτελέσματα αυτών των πειραμάτων παρουσιάζονται στο Σχήμα 4.15 και επικεντρώνονται στο precision και στο ποσοστό των `Exact Match` ερωτημάτων που επιτεύχθηκε. Σε αυτό το Σχήμα περιλαμβάνονται επίσης τα ποσοστά των ερωτημάτων που περιέχουν τουλάχιστον μία τιμή στο ℓ_0 και τουλάχιστον μία τιμή στο ℓ_1, τα οποία συμβολίζονται με `Queries_L0` και `Queries_L1` αντιστοίχως. Όσο περισσότερο αυξάνεται η πόλωση, τόσο μεγαλύτερο γίνεται (σχεδον 100%) το precision που παρατηρείται. Στα πολωμένα φορτία, το ποσοστό των ερωτημάτων που συμβολίζονται με `Queries_L0` είναι σημαντικά μεγαλύτερο από το ποσοστό των ερωτημάτων που συμβολίζονται με `Queries_L1`, και για αυτό το λόγο είναι ευκολότερο για το σύστημα να αποφασίσει την εκτέλεση των απαραίτητων roll-up διαδικασιών προς το επίπεδο ℓ_0 σε κάθε διάσταση. Έτσι, ακόμα και αν η επιλογή του επιπέδου ℓ_1 δεν είναι η κατάλληλη για την επίλυση των ερωτημάτων χωρίς να προωθηθούν σε όλους τους κόμβους του συστήματος, το σύστημα καταφέρνει να προσαρμόσει κατάλληλα τα επίπεδα των ιεραρχιών που ρωτιούνται και να απαντήσει ένα μεγάλο μέρος των ερωτημάτων ως exact match ερωτήματα (συμβολίζονται ως `Exact Match`), αποδεικνύοντας την αποτελεσματικότητα των λειτουργιών re-indexing.



Σχήμα 4.15: *Precision και exact match ερωτήματα για πολωμένα φορτία ερωτημάτων προς τα υψηλότερα επίπεδα των ιεραρχιών και για διάφορους συνδυασμούς* (TupleDist,levelDist)

Το Σχήμα 4.16 απεικονίζει τα αντίστοιχα αποτελέσματα, όταν το φορτίο ερωτημάτων ευνοεί τα χαμηλότερα επίπεδα των ιεραρχιών. Στα φορτία που έχουν το ίδιο *TupleDist* αλλά είναι περισσότερο πολωμένα ως προς το *levelDist* παρατηρείται μία μείωση του precision. Αυτό οφείλεται στο γεγονός ότι τα χαμηλότερα επίπεδα των ιεραρχιών έχουν ένα σημαντικά μεγαλύτερο αριθμό τιμών. Όσο αυξάνει ο αριθμός των ερωτημάτων προς αυτές τις τιμές, αυξάνει και η πιθανότητα της αναζήτησης μίας μη δεικτοδοτημένης τιμής από ένα ερώτημα, μέχρι οι μηχανισμοί προσαρμογής της δεικτοδότησης να προσαρμόσουν τα pivot levels των δημοφιλών δέντρων κατάλληλα.

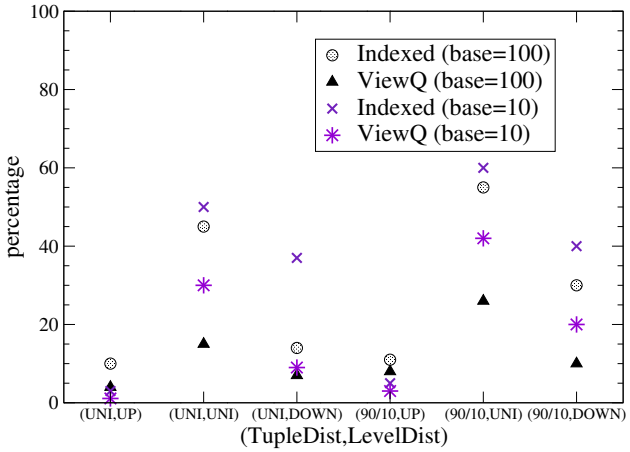


Σχήμα 4.16: *Precision και exact match ερωτήματα για πολωμένα φορτία ερωτημάτων προς τα χαμηλότερα επίπεδα των ιεραρχιών και για διάφορους συνδυασμούς* (TupleDist,levelDist)

4.6.5 Μελέτη της Χρήσης του Μερικού Materialization

Εκτός από τις λειτουργίες re-indexing, οι συνδυασμοί τιμών που έχουν γίνει materialized μπορούν να χρησιμοποιηθούν για την ελαχιστοποίηση του κόστους των ερωτημάτων. Στο επόμενο πείραμα, η προσέγγιση αυτή ελέγχεται για φορτία ερωτημάτων που στοχεύουν τα δεδομένα είτε ομοιόμορφα είτε πολωμένα (90/10) όσον αφορά το (*TupleDist*) και αντίστοιχα τα υψηλότερα επίπεδα (συμβολίζεται με `UP`) και τα χαμηλότερα επίπεδα (`DOWN`).

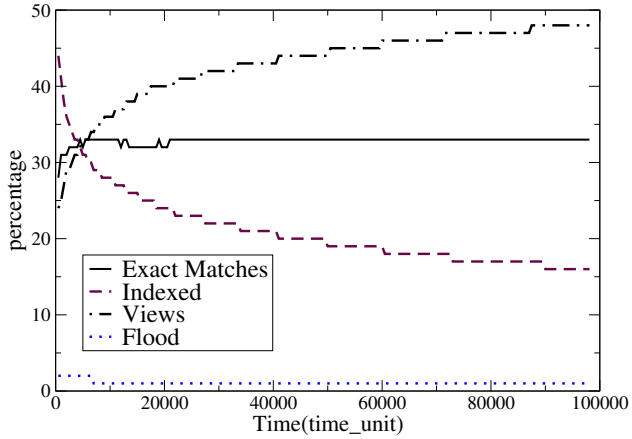
Όπως φαίνεται στο Σχήμα 4.17, το ποσοστό των ερωτημάτων που επιλύονται με τη χρήση ενός συνδυασμού που έχει γίνει materialized (`ViewQ`) αυξάνεται για τα φορτία ερωτημάτων που έχουν το ίδιο *levelDist* αλλά το *TupleDist* είναι 90/10 σε σύγκριση με αυτά που το *TupleDist* είναι UNI. Αυτή η συμπεριφορά οφείλεται στο εξής γεγονός: εάν ένα μέρος του συνόλου των δεδομένων ρωτάται κυρίως, τότε αυξάνεται και η πιθανότητα να ερωτηθεί ένα συνδυασμός που έχει γίνει materialized. Συνεπώς, περισσότερα ερωτήματα επιλύονται με τη χρήση αυτών των συνδυασμών. Επιπλέον, το ποσοστό ανάκτησης μίας απάντησης από έναν αποθηκευμένο συνδυασμό είναι υψηλότερο όταν το *levelDist* είναι ομοιόμορφο (`UNI`). Σε αυτήν την περίπτωση, οι μηχανισμοί re-indexing δεν μπορούν να προσαρμόσουν τα pivot levels κατάλληλα στα εισερχόμενα ερωτήματα και τα indexed ερωτήματα είναι συχνότερα αυξάνοντας τη χρήση των view identifiers.



Σχήμα 4.17: *Χρήση των συνδυασμών που έχουν γίνει materialized σε σύγκριση με τα ερωτήματα που επιλύονται ως indexed*

Το Σχήμα 4.18 παρουσιάζει τον τρόπο με τον οποίο επιλύονται τα ερωτήματα κατά τη διάρκεια της εξομοίωσης για το φορτίο ερωτημάτων του Σχήματος 4.17, το οποίο στοχεύει κυρίως ένα συγκεκριμένο μέρος του συνόλου των δεδομένων αλλά ομοιόμορφα όλα τα επίπεδα της ιεραρχίας. Ο συνολικός αριθμός των ερωτημάτων ανέρχεται σε 100K και το ποσοστό των ερωτημάτων που επιλύεται με τη χρήση view identifiers δε συνυπολογίζεται στο ποσοστό των indexed

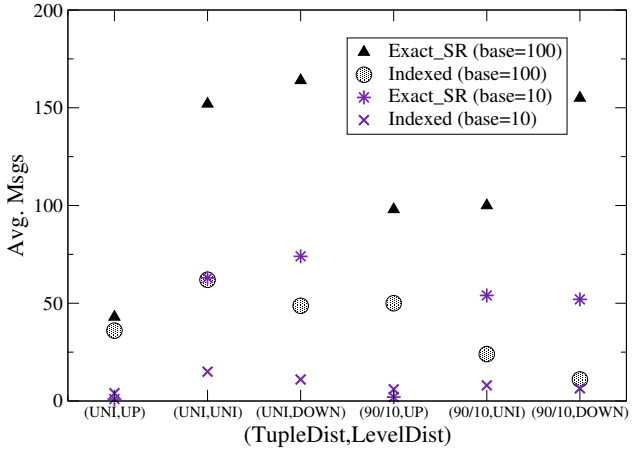
ερωτημάτων. Παρατηρείται ότι η χρήση των view identifiers αυξάνει όσο περνάει ο χρόνος και λιγότερα ερωτήματα χρειάζεται να προωθηθούν από τους δείκτες.



Σχήμα 4.18: *Τρόπος επίλυσης των ερωτημάτων κατά τη διάρκεια του χρόνου με έμφαση στα ερωτήματα που χρησιμοποιούν view identifiers που έχουν γίνει materialized και στα indexed ερωτήματα που προωθούνται*

4.6.6 Μελέτη του Κόστους Διάφορων Τρόπων Επίλυσης Ερωτημάτων

Το κόστος ενός ερωτήματος υπολογίζεται ως το κόστος των μηνυμάτων που χρειάζεται να εκδοθούν για την επίλυση του. Ένα ερώτημα που επιλύεται ως exact match στη primary διάσταση απαιτεί μόνο την εκτέλεση του DHT lookup μηχανισμού.



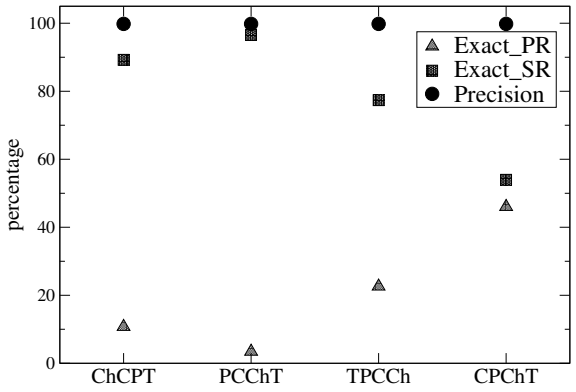
Σχήμα 4.19: *Μέσος αριθμός μηνυμάτων για τα exact match ερωτήματα στα secondary δακτυλίδια και για τα indexed ερωτήματα*

Το Σχήμα 4.19 παριστάνει το μέσο αριθμό μηνυμάτων μόνο για τα exact match ερωτήματα που επιλύθηκαν σε secondary διαστάσεις (`Exact_SR`), ενώ πρέπει να ληφθεί υπόψη ότι ο αριθμός

τους είναι σημαντικά μικρότερος (λιγότερο από 20% των ερωτημάτων σε όλες τις περιπτώσεις), και για τα indexed ερωτήματα (`Indexed`). Ο μέσος αριθμός των μηνυμάτων για τα `Exact_SR` εξαρτάται από τον τύπο των δεδομένων, δηλαδή από τον αριθμό των links μεταξύ των secondary pivot keys και των primary pivot keys. Όταν το φορτίο ερωτημάτων είναι πολωμένο προς τα υψηλότερα επίπεδα (`UP`), τότε τα μηνύματα μειώνονται, γιατί τα δημοφιλέστερα δέντρα έχουν εκτελέσει roll-up προς το επίπεδο ℓ_0. Συνεπώς, τα secondary keys συνδέονται με ένα μικρότερο αριθμό από primary keys. Η αντίθετη παρατήρηση ισχύει για τα (`DOWN`) φορτία ερωτημάτων.

4.6.7 Μελέτη της Απόδοσης του Συστήματος για Δεδομένα του APB benchmark

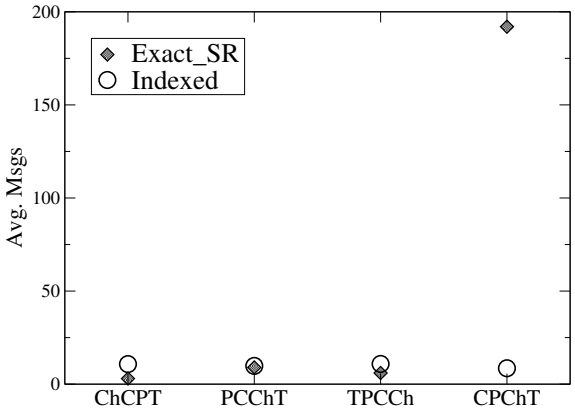
Η προσαρμοστικότητα του συστήματος δοκιμάζεται επίσης και σε μία άλλη κατηγορία πραγματικών δεδομένων. Για αυτό το λόγο, χρησιμοποιήθηκαν φορτία ερωτημάτων που προέρχονται από το APB-1 benchmark [apb]. Το APB-1 δημιουργεί μία δομή βάσης δεδομένων με πολλαπλές διαστάσεις και παράγει ένα σύνολο από επιχειρησιακές λειτουργίες που αντιπροσωπεύουν τη λειτουργικότητα των OLAP εφαρμογών. Τα παραγόμενα δεδομένα περιγράφονται από τέσσερις διαστάσεις. Η διάσταση customer (C) έχει 100 φορές τα μέλη της διάστασης channel και αποτελείται από 2 επίπεδα. Η διάσταση channel (Ch) έχει ένα επίπεδο και 10 μέλη. Η διάσταση product (P) είναι μία ιεραρχία με 6 επίπεδα και 10.000 μέλη. Τέλος η διάσταση time (T) περιγράφεται από μία ιεραρχία 3 επιπέδων και αποτελείται από τις τιμές για δύο χρόνια. Το σύνολο των δεδομένων είναι αραιό (0.1 πυκνότητα) και αποτελείται από 1.3M πλειάδες.



Σχήμα 4.20: *Το precision του LinkedPeers για το APB φορτίο ερωτημάτων*

Το Σχήμα 4.20 περιέχει αποτελέσματα για το ποσοστό των exact match ερωτημάτων που επιλύονται στα primary και secondary δακτυλίδια σε σύγκριση με τα exact match ερωτήματα ενός φορτίου 25K ερωτημάτων και για διαφορετικούς συνδυασμούς των διαστάσεων. Για όλους τους συνδυασμούς, το precision των non-flooded ερωτημάτων ξεπερνάει το 98%. Η επιλογή της διάστασης που θα χρησιμοποιηθεί ως primary διάσταση επηρρεάζει τον αριθμό των exact match ερωτημάτων στο primary δακτυλίδι.

Το Σχήμα 4.21 παρουσιάζει το μέσο αριθμό των μηνυμάτων για τα exact match ερωτήματα που επιλύθηκαν από ένα secondary δακτυλίδι και των indexed ερωτημάτων. Η επίλυση των exact match ερωτημάτων στο primary ring απαιτεί μόνο την εκτέλεση ενός DHT lookup. Ο μέσος αριθμός των μηνυμάτων είναι μικρός τόσο για τα exact όσο και για τα indexed ερωτήματα, εκτός από την περίπτωση όπου η διάσταση customer έχει επιλεχθεί ως primary διάσταση. Στις υπόλοιπες περιπτώσεις, η επίλυση των ερωτημάτων απαιτεί πολύ μικρό κόστος, δηλαδή ο αριθμός των επιπρόσθετων κόμβων που πρέπει να επισκεφτεί ένα ερώτημα είναι μικρός, αν και η πλειοψηφία των exact match ερωτημάτων επιλύεται από μία secondary διάσταση, όπως φαίνεται στο Σχήμα 4.20. Η αύξηση των μηνυμάτων για το CPChT συνδυασμό οφείλεται στο μεγάλο αριθμό των διακριτών τιμών που χρησιμοποιούνται ως pivot keys και επομένως κάθε κόμβος είναι υπεύθυνος για μικρότερα μέρη των συνολικών δεδομένων στο local database του. Για όλους τους συνδυασμούς, η επιβάρυνση από τις επιπρόσθετες δομές δεικτοδότησης που δημιουργούνται στο *LinkedPeers*, όπως οι δενδρικές δομές, τα root indices, τα links και η στατιστική πληροφορία ανέρχεται μέχρι το 1% επί του συνολικού όγκου. Έτσι το συμπέρασμα που προκύπτει είναι ότι το *LinkedPeers* μπορεί να θεωρηθεί ως μία ελαφριά λύση για τη δεικτοδότηση πολυδιάστατων, ιεραρχικών δεδομένων.



Σχήμα 4.21: *Μέσος αριθμός μηνυμάτων για exact match και indexed ερωτήματα*

4.7 Το Παράδειγμα των Διασυνδεδεμένων Δεδομένων

Η έλευση του Σημασιολογικού Ιστού (Semantic Web) [Sem] έχει επιφέρει σημαντικές εξελίξεις στην έννοια του "Web of Data", δηλαδή στην προσπάθεια για ενοποίηση και διασύνδεση διαφορετικών δεδομένων και των περιγραφών τους με στόχο να διευκολυνθεί η δημιουργία αποθηκών δεδομένων στο Διαδίκτυο, λεξικών και κανόνων για το χειρισμό τους. Η διαχείριση δεδομένων σε τόσο μεγάλη κλίμακα όσο υποδηλώνεται από τις εφαρμογές του Διαδικτύου θέτει κάποιες ενδιαφέρουσες προκλήσεις, που προκύπτουν κυρίως λόγω του μεγάλου όγκου των δεδομένων και της έλλειψης αυστηρώς ορισμένων σχημάτων που ακολουθούνται από όλες τις πηγές δεδομένων. Ο τελευταίος παράγοντας αποτελεί εμπόδιο για την επαναχρησιμοποίηση των δεδομένων, λόγω του γεγονότος ότι τα διάφορα εργαλεία μπορούν να χειριστούν και να επαναχρησιμοποιήσουν με μεγαλύτερη επιτυχία κανονικές και καλώς ορισμένες δομές. Για να ξεπεραστεί η έλλειψη μίας γενικής δομής, διαφορετικές διεπαφές αναπτύσσονται για κάθε ξεχωριστή πηγή δεδομένων. Παρόλα αυτά, η πολυπλοκότητα αυτής της λύσης εμποδίζει τις διαδικτυακές εφαρμογές από την απόκτηση πρόσβασης και το συνδυασμό δεδομένων που προέρχονται από διαφορετικές πηγές. Ωστόσο, η αποτελεσματική διασύνδεση και επεξεργασία δεδομένων από διαφορετικές πηγές είναι αυτή που καταλήγει συνήθως σε ενδιαφέροντα αποτελέσματα.

Οι παραδοχές που αναλύθηκαν παραπάνω ισχύουν και στο παράδειγμα των Διασυνδεδεμένων Δεδομένων (Linked Data) [Data], [BHBL09], όπου δεδομένα από διαφορετικές πηγές και τομείς διασυνδέονται παρέχοντας ενσωμάτωση ευρείας κλίμακας και επερώτηση δεδομένων του Διαδικτύου [Datb]. Ο όρος *Διασυνδεδεμένα Δεδομένα* αναφέρεται σε ένα σύνολο πρακτικών για τη δημοσίευση και διασύνδεση δομημένων δεδομένων στο Διαδίκτυο, ενώ έχει οριστεί ένα σύνολο αρχών που πρέπει να ακολουθούνται για να γίνονται διαθέσιμα τα δεδομένα στο πλαίσιο αυτό. Σύμφωνα με τις Αρχές των Διασυνδεδεμένων Δεδομένων ("Linked Data Principles") [BHBL09] ορίζονται οι ακόλουθοι βασικοί κανόνες για τη δημοσίευση δεδομένων στο Διαδίκτυο, ώστε να γίνουν μέρος ενός ενιαίου χώρου δεδομένων:

- Χρήση URIs ως ονόματα των πραγμάτων
- Χρήση HTTP URIs, ώστε αυτά τα ονόματα να μπορούν να αναζητηθούν
- Όταν κάποιος αναζητά ένα URI, συνιστάται η παροχή χρήσιμης πληροφορίας βάσει προτύπων (RDF, SPARQL).
- Να περιλαμβάνονται links σε άλλα URIs, ώστε να είναι εφικτή η ανακάλυψη περισσότερων πραγμάτων

Η διασύνδεση δεδομένων από διαφορετικές διαδικτυακές πηγές προϋποθέτει την ύπαρξη μηχανισμών που ακολουθούν τις βασικές αρχές των Διασυνδεδεμένων Δεδομένων, αλλά επίσης

τονίζουν την ύπαρξη και τη σημασία των συνδέσεων μεταξύ των αντικειμένων που περιγράφονται από αυτό το είδος δεδομένων. Το RDF μοντέλο έχει χρησιμοποιηθεί ευρέως για την αναπαράσταση και την ανταλλαγή τέτοιου είδους δεδομένων, καθώς παρέχει έναν ευέλικτο τρόπο για την περιγραφή πραγμάτων (όπως άνθρωποι, τοποθεσίες ή πιο αφηρημένες ιδέες) και πως αυτά σχετίζονται μεταξύ τους. Με τη χρήση του μοντέλου αυτού, τα δεδομένα αναπαριστώνται από τριπλέτες της μορφής: ⟨*subject, property, object*⟩. Κάθε τριπλέτα δηλώνει τη σχέση μεταξύ του subject και του object. Επίσης, η παρούσα τάση για τη δημοσίευση τέτοιου είδους δεδομένων είναι η δημιουργία SPARQL σημείων [SPA].

Η αξιολόγηση των ερωτημάτων σε τέτοιου είδους περιβάλλοντα γίνεται κυρίως με δύο τρόπους: με κεντρικοποιημένα συστήματα και με federation από πόρους. Η τυπική προσέγγιση στα υπάρχοντα συστήματα είναι η συγκέντρωση μεγάλων όγκων δεδομένων, η προεπεξεργασία τους και η φόρτωση των δεδομένων αυτών σε ένα κεντρικοποιημένο σημείο αποθήκευσης έτσι ώστε να επιτραπεί η επερώτηση των συγχωνευμένων δεδομένων τοπικά. Για παράδειγμα, το πρόγραμμα DBpedia [BLK+09] είναι μία προσπάθεια προς αυτήν την κατεύθυνση για να εξάγει δομημένη πληροφορία από το Wikipedia, να τη διασυνδέει με άλλους υπάρχοντες πόρους και να την κάνει διαθέσιμη στο Διαδίκτυο. Αν και οι κεντρικοποιημένες λύσεις πλεονεκτούν κατά την επεξεργασία των ερωτημάτων παρέχοντας πρόσβαση στο σύνολο της συλλογής δεδομένων, είναι ευάλωτες στην αύξηση του μεγέθους των δεδομένων ([HMZ10], [HHK+10]). Επιπλέον, η φάση της προ-επεξεργασίας μπορεί να είναι χρονοβόρα και έτσι να αποκρύβεται η χρησιμότητα των Διασυνδεδεμένων Δεδομένων. Ένα άλλο πρόβλημα είναι ο συγχρονισμός των τοπικών αντιγράφων σε αυτές τις προσεγγίσεις, ειδικά όταν οι πηγές των δεδομένων ή RDF υποστάσεις αλλάζουν συχνά.

Μία εναλλακτική προσέγγιση είναι η ενοποίηση των υπαρχόντων πόρων και η υλοποίηση μηχανισμών για κατανεμημένη επεξεργασία ερωτημάτων σε RDF πόρους [GS11]. Κατά την κατανεμημένη επεξεργασία, ένα ερώτημα διασπάται σε επιμέρους ερωτήματα, οι πηγές των δεδομένων ερωτώνται για να προσδιοριστούν αυτές με σχετική πληροφορία, το ερώτημα αξιολογείται στις σχετικές πηγές κατευθείαν και τελικά τα αποτελέσματα συγχωνεύονται στον κόμβο που εκτελεί την ενοποίηση. Το DARQ [QL08] παρέχει μια μηχανή για SPARQL ερωτήματα μεταξύ διαφόρων SPARQL σημείων. Χρησιμοποιεί περιγραφές των υπηρεσιών για τα δεδομένα που προσφέρουν έτσι ώστε να αποκτήσει πληροφορία για την επιλογή των πηγών κατά το σχεδιασμό της εκτέλεσης των ερωτημάτων. Στο [HHK+10], οι συγγραφείς προσπαθούν να δημιουργήσουν μία λύση που να συνδυάζει χαρακτηριστικά από εντελώς κεντρικοποιημένες και κατανεμημένες προσεγγίσεις. Για αυτό το σκοπό, δημιουργούν μία δομή δεικτοδότησης που στηρίζεται στο Q-tree για την αποθήκευση των περιγραφών του περιεχομένου των πηγών και χρησιμοποιούν αυτή τη δεικτοδότηση για να προσδιορίσουν σχετικές πηγές με υψηλή πιθανότητα. Το FedX [SHH+11] παρέχει ένα πλαίσιο για την εκτέλεση SPARQL ερωτημάτων για διαφορετικά σημεία. Προσπαθώντας να ελαττώσει την προώθηση σε πηγές χωρίς σχετικό περιεχόμενο,

χρησιμοποιεί SPARQL ASK ερωτήματα που επιστρέφουν θετική απάντηση εάν υπάρχει τουλάχιστον ένα σχετικό αποτέλεσμα. Επίσης, δίνεται προσοχή για τον προσδιορισμό της σειράς με την οποία εκτελούνται τα join ώστε να περιοριστούν τα ενδιάμεσα αποτελέσματα.

Η ενοποίηση διαφορετικών σημείων πρόσβασης σε πόρους προσθέτει κάποια επιβάρυνση, δεδομένου ότι οι μηχανισμοί επερώτησης δεν έχουν άμεση πρόσβαση στο σύνολο της συλλογής των δεδομένων και στην πληροφορία που χρειάζεται για τη βελτιστοποίηση του σχεδιασμού εκτέλεσης του ερωτήματος. Η έλλειψη ενός κοινού σχήματος μεταξύ των διαφορετικών σημείων αυξάνει επίσης την πολυπλοκότητα, ειδικά όταν αυξάνεται ο αριθμός των πηγών των δεδομένων ([HMZ10], [HHK$^+$10]). Το σύστημα δεικτοδότησης που προτείνεται μπορεί να αποτελέσει ένα επίπεδο υψηλής επίδοσης για την υλοποίηση των διαφόρων τεχνικών για το σχεδιασμό της εκτέλεσης των ερωτημάτων, οι οποίες είναι παρεμφερείς με αυτές που προτείνονται στις εργασίες [QL08], [SHH$^+$11]. Συνεπώς μπορεί να επιτευχθεί επιτάχυνση της διαδικασίας που απαιτείται πριν να ερωτηθούν οι πραγματικές πηγές των δεδομένων και να αντιμετωπιστούν προβλήματα που σχετίζονται με την επίδοση και την ενοποίηση των Διασυνδεδεμένων Δεδομένων.

4.8 PI4LD: Ένα P2P Σύστημα για τη Δεικτοδότηση Διασυνδεδεμένων Δεδομένων (P2P Indexing Scheme 'FOR' Linked Data)

Στη συνέχεια της διατριβής παρουσιάζεται το PI4LD (*Peer-to-peer Indexing FOR Linked Data*), που έχει σχεδιαστεί για την αποδοτική κατανομή δεδομένων σύμφωνα με τις αρχές για τα Διασυνδεδεμένα Δεδομένα, δηλαδή δεδομένων που περιγράφονται αυτοτελώς και περιέχουν σημασιολογική πληροφορία. Εφόσον το πρότυπο RDF παρέχει ένα γενικό, αφηρημένο μοντέλο για την περιγραφή των πόρων, οι ταξινομίες, τα λεξιλόγια και οι οντολογιές μπορούν να χρησιμοποιηθούν για να εκφράσουν πληροφορία που σχετίζεται περισσότερο με συγκεκριμένους τομείς. Εκτός από το RDF μοντέλο, το σύστημα PI4LD λαμβάνει υπόψη του και πιο εξειδικευμένα μοντέλα δεδομένων, όπως το RDF Vocabulary Definition Language (RDFS) και το Web Ontology Language (OWL), επιτρέποντας την αναπαράσταση δομημένων ή ημι-δομημένων δεδομένων.

Στοχεύοντας στην εξάλειψη των περιορισμών που συναντώνται στις κεντρικοποιημένες και ενοποιημένες προσεγγίσεις, προτείνεται μία μη κεντρικοποιημένη λύση για τη δεικτοδότηση διασυνδεδεμένων δεδομένων ώστε να επιτευχθεί η αποδοτική επεξεργασία των ερωτημάτων. Το σύστημα που προέκυψε μπορεί να χρησιμοποιηθεί είτε εξ'ολοκλήρου για την αποθήκευση και επερώτηση μεγάλου όγκου RDF συλλογών είτε ως ένα ενδιάμεσο επίπεδο για τη δεικτοδότηση των διασυνδεδεμένων δεδομένων πάνω από υπάρχουσες πηγές. Για την επίτευξη αυτού του στόχου, χρησιμοποιείται μια DHT επικάλυψη για τη διατήρηση συνόψεων των οντοτήτων και των μεταξύ τους σχέσεων σε μία ιεραρχική μορφή, ενώ τα αποθηκευμένα δεδομένα κατανέμονται ανάμεσα στους διαθέσιμους κόμβους. Η συγκεκριμένη επικάλυψη αποτελείται από δύο

'εικονικά' δακτυλίδια, ένα για το subject και ένα για το object, ενώ το property εμπεριέχεται μέσα στη δομή δεικτοδότησης. Το συνολικό αποτέλεσμα είναι ότι επιτυγχάνεται επικοινωνία με όσον το δυνατόν μικρότερο αριθμό κόμβων και μειώνονται οι πολύπλοκες λειτουργίες.

Συνοπτικά, το PI4LD σύστημα καταφέρνει να επιτύχει τα ακόλουθα:

- Αποτελεί μία ολοκληρωμένη λύση για την κατανομή, την αποθήκευση και τη δεικτοδότηση διασυνδεδεμένων δεδομένων μεταξύ των κόμβων. Το σύστημα είναι επίσης ικανό να εκτελεί online ενημερώσεις και να διατηρεί τα δεδομένα σε ένα κατανεμημένο σύστημα.

- Το προτεινόμενο σύστημα προσφέρει μία λύση για τον τελικό χρήστη, που μπορεί να επερωτήσει όλα τα δεδομένα με ενιαίο τρόπο. Τα ερωτήματα σε διαφορετικά επίπεδα των εννοιών μπορούν να εκτελεστούν αποδοτικά λόγω της ιεραρχικής οργάνωσης τους. Η προσαρμογή της δεικτοδότησης ανάμεσα σε διαφορετικά επίπεδα των οντοτήτων και των κλάσεων μπορεί να γίνεται ανάλογα με τα ερωτήματα των χρηστών.

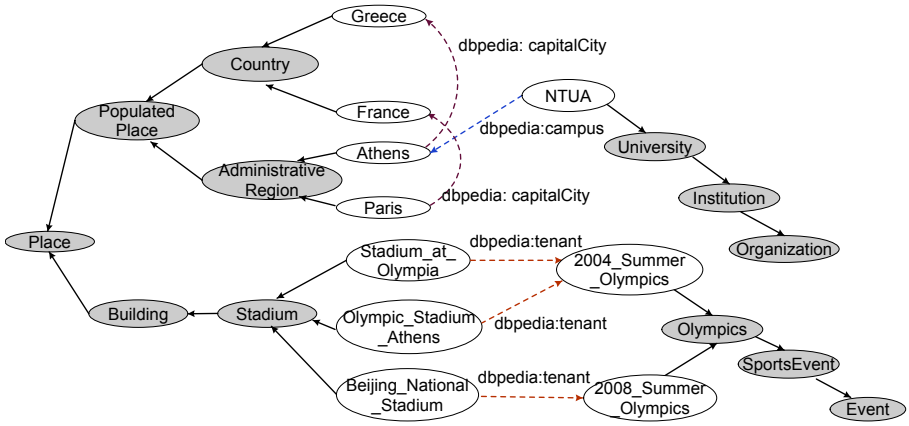
- Μία εκτενής αξιολόγηση της απόδοσης του συστήματος δείχνει ότι το σύστημα είναι ικανό για τη φιλοξενία διαφορετικών τύπων δεδομένων από διαφορετικές πηγές στο Διαδίκτυο και μπορεί να αποτελέσει μία βιώσιμη λύση.

4.9 Μοντέλο Δεδομένων στο PI4LD

Από την έλευση των linked data, οι περισσότεροι από τους πάροχους δεδομένων (data providers) υιοθετούν το RDF μοντέλο για τη δημοσίευση των δεδομένων τους. Ένα σημαντικό χαρακτηριστικό που διακρίνει αυτές τις προσπάθειες για τη δημοσίευση δομημένης πληροφορίας στο Διαδίκτυο είναι η υιοθέτηση μοντέλων που βασίζονται στο RDFS και στο Web Ontology Language OWL για την αναπαράσταση της σημασιολογικής πληροφορίας, δηλαδή για να εκφράσουν οντότητες, facts, σχέσεις μεταξύ των facts και ιδιότητες των σχέσεων. Για παράδειγμα, η κοινότητα του DBpedia χρησιμοποιεί μία οντολογία για διαφορετικούς τομείς που δεν έχει δημιουργηθεί με αυτόματο τρόπο και έχει εξαχθεί από τα infoboxes που υπάρχουν στο Wikipedia. Όλα τα αντικείμενα (όπως χώρες, πόλεις, οργανισμοί, άνθρωποι) μπορούν να θεωρηθούν ως οντότητες, ενώ κάθε οντότητα είναι υπόσταση (instance) τουλάχιστον μίας κλάσης που ορίζεται στην οντολογία [SKW07]. Οι κλάσεις μπορούν επίσης να θεωρηθούν ως οντότητες. Η ιδιότητα `rdf:type` χρησιμοποιείται για να δηλώσει ότι ένας πόρος (όλα τα πράγματα που περιγράφονται από RDF ονομάζονται *πόροι-resources*) είναι instance (ή individual) μίας κλάσης.

Η χρήση των οντολογιών υπονοεί μία ιεραρχική οργάνωση των δεδομένων δεδομένου ότι μία οντολογία αναπαριστά τη γνώση ως ένα σύνολο ιδεών σε έναν τομέα και αυτές οι έννοιες μπορούν να δομηθούν με ιεραρχικό τρόπο. Η διάταξη των εννοιών μίας οντολογίας σε μια ταξινομία (ή κατηγορική ιεραρχία - category hierarchy) συμβάλλει στη χρήση και το reasoning

[SKW07], [JHS+10]. Στις περιπτώσεις που μελετώνται, οι κλάσεις της οντολογίας διατάσσονται σε ιεραρχίες και η σχέση `rdfs:subClassOf` χρησιμοποιείται για να δηλώσει ότι μια κλάση είναι υποκλάση μιας άλλης. Στο προτεινόμενο σύστημα, η ύπαρξη της ιεραρχικής δομής αξιοποιείται, η οποία εξάγεται από τέτοιου είδους δεδομένα και χρησιμοποιείται για την κατανομή τους στους κόμβους του συστήματος με τέτοιον τρόπο ώστε να διατηρούνται οι ιδιότητες της ιεραρχίας και να επιτευχθεί πιο αποτελεσματική επεξεργασία των ερωτημάτων. Συνεπώς, το μοντέλο δεδομένων που προτείνεται σε αυτό το σύστημα οργανώνει τις οντότητες σε δέντρα σύμφωνα με τις σχέσεις `rdfs:subClassOf` μεταξύ των κλάσεων και των οντοτήτων που είναι υποστάσεις τους. Στη συνέχεια όλες οι αναφορές στις υποστάσεις γίνονται ως οντότητες και θεωρείται ότι μία οντότητα μπορεί να εμφανίζεται είτε ως subject είτε ως object σε ένα triple.



Σχήμα 4.22: *Αναπαράσταση μερικών DBpedia instances*

Το Σχήμα 4.22 δείχνει μερικά triples από διαφορετικά instances και τις σχέσεις που τα συνδέουν, όπως αυτά προέκυψαν από το DBpedia σύνολο δεδομένων. Τα triples αυτά οργανώνονται σε διαφορετικές δενδρικές δομές σύμφωνα με τις κοινές τιμές τους. Στο λογικό επίπεδο, οι γκρι κόμβοι που αναπαριστούν το επίπεδο του σχήματος (*schema* layer) διαχωρίζονται από το επίπεδο των πραγματικών υποστάσεων που απεικονίζονται με λευκούς κόμβους. Τα συμπαγή βέλη προέκυψαν από την ιδιότητα `rdf:type` και τις σχέσεις που ορίζονται στην οντολογία (οι ετικέτες παραλείπονται για αυτές τις άκρες για λόγους αναγνωσιμότητας), ενώ οι έγχρωμες, διακεκομμένες γραμμές οπτικοποιούν τις σχέσεις μεταξύ των οντοτήτων, όπου το είδος της κάθε σχέσης φαίνεται στην ετικέτα της κάθε γραμμής. Όλα τα βέλη κατευθύνονται από το subject του triple προς το object. Για παράδειγμα, οι τύποι της οντότητας σχετικά με το `Athens` και η σχέση του με την οντότητα `NTUA` περιγράφονται από τα ακόλουθα triples [1]:

[1] Τα URIs παραλείπονται για λόγους αναγνωσιμότητας

```
Athens rdf:type Place
Athens rdf:type Populated Place
Athens rdf:type Administrative Region
NTUA dbpedia:campus Athens
```

4.9.1 Συμβολισμοί και Ορισμοί

Στο προτεινόμενο σύστημα, τα δεδομένα εισάγονται με τη μορφή των triples όπως περιγράφηκε ήδη στην προηγούμενη Ενότητα. Μία απαίτηση που τίθεται από την αρχιτεκτονική του συστήματος είναι ότι μία καινούργια οντότητα εισάγεται στο PI4LD μαζί με τα triples που ορίζουν τους τύπους της έτσι ώστε το "*πλήρες μονοπάτι*" ("*full path*") της ιεραρχίας των κλάσεων που περιγράφουν τη συγκεκριμένη οντότητα να προσδιοριστεί. Για παράδειγμα, η εισαγωγή του triple ⟨*NTUA, campus, Athens*⟩ απαιτεί όλα τα triples από τη γενικότερη έννοια μέχρι την πιο λεπτομερή να είναι διαθέσιμα κατά την εισαγωγή τους και για τις δύο οντότητες. Η προσέγγιση αυτή ακολουθείται έτσι ώστε να δημιουργηθεί ένα επεκτάσιμο και ευέλικτο σύστημα, χωρίς να απαιτείται η γνωστοποίηση των υπαρχουσών οντολογιών σε όλους τους κόμβους του συστήματος.

Οι οντότητες οργανώνονται σε ιεραρχίες με L_i επίπεδα σύνοψης ℓ_{ij}, όπου το i αντιστοιχεί στην *i-th* οντολογία και το j ($j \in [0, L_i - 1]$) αναπαριστά το *j-το* επίπεδο της *i-της* οντολογίας. Το επίπεδο ℓ_{i0} ονομάζεται *root level* για την *i-τη* οντολογία και το hashed value του ονομάζεται *root key* και αντιστοιχεί στη βασικότερη ιδέα ενός τομέα. Αν και κάθε individual στο OWL είναι μέρος της κλάσης `owl:Thing` και συνεπώς όλες οι οντότητες πρέπει να έχουν το `Thing` ως root value, το επίπεδο αυτό παραλείπεται στις ιεραρχίες και οι οντότητες του επόμενου επιπέδου θεωρούνται ως root values.

Επίσης ορίζεται ότι το ℓ_{ik} βρίσκεται *υψηλότερα* (*χαμηλότερα*) από το ℓ_{il} και συμβολίζεται ως $\ell_{ik} < \ell_{il}$ ($\ell_{ik} > \ell_{il}$) *iff* $k < l$ ($k > l$), δηλαδή εάν το ℓ_{ik} αντιστοιχεί σε ένα λιγότερο (περισσότερο) λεπτομερές επίπεδο από το ℓ_{il} (π.χ., *PopulatedPlace* < *Country*).

Οι τιμές της ιεραρχίας οργανώνονται σε δενδρικές δομές, διαφορετικές για κάθε root key. Χωρίς απώλεια της γενικότητας, θεωρείται ότι η κάθε τιμή του επιπέδου ℓ_{ij} έχει το πολύ ένα πρόγονο στο αμέσως υψηλότερο επίπεδο $\ell_{i(j-1)}$. Αξίζει επίσης να σημειωθεί ότι οι κλάσεις μίας οντολογίας σχηματίζουν μία ιεραρχία, όπου ο αριθμός των επιπέδων σε κάθε κλάδο δεν είναι σταθερός. Στο προτεινόμενο σύστημα, θεωρείται ως προαπαιτούμενο ότι όλοι οι κλάδοι του ίδιου root value έχουν τον ίδιο αριθμό επιπέδων, έτσι ώστε οι λειτουργίες re-indexing να μπορούν να εκτελεστούν όπως περιγράφηκε στην Ενότητα 3.5. Εφόσον σε μια πραγματική οντολογία, μπορεί να υπάρχει ένα διαφορετικός αριθμός επιπέδων κάτω από έναν κόμβο, τα επίπεδα που απουσιάζουν πάνω από το leaf value γεμίζουν με ψευδοτιμές.

4.9.2 Οργάνωση των Δεδομένων

Η προτεινόμενη αρχιτεκτονική ενός πλήρως κατανεμημένου συστήματος για τη διαχείριση διασυνδεδεμένων δεδομένων βασίζεται στις θεμελιώδεις αρχές των DHTs. Στο PI4LD αναπτύσσονται επεκτάσιμοι μηχανισμοί για την κατανομή, τη δεικτοδότηση και την επερώτηση διασυνδεδεμένων δεδομένων με ένα πλήρως κατανεμημένο τρόπο.

Η κατανομή των δεδομένων στους κόμβους της επικάλυψης γίνεται με τέτοιο τρόπο ώστε να διατηρηθεί η πληροφορία που εκφράζεται στην οντολογία σχετικά με κάθε οντότητα, ενώ οι διάφορες οντότητες διασυνδέονται σύμφωνα με τις ιδιότητες τους. Ο στόχος προς επίτευξη στο σχεδιασμό της αρχιτεκτονικής είναι η αποθήκευση κάθε οντότητας με ένα πλήρως περιγραφικό τρόπο. Συμπερασματικά, ο κόμβος που είναι υπεύθυνος για μία οντότητα αποθηκεύει και όλες τις κλάσεις (ή τις έννοιες) που σχετίζονται με τη συγκεκριμένη οντότητα. Σύμφωνα με το DHT πρωτόκολλο, ένα *key* πρέπει να αντιστοιχηθεί σε κάθε οντότητα που εισάγεται στο σύστημα, έτσι ώστε η οντότητα να καταλήξει στον κόμβο που είναι υπεύθυνος για αυτό. Στο PI4LD επιλέχθηκε η δημιουργία αυτού του κλειδιού να γίνεται βάσει της τιμής ενός επιπέδου της ιεραρχίας. Η τιμή αυτή αναφέρεται ως *pivot value*, η hashed τιμή της ως *pivot key* και το επίπεδο της ως *pivot level*. Το υψηλότερο και το χαμηλότερο pivot level της συνολικής ιεραρχίας για ένα root value ονομάζεται *MinPivotLevel* και *MaxPivotLevel* αντιστοίχως. Δεδομένου ότι πολλαπλές οντότητες μπορούν να περιγραφούν από την ίδια διάταξη κλάσεων (ή εννοιών) που ορίζονται σε μία οντολογία, κάθε κόμβος οργανώνει όλες τις σχετικές οντότητες σε δενδρικές δομές που χαρακτηρίζονται από κοινές κλάσεις (ή έννοιες ή οντότητες) στο root level και το ίδιο pivot value.

Η επίλυση των ερωτημάτων που αναζητούν διασυνδεδεμένες οντότητες απαιτεί την αποθήκευση της πληροφορίας που σχετίζεται με τη μεταξύ τους διασύνδεση στην επικάλυψη. Για παράδειγμα, το ερώτημα για την ανάκτηση των ονομάτων των Stadiums που φιλοξένησαν Olympic Games είναι:

```
SELECT ?Stadium ?Event
WHERE { ?Stadium dbpedia:tenant ?Event.
?Stadium rdf:type  Stadium. ?Event rdf:type  Olympics.}
```

Για να είναι δυνατή η επίλυση ενός τέτοιου ερωτήματος, δεν είναι αρκετό να βρεθούν οι οντότητες που είναι μέλη των κλάσεων `Stadium` και `Olympics`, αλλά να βρεθούν και ποιες από αυτές τις οντότητες διασυνδέονται με την ιδιότητα `dbpedia:tenant`. Για αυτό το λόγο, αποφασίστηκε να αποθηκεύεται μία πλειάδα που περιέχει τις τιμές που σχετίζονται τόσο με τη σημασιολογική πληροφορία όσο και με το property στο local database του κόμβου που είναι υπεύθυνος για την οντότητα που εμφανίζεται ως subject. Όσον αφορά το παράδειγμα που περιγράφηκε, μία τέτοια

πλειάδα για δύο από τις οντότητες που απεικονίζονται στο Σχήμα 4.22 και απαντούν το παραπάνω ερώτημα θα είναι η εξής:

```
⟨Place, Building, Stadium, Stadium_of_Olympia, Event, SportsEvent, Olympics,
2004_Summer_Olympics,dbpedia:tenant⟩
```

όπου η τελευταία τιμή είναι η ιδιότητα που διασυνδέει τις δύο οντότητες. Η γενικότερη ιδέα είναι η διατήρηση ενός *primary virtual ring* που αποθηκεύει όλη την πληροφορία σχετικά με τα triples και ενός *secondary virtual ring* που δεικτοδοτεί τις οντότητες που εμφανίζονται ως objects. Τα pivot keys που αντιστοιχούν στο primary ring ονομάζονται *primary keys* και οι πλειάδες αποθηκεύονται στο local database του primary ring. Λόγω του γεγονότος ότι οι περισσότερες τιμές που εμφανίζονται μέχρι το τελευταίο επίπεδο είναι κοινές, η ιδιότητα αυτή μπορεί να αξιοποιηθεί για το σχεδιασμό ενός πιο αποτελεσματικού σχήματος σε σύγκριση με την απλή λύση της χρήσης ενός απλού πίνακα με πολλαπλές στήλες. Ένας άλλος παράγοντας που πρέπει να ληφθεί υπόψη είναι ότι η σημασιολογική πληροφορία που περιγράφει την οντότητα που αντιστοιχεί σε ένα subject μπορεί να βρεθεί και στις δενδρικές δομές που διατηρούνται από τον κόμβο. Παρόλα αυτά, η οργάνωση και βελτιστοποίηση του local database δεν αποτελεί βασικό μέρος της περιγραφόμενης αρχιτεκτονικής και δεν αναλύεται περαιτέρω.

Μία άλλη παρατήρηση είναι ότι οι εικονικές επικαλύψεις δεικτοδοτούν μόνο τα subjects και τα objects. Αυτό έχει ως αποτέλεσμα να μην είναι δυνατή η επίλυση των ερωτημάτων που περιέχουν μόνο ιδιότητες χωρίς να προωθηθούν και να αξιολογηθούν από όλους τους κόμβους του συστήματος. Η δημιουργία των δεικτών για τα properties παραλείπεται λόγω της θεώρησης ότι τα properties δεν ακολουθούν ιεραρχικές σχέσεις. Εάν αποφασιστεί ότι χρειάζεται να δεικτοδοτηθούν και αυτές στο PI4LD, τότε οι τιμές τους μπορούν απλά να γίνουν hashed και να εισαχθούν στη DHT επικάλυψη σύμφωνα με τις διαδικασίες που ακολουθούνται για τις εισαγωγές δεδομένων σε μία secondary διάσταση. Επίσης έχει παρατηρηθεί συχνά ότι στα ερωτήματα που αφορούν διασυνδεδεμένα δεδομένα είναι πιθανό να προσδιορίζεται τουλάχιστον ο τύπος του subject ή του object [Hal09]. Εάν αυτό ισχύει, η επίλυση του ερωτήματος μπορεί να ξεκινήσει από αυτήν την τιμή και να ακολουθήσει τα links που διατηρούνται στο σύστημα μέχρι να επιλυθεί. Ωστόσο, εάν πρόκειται το σύστημα να εξυπηρετεί κυρίως ερωτήματα που αφορούν μόνο properties, τότε προστίθεται ένα ακόμα secondary δακτυλίδι στο σύστημα όπως αυτό που έχει δημιουργηθεί για τα objects χωρίς να χρειάζονται περαιτέρω τροποποιήσεις.

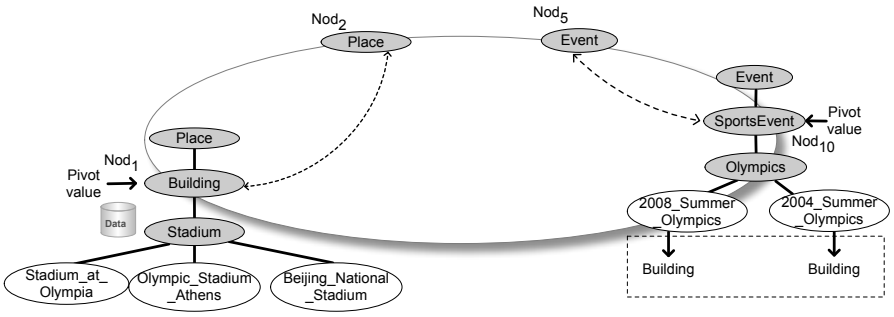
Κατά τη διάρκεια των εισαγωγών των δεδομένων, η πληροφορία για το pivot level είναι ζωτικής σημασίας (μόνο το pivot level που χρησιμοποιείται κατά τις αρχικές εισαγωγές μπορεί να επιλεχθεί αυθαίρετα ανάλογα με τις ανάγκες της εφαρμογής). Ο σχεδιασμός του συστήματος απαιτεί ότι αν μία τιμή έχει επιλεχθεί ως pivot key κατά την εισαγωγή μίας οντότητας, τότε το ίδιο pivot key πρέπει επίσης να επιλεχθεί ως pivot key για κάθε άλλη οντότητα που σχετίζεται

με αυτό μέσω της ιδιότητας `rdf:type`. Για να τηρηθεί αυτός ο περιορισμός, ένας κόμβος πρέπει να γνωρίζει τα υπάρχοντα pivot keys όταν εισάγει μία καινούργια πλειάδα. Για αυτόν το λόγο υλοποιείται ένας πλήρως κατανεμημένος κατάλογος που αποθηκεύει πληροφορία σχετική με τα root keys και τα αντίστοιχα pivot keys τους στο PI4LD. Κάθε root key αποθηκεύεται στον κόμβο που είναι "πλησιέστερος" στο ID του. Κάθε φορά που ένα καινούργιο pivot key που αντιστοιχεί σε αυτό το root key εισάγεται στο σύστημα, ο κόμβος του root key ενημερώνεται για αυτό και το προσθέτει στη λίστα των γνωστών pivot keys. Ο κόμβος του root key γνωρίζει επίσης και το MaxPivotLevel που χρησιμοποιήθηκε κατά την εισαγωγή των τιμών στη συγκεκριμένη διάσταση.

Αρχικά, ο κόμβος που εισάγει μία πλειάδα επικοινωνεί με τον κόμβο που είναι υπεύθυνος για το root key της υποπλειάδας στην primary διάσταση. Το root key της primary διάστασης ενημερώνεται για την καινούργια οντότητα και υπαγορεύει το κατάλληλο pivot level (εάν το ίδιο pivot key υπάρχει ήδη, τότε χρησιμοποιείται το pivot level του, διαφορετικά το MaxPivotLevel). Στη συνέχεια, η DHT λειτουργία για την εισαγωγή μίας οντότητας στην primary διάσταση ξεκινάει και η πλειάδα καταλήγει στον κόμβο που είναι υπεύθυνος για το pivot key που αποφασίστηκε. Ο κόμβος που είναι υπεύθυνος για το pivot key της primary διάστασης αποθηκεύει την οντότητα στην αντίστοιχη δενδρική δομή και ολόκληρη την πλειάδα στο local database του.

Το επόμενο βήμα είναι η αποθήκευση της οντότητας που εμφανίζεται ως object στο νέο triple. Ο κόμβος που είναι υπεύθυνος για το primary key επικοινωνεί με τον κόμβο που είναι υπεύθυνος για το root key αυτής της οντότητας και ενημερώνεται για το κατάλληλο pivot level. Αφού αποφασιστεί το pivot value για τη συγκεκριμένη οντότητα, αυτή αποθηκεύεται στη δενδρική δομή του κόμβου που είναι υπεύθυνος για το pivot key της. Το object συσχετίζεται με τη primary διάσταση μέσω του primary key. Κάθε οντότητα του secondary tree διατηρεί μία λίστα με τα primary keys που σχετίζεται. Η δομή που αποθηκεύει τις αντιστοιχίσεις μεταξύ των leaf values και των primary keys αναφέρεται ως *Linked Table*. Όλες οι οντότητες που εισάγονται με το ίδιο pivot key καταλήγουν στον ίδιο κόμβο που καθορίζεται από την τιμή του pivot key. Η μόνη διαφοροποίηση μεταξύ ενός subject και ενός object είναι ότι στην περίπτωση του subject μπορεί να βρεθεί όλο το tuple στο local database, ενώ στην αντίθετη περίπτωση υπάρχει μόνο ένα link προς το primary pivot key. Δεδομένης της σημασίας του property `owl:sameAs` για την επερώτηση διασυνδεδεμένων δεδομένων από διαφορετικές πηγές, τα links προς το primary ring που χαρακτηρίζονται από αυτό το property μπορούν να σημειωθούν για να διαχωριστούν από τις υπόλοιπες ιδιότητες. Κατά τη διάρκεια των ενημερώσεων εκτελείται η ίδια διαδικασία που περιγράφηκε για την εισαγωγή των νέων triples. Μία επιπρόσθετη απαίτηση που τίθεται είναι ότι πρέπει να ενημερωθούν όποιοι δείκτες είναι πιθανόν να υπάρχουν προς τις υποστάσεις των κλάσεων που κληρονομεί η καινούργια οντότητα και βρίσκονται πάνω από το pivot level, εάν το pivot key που χρησιμοποιήθηκε δεν υπάρχει ήδη στην επικάλυψη. Για αυτό το λόγο εκτελούνται

lookups και ενημερώνεται οποιαδήποτε σχετική πληροφορία (όπως αναφέρεται στην Ενότητα 3.5.1).



Σχήμα 4.23: *Κατανομή των διασυνδεδεμένων δεδομένων στο PI4LD*

Στο παράδειγμα του Σχήματος 4.23, το επίπεδο ℓ_1 επιλέχθηκε για την εισαγωγή όλων των triples που παρουσιάζονται στο Σχήμα 4.22 και σχετίζονται ευθέως ή μη με το property `dbpedia:tenant`. Τα root keys των primary και secondary διαστάσεων για αυτές τις πλειάδες είναι `Place` και `Event` αντιστοίχως. Η δενδρική δομή στον κόμβο Nod_1 αποθηκεύει όλες τις οντότητες που εμφανίζονται ως subject στα triples με την ιδιότητα `dbpedia:tenant`, ενώ όλα τα objects σχηματίζουν τη δενδρική δομή που φαίνεται για τον κόμβο Nod_{10}. Οι οντότητες του χαμηλότερου επιπέδου έχουν επίσης links προς το pivot key της primary διάστασης. Εάν μία καινούργια οντότητα τύπου `Populated Place` εισαχθεί στην επικάλυψη, τότε μία καινούργια δενδρική δομή θα δημιουργηθεί σε έναν άλλο κόμβο της επικάλυψη και το root key για την τιμή `Place` θα δείχνει επίσης προς αυτόν τον κόμβο. Στην περίπτωση που εισαχθεί ένα νέο triple που περιέχει την οντότητα `2012_Summer_Olympics` ως subject, τότε και αυτό θα καταλήξει στον κόμβο Nod_{10} και θα προστεθεί στο υπάρχον δέντρο (εάν δεν υπάρχει ήδη). Η μόνη διαφορά είναι ότι δε θα δημιουργηθεί ένα link προς το primary key, αλλά θα προστεθεί η αντίστοιχη πλειάδα στο local database.

4.10 Επερώτηση Διασυνδεδεμένων Δεδομένων

Το προτεινόμενο σύστημα επεκτείνει τη DHT lookup λειτουργία για να παρέχει ένα κατανεμημένο μηχανισμό ικανό να εντοπίζει οποιαδήποτε τιμή των αποθηκευμένων triples. Εφόσον το σύστημα προορίζεται για την αποθήκευση RDF triples, είναι αναμενόμενη η χρήση SPARQL για την επερώτηση των αποθηκευμένων δεδομένων. Ο βασικός τύπος των SPARQL ερωτημάτων αποτελείται από δύο μέρη: το `SELECT` όρο για την αναγνώριση των μεταβλητών που θα εμφανιστούν στα αποτελέσματα του ερωτήματος και το `WHERE` όρο που παρέχει το βασικό μοντέλο γράφου που θα αντιπαραβληθεί με το γράφο των δεδομένων. Η βασική μονάδα SPARQL ερωτήματος αναφέρεται στο μοντέλο <*subject, property, object*>, όπου μπορούν να χρησιμοποιηθούν

μεταβλητές αντί για συγκεκριμένες τιμές για κάθε ένα από τα συστατικά του μέρη. Εκτός από τα `SELECT` ερωτήματα, το SPARQL πρότυπο [SPA] ορίζει επίσης και άλλους τύπους ερωτημάτων, όπως `ASK`, `DESCRIBE` και `CONSTRUCT` ερωτήματα. Όμως δίνεται έμφαση μόνο στα `SELECT` ερωτήματα, δεδομένου ότι η αρχιτεκτονική που παρουσιάζεται επικεντρώνεται κυρίως σε παραμέτρους που αφορούν την κατανομή των δεδομένων μεταξύ πολλαπλών κόμβων και την αποτελεσματική δεικτοδότηση τους για την επίλυση των ερωτημάτων σε ένα πλήρως κατανεμημένο περιβάλλον.

Ο βασικότερος τύπος ερωτημάτων που είναι πιθανό να τεθεί στο σύστημα μπορεί να περιέχει είτε μία μόνο τιμή είτε ένα συνδυασμό τιμών προς αναζήτηση. Ο βασικός τύπος του ερωτήματος που υποστηρίζεται από το προτεινόμενο σύστημα είναι της ακόλουθης μορφής:

$$q = (?s, ?o, ?p, restrictions, measures)$$

Οι τιμές των μεταβλητών s, p και o μπορεί να έχουν είτε μία προκαθορισμένη τιμή ή να αποκτήσουν τιμή κατά την επίλυση του ερωτήματος. Το ερώτημα μπορεί να περιέχει και περιορισμούς για τις τιμές που ζητούνται και μετρήσεις (measures) που μπορούν να προσδιοριστούν για ανάκτηση. Τα measures αναφέρονται σε οποιαδήποτε άλλη σχετική πληροφορία και ιδιότητες των οντοτήτων που είναι τύπου attribute ή datatype. Η πληροφορία που αναφέρεται ως measures δεν έχει δεικτοδοτηθεί στο DHT και η αποθήκευση και η ανάκτηση της γίνεται μόνο από τα local databases. Για παράδειγμα, εάν μία οντότητα σχετίζεται με μία περίληψη, τότε η περίληψη αυτή αποθηκεύεται μόνο στο local database του κόμβου που αποθηκεύει τη συγκεκριμένη οντότητα. Το ακόλουθο ερώτημα σχετικά με τα campus που βρίσκονται στην τοποθεσία "Athens":

```
SELECT ?s
WHERE { ?s dbpedia:campus Athens.}
```

είναι ένα βασικό ερώτημα που εκφράζεται με την ακόλουθη εσωτερική μορφή του συστήματος:

```
(?s,"Athens","dbpedia:campus")
```

Σε γενικές γραμμές, η βασική lookup λειτουργία μίας DHT επικάλυψης επιτρέπει την ανάκτηση ενός αντικειμένου εάν το hashed value του χρησιμοποιήθηκε ως *key* κατά την εισαγωγή του. Εάν μία οντότητα που χρησιμοποιήθηκε ως pivot value ερωτάται, τότε ο κόμβος που είναι υπεύθυνος για αυτήν την τιμή μπορεί να βρεθεί με ένα απλό DHT lookup. Διαφορετικά, οι μηχανισμοί του DHT δεν είναι επαρκείς για τον εντοπισμό αυτής της οντότητας. Επιπλέον, το προτεινόμενο σύστημα επιτρέπει την προσαρμογή των pivot levels σύμφωνα με τα εισερχόμενα ερωτήματα. Λόγω αυτής της ιδιότητας, ένας κόμβος που θέτει ένα ερώτημα δε γνωρίζει εάν κάποια από τις ζητούμενες τιμές αντιστοιχεί σε ένα pivot value. Η στρατηγική που ακολουθείται στην προτεινόμενη προσέγγιση είναι ότι ο κόμβος αποστέλλει διαδοχικά lookups για κάθε μία από τις τιμές που περιλαμβάνει το ερώτημα ξεκινώντας από την τιμή του subject. Εάν μία lookup λειτουργία δεν επιστρέψει κάποιο αποτέλεσμα για την τιμή του subject, τότε το επόμενο lookup

αφορά την τιμή του object, εφόσον και οι δύο τιμές ορίζονται στο ερώτημα. Η λογική στη συγκεκριμένη αντιμετώπιση είναι ότι οι lookup λειτουργίες δεν έχουν μεγάλο κόστους δεδομένου του λογαριθμικού κόστους δρομολόγησης που παρουσιάζουν. Λόγω του μικρού τους κόστους, η εκτέλεση ενός επιπρόσθετου lookup είναι προτιμότερη από την προώθηση του ερωτήματος σε όλους τους κόμβους της επικάλυψης. Ωστόσο, τα DHT lookups δεν επαρκούν για την επίλυση των ερωτημάτων που δεν περιέχουν κανένα pivot value. Για αυτό το λόγο, αναπτύχθηκαν πιο προηγμένες λειτουργίες αναζήτησης που μπορούν να χρησιμοποιηθούν για τον εντοπισμό οποιασδήποτε αποθηκευμένης οντότητας. Οι βασικές τεχνικές για την αναζήτηση των οντοτήτων στο PI4LD είναι οι ακόλουθες:

Exact Match ερωτήματα: Τα ερωτήματα που αφορούν οποιοδήποτε pivot key που μπορεί να εμφανίζεται ως subject ή ως object ανήκουν στην κατηγορία των `exact match` ερωτημάτων. Εάν μία οντότητα που αναζητείται αντιστοιχεί σε ένα pivot key, τότε το DHT lookup για την τιμή της καταλήγει στον κόμβο που είναι υπεύθυνος για αυτή και αναζητείται η αντίστοιχη δενδρική δομή για την ανάκτηση των σχετικών triples. Οι δενδρικές δομές χρησιμοποιούνται μόνο για την επίλυση των ερωτημάτων που αφορούν μία οντότητα και πως αυτή σχετίζεται με τις υπόλοιπες οντότητες στην ιεραρχία. Ένα τέτοιο παράδειγμα είναι ένα ερώτημα για όλα τα αποθηκευμένα κτίρια (buildings) `(?s, Building,rdf:type)` και επιλύεται κατευθείαν από την πληροφορία που περιλαμβάνεται στην αντίστοιχη δενδρική δομή. Σε όλες τις υπόλοιπες περιπτώσεις, το ερώτημα αποτιμάται με τη χρήση της πληροφορίας που αποθηκεύεται στο local database. Όταν χρειάζεται να τεθεί το ερώτημα στο local database και το pivot key αντιστοιχεί στην οντότητα του subject, τότε το local database του κόμβου, που κατάληξε το ερώτημα, διαθέτει όλη τη σχετική πληροφορία. Διαφορετικά, το ερώτημα πρέπει να προωθηθεί σύμφωνα με τα links του Linked Table προς τα σχετικά κλειδιά, ώστε να ερωτηθούν τα local databases τους για την τελική επίλυση του ερωτήματος.

Flood ερωτήματα: Τα ερωτήματα που δεν περιέχουν κάποιο pivot key δεν είναι δυνατόν να επιλυθούν από τη λειτουργία lookup του DHT. Η μόνη εναλλακτική είναι να αποσταλεί το ερώτημα σε όλους τους κόμβους για να γίνει η επεξεργασία του από τον κάθε κόμβο μεμονωμένα και να επιστραφούν τα σχετικά αποτελέσματα στον κόμβο που έθεσε το ερώτημα. Στο PI4LD εφαρμόζεται μία απλούστερη flood διαδικασία σε σύγκριση με αυτήν που περιγράφηκε για το *LinkedPeers* και το ερώτημα προωθείται από τον ένα κόμβο στον πλησιέστερο γείτονα του μέχρι να φτάσει σε όλους τους κόμβους της επικάλυψης. Κάθε κόμβος επεξεργάζεται τοπικά ένα flood ερώτημα που λαμβάνει αναζητώντας τις δενδρικές δομές του και το local database του. Αναλυτικότερα, εάν το ερώτημα περιλαμβάνει μόνο μία οντότητα, τότε η αποθηκευμένη πληροφορία στις δενδρικές δομές είναι επαρκής για την απάντηση του και έτσι αποφεύγεται η αναζήτηση του local database. Σε περίπτωση που αναζητούνται περισσότερες από μία τιμές, τότε το ερώτημα απαντιέται απευθείας από το local database. Οι δενδρικές δομές αναζητούνται μόνο για την να

ανακαλυφθεί εάν κάποια από τις οντότητες του ερωτήματος βρίσκεται σε αυτές ώστε να ενημερωθούν οι αντίστοιχες εγγραφές που διατηρούν τη στατιστική πληροφορία για το συγκεκριμένο δέντρο.

Indexed ερωτήματα: Κατά τη διάρκεια της flood διαδικασίας, όλοι οι κόμβοι της επικάλυψης επερωτούνται και συνεπώς μπορεί να αξιοποιηθεί περαιτέρω η γνώση που ανακαλύπτεται. Για την εκμετάλλευση αυτής της γνώσης προτείνεται η δημιουργία των soft-state δεικτών. Όταν ένας κόμβος απαντά ένα flood ερώτημα, ελέγχει αν χρειάζεται μία λειτουργία re-indexing (όπως αναφέρεται στην Ενότητα 4.11). Εάν το σύστημα αποφασίσει ότι μια λειτουργία re-indexing δε συνιστάται, τότε ο κόμβος που ξεκίνησε το ερώτημα ξεκινάει τη διαδικασία για τη δημιουργία των soft-state δεικτών για όλες τις τιμές του ερωτήματος. Εφόσον ένα flood ερώτημα φτάνει σε όλους τους κόμβους, μπορούν να βρεθούν οι κόμβοι που είναι υπεύθυνοι για τις δενδρικές δομές που περιέχουν τις τιμές που υπάρχουν στο ερώτημα. Ο κόμβος initiator ξεκινάει τη διαδικασία της δημιουργίας των δεικτών παράγοντας τα keys των ζητούμενων τιμών και στη συνέχεια τα εισάγει στην DHT επικάλυψη μαζί με τους κόμβους που είναι υπεύθυνα για αυτά. Όταν ένα ερώτημα για μία δεικτοδοτήμενη οντότητα ξεκινάει, τότε εντοπίζεται ο κόμβος που είναι υπεύθυνος για το δείκτη και το ερώτημα προωθείται απευθείας στον κόμβο (ή κόμβους) που είναι υπεύθυνοι για το pivot key (ή pivot keys) της συγκεκριμένης τιμής. Εάν η δεικτοδοτημένη οντότητα εμφανίζεται ως ένα object και χρειάζεται να ερωτηθεί το local database, τότε οι κόμβοι που έλαβαν το ερώτημα απαντούν με τις τοποθεσίες του primary δακτυλιδιού που σχετίζονται με τη δεικτοδοτημένη οντότητα. Το ερώτημα προωθείται σε αυτούς τους κόμβους που απευθύνονται στο local database τους για την τελική επίλυση του ερωτήματος.

Όπως και στις προηγούμενες μεθόδους, οι κόμβοι που αποθηκεύουν μία δεικτοδοτημένη οντότητα στα local databases τους πρέπει να γνωρίζουν την ύπαρξη ενός δείκτη. Η διπλή κατεύθυνση των δεικτών εισάγεται μόνο για να εξασφαλιστεί η συνοχή των δεδομένων, αν και οι δείκτες είναι soft-state. Κατά τη διάρκεια των λειτουργιών re-indexing, οι τοποθεσίες των αποθηκευμένων πλειάδων αλλάζουν και οι δείκτες που σχετίζονται με αυτές τις πλειάδες πρέπει είτε να ενημερωθούν για τις νέες τοποθεσίες ή να διαγραφούν αποτρέποντας έτσι την ύπαρξη μη έγκυρων δεικτών. Από τις δύο αυτές εναλλακτικές έχει επιλεχθεί η διαγραφή τους, ώστε να αποφευχθεί η αύξηση της πολυπλοκότητας του συστήματος. Κάθε lookup για μία δεικτοδοτημένη τιμή ανανεώνει την τιμή του TTL και στις δύο πλευρές του δείκτη και μόνο οι έγκυροι δείκτες διαγράφονται κατά τη διάρκεια των λειτουργιών re-indexing.

Σύμφωνα με την κατανομή των triples στο Σχήμα 4.23, ένα ερώτημα της μορφής (“Building”, “SportsEvent”,?p) είναι ένα exact match ερώτημα που επιλύεται στο primary ring, όταν το ερώτημα φτάνει στον κόμβο που είναι υπεύθυνος για το pivot key `Building`. Από την άλλη πλευρά, ένα ερώτημα (“Stadium”,“SportsEvent”,?p) είναι ένα exact-match ερώτημα που επιλύεται σε ένα secondary δακτυλίδι και χρειάζεται να προωθηθεί στον κόμβο που είναι υπεύθυνος για το `Building` ώστε να αποτιμηθεί στο local database του. Όταν τίθεται το ερώτημα (“Stadium”,?p,?o)

για πρώτη φορά στο σύστημα, το ερώτημα αυτό αποστέλλεται σε όλους τους κόμβους του δικτύου. Όμως, τις επόμενες φορές το ερώτημα αυτό επιλύεται ως ένα indexed ερώτημα για όσο διάστημα ο δείκτης για το `Stadium` είναι έγκυρος. Τέλος, αξίζει να σημειωθεί ότι ένα ερώτημα

```
SELECT ?Stadium ?Event
WHERE { ?Stadium dbpedia:tenant ?Event.
?Stadium rdf:type  Building.
?Event rdf:type  Olympics.}
```

αναπαριστάται ως ένα ενιαίο ερώτημα με τη μορφή ("Building","Olympics","dbpedia:tenant") για να επιλυθεί από το σύστημα, αλλά ταυτόχρονα προσδιορίζεται στο ερώτημα ότι όλες οι οντότητες του συγκεκριμένου τύπου πρέπει να επιστραφούν ως απάντηση.

Τα ερωτήματα που εκτελούνται στο σύστημα μεταφράζονται στην εσωτερική αναπαράσταση που περιγράφηκε. Παρά το γεγονός ότι οποιαδήποτε προηγμένη τεχνική για τη βελτιστοποίηση των joins (join optimization) δεν έχει υλοποιηθεί ακόμα, όπως η χρήση στατιστικών, ιστογραμμάτων, κλπ., κάποια χαρακτηριστικά των περιγραφόμενων μηχανισμών δεικτοδότησης μπορούν να αξιοποιηθούν κατά την επίλυση των ερωτημάτων. Σύμφωνα με εγγενή χαρακτηριστικά των RDF και SPARQL, η επίλυση δύο γενικών κλάσεων ερωτημάτων αναλύεται. Μία μεγάλη κατηγορία αφορά τα ερωτήματα που αποτελούνται από "star-shaped" υποερωτήματα και συνδυάζουν διάφορα properties της ίδιας οντότητας. Σε αυτή την περίπτωση διαδοχικά lookups εκτελούνται για τις οντότητες υπό αναζήτηση μέχρι να βρεθεί κάποια από αυτές να εμφανίζεται είτε ως pivot key είτε ως δεικτοδοτημένη οντότητα. Όταν μία οντότητα βρεθεί, τότε ολόκληρο το ερώτημα αποτιμάται στο local database είτε κατευθείαν (εάν η οντότητα που βρέθηκε εμφανίζεται στο subject του ερωτήματος) είτε μετά από προώθηση. Μία άλλη μεγάλη κατηγορία είναι τα ερωτήματα με "εκφράσεις μονοπατιού" (path expressions), δηλαδή με *αλυσίδες* ("*chains*") από triple υποερωτήματα, όπου το object του πρώτου υποερωτήματος αποτελεί το subject του επόμενου υποερωτήματος, πάλι με τα ίδια properties. Εάν ένα triple περιλαμβάνει το property `rdf:type`, αυτό είναι ισοδύναμο με τη συγχώνευση του συγκεκριμένου triple με το αμέσως επόμενο και συνεπώς χρειάζεται να εκτελεστούν λιγότερα joins στα αποτελέσματα που ανακτώνται στο περιγραφόμενο σύστημα. Η στρατηγική που ακολουθείται για την αποτίμηση τέτοιων ερωτημάτων είναι η έναρξη της επίλυσης τους από το υποερώτημα που περιλαμβάνει οντότητες με συγκεκριμένες τιμές και για το οποίο χρειάζεται η ανάκτηση των αποτελεσμάτων με το μικρότερο αριθμό.

4.11 Προσαρμογή της Δεικτοδότησης ανάλογα με τα Ερωτήματα

Το PI4LD διατηρεί το βασικό χαρακτηριστικό που περιγράφηκε και για το *LinkedPeers*, δηλαδή να αντιλαμβάνεται τις τάσεις των ερωτημάτων. Για αυτό το λόγο, υλοποιούνται και σε αυτό

οι μηχανισμοί για τις λειτουργίες roll-up και drill-down. Η εκκίνηση της διαδικασίας για τη λήψη απόφασης σχετικά με τη χρησιμότητα μίας λειτουργίας re-indexing ξεκινάει μετά από ένα flood ή indexed ερώτημα. Αναλυτικότερα, όταν γίνεται flood ενός ερωτήματος, τότε όλοι οι κόμβοι της επικάλυψης λαμβάνουν το ερώτημα και αναζητούν τις δενδρικές δομές τους για να ανακαλύψουν εάν μία πιθανή λειτουργία re-indexing στο επίπεδο της ζητούμενης οντότητας είναι κατάλληλη σύμφωνα με την πληροφορία που διατηρούν. Εάν περισσότερες οντότητες περιέχονται στο ερώτημα, τότε είναι πιθανό να εκτελεστούν πολλαπλές και ταυτόχρονες λειτουργίες re-indexing σε διαφορετικά δακτυλίδια. Όταν επιλύεται ένα indexed ερώτημα από έναν κόμβο και ο συγκεκριμένος κόμβος έχει λάβει έναν προκαθορισμένο αριθμό από ερωτήματα βάσει ενός κυλιόμενου παραθύρου ερωτημάτων, τότε η πιθανότητα για re-indexing προς το αντίστοιχο επίπεδο της δεικτοδοτημένης οντότητας διερευνάται.

Εάν ένα drill-down προς ένα επίπεδο που βρίσκεται χαμηλότερα από ένα pivot level απαιτείται, τότε ο κόμβος που ανίχνευσε τη ζήτηση προχωράει στις απαιτούμενες ενέργειες, όπως περιγράφεται στην Ενότητα 4.5. Όταν μια τιμή που ρωτάται ανήκει σε ένα επίπεδο που βρίσκεται υψηλότερα από το pivot level, τότε κάθε κόμβος που λαμβάνει το flood ερώτημα ελέγχει εάν ένα roll-up θα είναι επικερδές σύμφωνα με τις τάσεις των εισερχόμενων ερωτημάτων και αν αυτό ισχύει, τότε καταγράφει την ένδειξη αυτή στην απάντηση του προς τον initiator του ερωτήματος. Όταν ο κόμβος initiator συλλέξει όλες τις απαντήσεις και ανακαλύψει ότι τουλάχιστον ένας κόμβος έχει ζητήσει να εξαταστεί η πιθανότητα ενός roll-up, τότε ο κόμβος initiator συλλέγει πληροφορία από όλους τους κόμβους και αποφασίζει εάν χρειάζεται ένα roll-up ή ένα group-drill-down σύμφωνα με τη διαδικασία που περιγράφηκε στις Ενότητες 3.5 και 4.5. Μία παρατήρηση σχετικά με το PI4LD αφορά το γεγονός ότι το ίδιο pivot key μπορεί να εμφανίζεται ως subject σε μερικά triples και ως object σε μερικά άλλα. Αυτό έχει ως αποτέλεσμα την ενημέρωση των links ανάμεσα στις primary και secondary διαστάσεις και προς τις δύο κατευθύνσεις, δηλαδή τόσο τα pivot levels για τη secondary διάσταση στις πλειάδες του local database όσο και οι εγγραφές των Linked Tables ενημερώνονται.

4.12 Πειραματική Αξιολόγηση

4.12.1 Γενική Περιγραφή της Πειραματικής Διαδικασίας

Το προτεινόμενο σύστημα υλοποιείται βασιζόμενο σε μία εκτενώς τροποποιημένη έκδοση του FreePastry [Fre], αν και θα μπορούσε να χρησιμοποιηθεί οποιοδήποτε πρωτόκολλο που εφαρμόζεται για τη δημιουργία των DHT επικαλύψεων. Οι κόμβοι του συστήματος φιλοξενούνται από διαφορετικά μηχανήματα με Xeon επεξεργαστή στα 2GHz και 8GB μνήμης, τα οποία έχουν εγκατεστημένο ένα 64-bit Debian Linux πυρήνα. Οι κόμβοι διασυνδέονται με Gigabit

Ethernet και επικοινωνούν με FreePastry sockets. Στη προεπιλεγμένη πειραματική διάταξη εκτελέστηκαν πειράματα σε μία επικάλυψη που αποτελείται από 16 κόμβους.

Τα local databases για κάθε κόμβο έχουν εγκατεστημένο το λογισμικό SQLite [SQL]. Το σχήμα που χρησιμοποιείται είναι ένας απλός πίνακας; ένα πιο πολύπλοκο σχήμα θα μπορούσε να χρησιμοποιηθεί ώστε να βελτιωθεί η απόδοση κατά την ανάκτηση των δεδομένων. Ωστόσο, ο στόχος της πειραματικής αξιολόγησης είναι η επίδειξη των πλεονεκτημάτων ενός ενοποιημένου και κατανεμημένου συστήματος δεικτοδότησης για πολλαπλές RDF αποθήκες. Η μελέτη της βελτίωσης της αποδοτικότητας του συστήματος λόγω των local databases είναι εκτός των στόχων της συγκεκριμένης αξιολόγησης.

Η προσέγγιση που υιοθετείται στο PI4LD συγκρίνεται με μια διάταξη που αποτελείται από μία κεντρική αποθήκη (που συμβολίζεται ως `VirtStore` στα Σχήματα), η οποία δημιουργήθηκε με χρήση του *Virtuoso Open-Source Edition version 6.1* [Vir], δηλαδή μίας κεντρικοποιημένης λύσης για αποθήκες των triples που προέρχονται από RDF δεδομένα. Η συγκεκριμένη αποθήκη είναι μία δημοφιλής λύση ανοικτού κώδικα για τέτοιου είδους δεδομένα, ενώ το λογισμικό Virtuoso χρησιμοποιείται ως μία back-end λύση για τη βάση δεδομένων του σημείου πρόσβασης του DBpedia [DBpa]. Για τη δημιουργία της κεντρικής αποθήκης χρησιμοποιείται η προεπιλεγμένη ρύθμιση του Debian αποθετηρίου (repository). Τα ερωτήματα στην αποθήκη γίνονται από έναν πελάτη που φιλοξενείται σε ένα διαφορετικό μηχάνημα. Επίσης, τα ερωτήματα εκτελούνται με τη χρήση του Virtuoso Jena provider [Jen], ενός πλήρως λειτουργικού Native Graph Model Storage Provider που επιτρέπει στις εφαρμογές του Σημασιολογικού Ιστού να θέτουν ερωτήματα απευθείας σε μία Virtuoso RDF αποθήκη. Πριν από την εκτέλεση μίας ομάδας ερωτημάτων, το σύστημα υποχρεώνεται να "αδειάσει" όλες τις προσωρινές μνήμες του (caches) και γίνεται επανεκκίνηση της βάσης δεδομένων για την εκκαθάριση των εσωτερικών, προσωρινών μνημών της. Κατά τη διάρκεια ενός συνόλου ερωτημάτων, η βάση δεδομένων χρησιμοποιεί κανονικά τις εσωτερικές, προσωρινές της μνήμες.

Η σύγκριση των δύο συστημάτων εκτελέστηκε με τη χρήση δύο RDF συνόλων δεδομένων (Ενότητα 4.12.2) και μετρήθηκαν οι χρόνοι απόκρισης για διαφορετικούς τύπους ερωτημάτων, όπως περιγράφεται στη συνέχεια (Ενότητα 4.12.3). Οι χρόνοι που αναπαριστώνται είναι οι μέσοι όροι 1000 επαναλήψεων, εκτός και αν δηλώνεται διαφορετικά.

4.12.2 Σύνολα Δεδομένων

Το LUBM Σύνολο Δεδομένων

Τα δεδομένα που χρησιμοποιούνται για τη βασική σύγκριση στα πειράματα που παρουσιάζονται είναι το Lehigh University benchmark [GPH04] (στη συνέχεια αναφέρεται ως LUBM). Το συγκεκριμένο μετροπρόγραμμα (benchmark) παράγει συνθετικά σύνολα δεδομένων οποιαδήποτε μεγέθους σύμφωνα με μία μοναδική και ρεαλιστική οντολογία, ενώ τα δεδομένα αυτά

αντιστοιχούν σε πληροφορία που συναντάται στον ακαδημαϊκό τομέα. Η LUBM οντολογία, αν και δεν είναι πολύ μεγάλη, χαρακτηρίζεται από την ιδιότητα της να περιέχει ένα μικρό αριθμό διαφορετικών τιμών στο γενικότερο επίπεδο, ενώ είναι εξαιρετικά "ευρεία" στο επίπεδο των φύλλων (δηλαδή στο επίπεδο των πραγματικών οντοτήτων). Το benchmark περιλαμβάνει και ένα σύνολο διαφορετικών κατηγοριών ερωτημάτων, οι οποίες διαφέρουν ως προς την πολυπλοκότητα και την επιλεκτικότητα (selectivity). Ένα σημαντικό χαρακτηριστικό των ερωτημάτων αυτών είναι ότι μερικές από τις κατηγορίες προϋποθέτουν να ληφθεί υπόψη η σχέση `rdfs:subClassOf` μεταξύ των εννοιών. Η συγκεκριμένη ιδιότητα αντιμετωπίζεται αυτόματα από το προτεινόμενο σύστημα. Για την πειραματική αξιολόγηση, χρησιμοποιείται ένα σύνολο δεδομένων που αποτελείται από 100 πανεπιστήμια με 18 διαφορετικά κατηγορήματα (predicates) και επομένως έχουν παραχθεί 13.5M triples. Το προεπιλεγμένο pivot level είναι το ℓ_2.

Το DBpedia Σύνολο Δεδομένων

Στην αξιολόγηση του συστήματος που παρουσιάζεται χρησιμοποιείται ένα πραγματικό σύνολο δεδομένων από το DBpedia πρόγραμμα [BLK+09]. Τα συγκεκριμένα δεδομένα περιέχουν πληροφορία που έχει εξαχθεί από το Wikipedia, ενώ έχει λάβει χώρα μία μεγάλη προσπάθεια για την κατηγοριοποίηση της πληροφορίας. Η συγκεκριμένη οντολογία [DBpb] αποτελεί μία "ρηχή", διατομειακή οντολογία, όπου η πλειοψηφία των κλάδων της έχουν το πολύ 3 διαφορετικά επίπεδα. Το σύνολο δεδομένων που χρησιμοποιήθηκε στα πειράματα αποτελείται από triples που ορίζουν τον τύπων των οντοτήτων (δηλαδή triples της μορφής $< \$object\ rdf:type\ \$class >$ που προέκυψαν από την εξαγωγή βάσει της οντολογίας) και triples που περιέχουν κατηγορίες από το namespace της οντολογίας. Τα triples που περιγράφουν συγκεκριμένα properties μίας οντότητας δεν έχουν συμπεριληφθεί, όπως titles, abstracts, images, geographic coordinates, κλπ., δεδομένου ότι αυτά τα triples αποθηκεύονται και ανακτώνται μόνο από τα local databases των κόμβων. Το σύνολο των δεδομένων αποτελείται από 6.7M triples για τους τύπους των οντοτήτων και 8M triples για τις υπόλοιπα predicates.

4.12.3 Περιγραφή των Ερωτημάτων

Τα LUBM Ερωτήματα

Ένα σύνολο εννέα ουσιαστικών ερωτημάτων δημιουργήθηκε βάσει του LUBM benchmark κατά τη σύγκριση του προτεινόμενου συστήματος με την κεντρική αποθήκη. Τα ερωτήματα αυτά δεν ευνοούν ιδιαίτερα κάποια συγκεκριμένη αποθηκευτική λύση και δεν απαιτούν πολύπλοκο

σχεδιασμό για την επίλυση τους [2]. Κάθε κατηγορία ερωτημάτων περιγράφεται λεπτομερώς στη συνέχεια:

Query 1 (LQ1): Η συγκεκριμένη κατηγορία ερωτημάτων στοχεύει στην εύρεση οποιοδήποτε person του τύπου `GraduateStudent`, το οποίο σχετίζεται με ένα συγκεκριμένο course (e.g., `GraduateCourse0`) σύμφωνα με τη σχέση `takesCourse`. Η επίλυση ενός τέτοιου ερωτήματος ξεκινάει από την αναζήτηση της τιμής `GraduateStudent`. Εάν δεν επιστραφούν αποτελέσματα, τότε η τιμή του course αναζητείται και αν δεν αντιστοιχεί σε pivot key, τότε το ερώτημα αυτό επιλύεται ως flood ερώτημα.

Query 2 (LQ2): Ένα ερώτημα της κατηγορίας αυτής αναφέρεται σε όλα τα publications που σχετίζονται μέσω του property `publicationAuthor` με μία συγκεκριμένη τιμή professor (e.g., `AssociateProfessor0`). Η διαφορά με το LQ1 είναι ότι η κλάση `Publication` έχει μία ευρεία ιεραρχία.

Query 3 (LQ3): Τα ερωτήματα αυτής της κατηγορίας στοχεύουν στην ανάκτηση της πληροφορίας που σχετίζεται με έναν professor που σχετίζεται μέσω του property `worksFor` με ένα συγκεκριμένο department ενός university (i.e. `Department0`). Επίσης, ερωτώνται πολλαπλά properties μίας κλάσης και τα περισσότερα από αυτά τα properties χαρακτηρίζονται από το ότι δε διασυνδέσουν ένα subject με άλλες οντότητες: για αυτό το λόγο η σχετική πληροφορία αποθηκεύεται μόνο τοπικά στον κόμβο του συγκεκριμένου professor (π.χ., το property για το `name` και το αντίστοιχο για το `emailAddress`). Επίσης, το ερώτημα αυτό προϋποθέτει ότι λαμβάνεται υπόψη η σχέση `rdf:subClass` μεταξύ της κλάσης `Professor` και των υποκλάσεων της. Η διαδικασία της επίλυσης ξεκινάει από την αναζήτηση των τιμών που περιέχουν το property `worksFor`.

Query 4 (LQ4): Η κατηγορία αυτή περιέχει ερωτήματα για την εύρεση όλων των `Professors` που συνδέονται με το `worksFor` property με ένα συγκεκριμένο department ενός university (π.χ., `University0`) και επιλέγει κάποια οντότητα τύπου `Professor` μόνο εάν είναι και `Chair` ενός department. Αξίζει να σημειωθεί εδώ, ότι το `Department` και το `University` βρίσκονται στο ίδιο επίπεδο στη LUBM οντολογία. Κατά την επίλυση του συγκεκριμένου τύπου ερωτήματος εντοπίζονται πρώτα όλα τα departments ενός συγκεκριμένου university και στη συνέχεια αναζητούνται όλες οι οντότητες τύπου `Chairs` του κάθε department.

Query 5 (LQ5): Σε ένα ερώτημα αυτής της κατηγορίας εντοπίζονται όλα τα persons που συνδέονται με το property `memberOf` με ένα συγκεκριμένο department (π.χ., `Department0`) ενός university. Η εκτέλεση αυτού του ερωτήματος μπορεί να γίνει "παράλληλα", εάν το ερώτημα χωριστεί σε δύο υποερωτήματα που αντιστοιχούν στις υποκλάσεις του `Person`. Δεδομένου ότι οι περισσότερες οντότητες του LUBM συνόλου δεδομένων κληρονομούν την κλάση `Person`, η συγκεκριμένη τιμή έχει αφαιρεθεί από το root level. Αυτό έγινε ώστε να αποφευχθεί η δημιουργία

[2] Οι τεχνικές σχεδιασμού για την επίλυση των ερωτημάτων (query planning techniques) μπορούν να εφαρμοστούν πάνω από το προτεινόμενο σύστημα δεικτοδότησης. Το συγκεκριμένο σύστημα δε στοχεύει σε ερωτήματα που απαιτούν τέτοιες στρατηγικές.

μίας ιεραρχίας πολλαπλών επιπέδων, όπου συγκεντρώνεται η πλειοψηφία των triples σε έναν μόνο κόμβο εάν εκτελεστεί μία λειτουργία roll-up προς το root level. Τα αποτελέσματα για κάθε υποερώτημα επιστρέφονται στον κόμβο που έθεσε το ερώτημα και η ένωση τους παρουσιάζεται στο χρήστη.

Query 6 (LQ6): Η κατηγορία αυτή αφορά την εύρεση όλων των οντοτήτων μίας συγκεκριμένης κλάσης ή υποκλάσης, ενώ οι οντότητες αυτές μπορεί να εμφανίζονται είτε ως subject είτε ως object. Για παράδειγμα, ο στόχος θα μπορούσε να είναι η εύρεση όλων των μελών της κλάσης `Student` ή της υποκλάσης `UndergraduateStudent`. Η επίλυση του ερωτήματος παρουσιάζει τη μικρότερη πολυπλοκότητα, αφού μόνο οι δενδρικές δομές που περιέχουν αυτήν την τιμή πρέπει να εντοπιστούν και να επιστραφούν οι οντότητες του χαμηλότερου επιπέδου.

Query 7 (LQ7): Ο σκοπός ενός τέτοιου ερωτήματος είναι η ανακάλυψη όλων των `Students` που συνδέονται με το property `takesCourse` με τα courses που διδάσκονται από ένα συγκεκριμένο professor (π.χ., `AssociateProfessor0`). Σε αυτήν την κατηγορία το `Course` εμφανίζεται πάντα ως object στα ερωτήματα. Κατά την επίλυση του ερωτήματος, βρίσκονται όλα τα courses ενός συγκεκριμένου proffessor αρχικά. Στη συνέχεια, ανακτώνται οι οντότητες που είναι τύπου student και σχετίζονται με τα courses που έχουν βρεθεί.

Query 8 (LQ8): Η κατηγορία αυτή είναι παρεμφερής με την προηγούμενη κατηγορία, αλλά παρουσιάζει αυξημένη πολυπλοκότητα. Ένα ερώτημα της αναζητά όλες τις οντότητες τύπου `Student` που είναι μέλη όλων των `Departments`, τα οποία με τη σειρά τους συνδέονται με το `subOrganization` με ένα συγκεκριμένο university. Σε αυτό το ερώτημα, το department εμφανίζεται και ως subject και ως object. Για την επίλυση του ερωτήματος, αποστέλλεται αρχικά ένα υποερώτημα για την εύρεση όλων των departments του συγκεκριμένου university που προσδιορίζεται στο ερώτημα. Στη συνέχεια, εκτελείται ένα υποερώτημα για την ανάκτηση των students των departments που βρέθηκαν στο προηγούμενο υποερώτημα. Εφόσον, η τιμή `Student` αντιστοιχεί σε root key των primary keys, το υποερώτημα αυτό επιλύεται πάντα στην primary διάσταση. Ο υπάρχον root index (εάν η τιμή `Student` δεν αντιστοιχεί η ίδια σε root key) προωθεί το υποερώτημα στους κόμβους που είναι υπεύθυνοι για τα triples που περιέχουν την τιμή Student ως subject; οι κόμβοι αυτοί ρωτούν τα local databases τους για την ανάκτηση των οντοτήτων που είναι τύπου `Student` και συνδέονται με τα courses που βρέθηκαν.

Query 9 (LQ9): Η κατηγορία αυτή περιέχει ερωτήματα που αναζητούν όλες τις οντότητες τύπου student που συνδέονται μέσω του `takesCourse` με ένα συγκεκριμένο course (π.χ., `GraduateCourse0`). Ένα τέτοιο ερώτημα επιλύεται είτε ως indexed ερώτημα είτε ως exact match ερώτημα στο primary δακτυλίδι, όταν έχει χρησιμοποιηθει το root level ως pivot level κατά τις αρχικές εισαγωγές είτε όταν τα δέντρα με την τιμή `Student` έχουν εκτελέσει μία λειτουργία roll-up προς το root level.

Τα DBpedia ερωτήματα

Λαμβάνοντας υπόψη ότι δεν υπάρχει κάποιο benchmark για τα DBpedia δεδομένα, τα ερωτήματα του FedBench benchmark suite [SGH+11] που αναφέρονται στο DBpedia έχουν προσαρμοστεί. Τα ερωτήματα αυτά επικεντρώνονται στην ανάλυση της αποτελεσματικότητας των federated στρατηγικών επίλυσης ερωτημάτων για σημασιολογικά δεδομένα. Λαμβάνοντας υπόψη τα σενάρια του FedBench σχεδιάστηκε μία ομάδα από έξι διαφορετικά ερωτήματα με διαφορετικό μέγεθος αποτελεσμάτων:

Query 1 (DBQ1): Αυτή είναι μία κατηγορία απλών ερωτημάτων που αναζητά όλες τις οντότητες που κληρονομούν την κλάση `President`, η οποία ανήκει στο επίπεδο ℓ_2 της DBpedia οντολογίας.

Query 2 (DBQ2): Ο στόχος ενός τέτοιου ερωτήματα είναι η εύρεση όλων των οντοτήτων τύπου `President` που έχουν ένα συγκεκριμένο nationality (e.g., `United States`) και τα political parties στα οποία ανήκουν.

Query 3 (DBQ3):Ένα ερώτημα αυτής της κατηγορίας βρίσκει κάθε οντότητα τύπου `Politician` και τις ομαδοποιεί ανάλογα με το `Country` του οποίου είναι residence.

Query 5 (DBQ5): Ο σκοπός ενός ερωτήματος αυτής της κατηγορίας είναι η ανάκτηση όλων των οντοτήτων που είναι τύπου `AmericanFootballPlayer` μαζί με όλες τις οντότητες τύπου `Team` που σχετίζονται και τις οντότητες τύπου `City` που συνδέονται με τις οντότητες τύπου `Team`. Αυτό το ερώτημα μπορεί να χωριστεί σε δύο διαφορετικά ερωτήματα τα οποία θα εκτελεστούν παράλληλα. Το πρώτο ερώτημα θα ανακτήσει όλους τους παίκτες και τις ομάδες, ενώ το δεύτερο συλλέγει τα αποτελέσματα για όλα τα teams και τα αντίστοιχα cities. Όταν βρεθούν τα αποτελέσματα και για τα δύο ερωτήματα, τότε αυτά συγχωνεύονται στον κόμβο που έθεσε το ερώτημα.

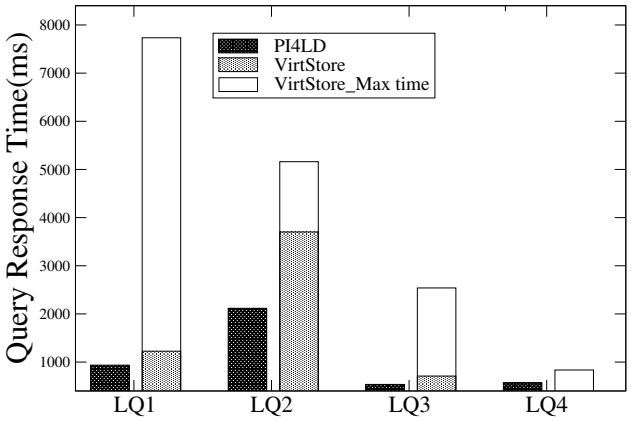
Query 6 (DBQ6):Ένα ερώτημα αυτής της κατηγορίας αφορά μόνο τις οντότητες τύπου `Team` που συνδέονται με μια συγκεκριμένη οντότητα τύπου `Team` (π.χ., `Baltimore`).Ένα ερώτημα αυτής της κατηγορίας μπορεί να θεωρηθεί ως υποερώτημα της προηγούμενης κατηγορίας.

4.12.4 Μελέτη της Απόδοσης για τις Διάφορες Κατηγορίες Ερωτημάτων

Ο χρόνος απόκρισης των ερωτημάτων παρουσιάζεται στα Σχήματα 4.24, 4.25 και 4.27 για όλες τις κατηγορίες ερωτημάτων που περιγράφηκαν στην Ενότητα 4.12.3. Για αυτά τα πειράματα παράχθηκαν ερωτήματα όπου έχουν διάφορες τιμές για τις σταθερές που ρωτιούνται (π.χ., `GraduateCourse0`, `AssociateProfessor0`, κλπ.), οι οποίες επιλέχθηκαν τυχαία από τους απογόνους που σχετίζονται με τον τύπο που ορίζεται στην αντίστοιχη κατηγορία ερωτημάτων. Επίσης, υπολογίστηκαν οι μέσοι χρόνοι απόκρισης για 1000 επαναλήψεις. Ένας πελάτης (client) θέτει ερωτήματα και στα δύο συστήματα και περιμένει μέχρι να επιστρέψουν τα αποτελέσματα πριν θέσει ένα νέο ερώτημα. Για την κεντρικοποιημένη προσέγγιση καταγράφεται και ο μέγιστος

χρόνος απόκρισης ανά ερώτημα: ο χρόνος αυτός αντιστοιχεί στο χρόνο που απαιτείται για την πρώτη ερώτηση, όπου το σύστημα δεν έχει αποθηκεύσει προσωρινά κάποια αποτελέσματα ώστε να απαντά γρηγορότερα στα ερωτήματα που ακολουθούν.

Query Cat.	LQ1	LQ2	LQ3	LQ4
Avg. rows	4	14	29	20

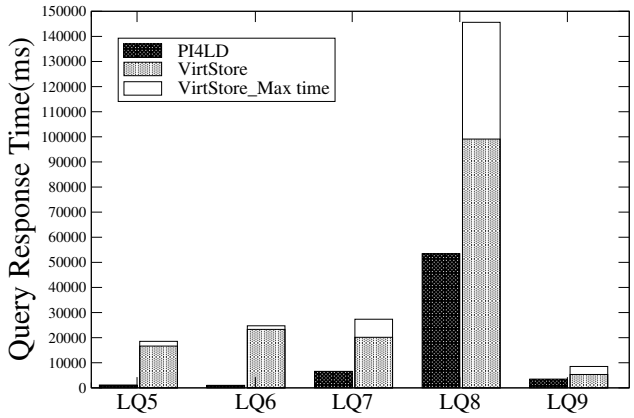


Σχήμα 4.24: *Σύγκριση των χρόνων απόδοσης(ms) για τις κατηγορίες ερωτημάτων 1–4 του LUBM benchmark*

Το Σχήμα 4.24 απεικονίζει το χρόνο απόκρισης για τις κατηγορίες των ερωτημάτων που είναι κυρίως της μορφής *(?s, o, p)*, όπου το subject ζητείται να είναι ενός συγκεκριμένου τύπου. Επιπλέον, αυτές οι κατηγορίες ερωτημάτων έχουν μικρή είσοδο και μεγάλο selectivity. Στο Σχήμα 4.25, τα ερωτήματα των κατηγοριών που παρουσιάζονται είναι περισσότερο πολύπλοκα και επιστρέφουν μεγαλύτερο αριθμό αποτελεσμάτων. Ο μέσος αριθμός των triples που επιστρέφονται φαίνεται στους αντίστοιχους πίνακες πάνω από τις γραφικές παραστάσεις. Τα αποτελέσματα δείχνουν ότι το PI4LD επιτυγχάνει καλύτερους χρόνους απόκρισης για όλες τις κατηγορίες ερωτημάτων.

Το PI4LD καταφέρνει να επιλύσει τα ερωτήματα αποτελεσματικά ακόμα και όταν αυξάνει η πολυπλοκότητα τους, όπως φαίνεται για τα ερωτήματα LQ7 και LQ8, τα οποία είναι τα ερωτήματα που χρειάζονται περισσότερα joins. Επιπλέον, κάθε κόμβος αποθηκεύει ένα μικρότερο τμήμα επί του συνόλου των δεδομένων και αυτό συμβάλλει στη γρηγορότερη επεξεργασία τους. Δεδομένου ότι το PI4LD ενσωματώνει τη συνολική πληροφορία της ιεραρχίας, λιγότερα lookups εκτελούνται για να ανακαλυφθεί εάν μία οντότητα που βρέθηκε είναι του ζητούμενου τύπου. Τα ερωτήματα επιλύονται 2.7 και 1.8 φορές γρηγορότερα σε σύγκριση με το VirtStore. Το LQ5 είναι μία άλλη κατηγορία που ευνοείται από αυτή την κατανεμημένη προσέγγιση, επειδή το ερώτημα μπορεί να χωριστεί σε δύο ξεχωριστά ερωτήματα που εκτελούνται παράλληλα με αποτέλεσμα γρηγορότερο χρόνο απόκρισης κατά 15 φορές. Σημαντική επιτάχυνση καταγράφεται και για

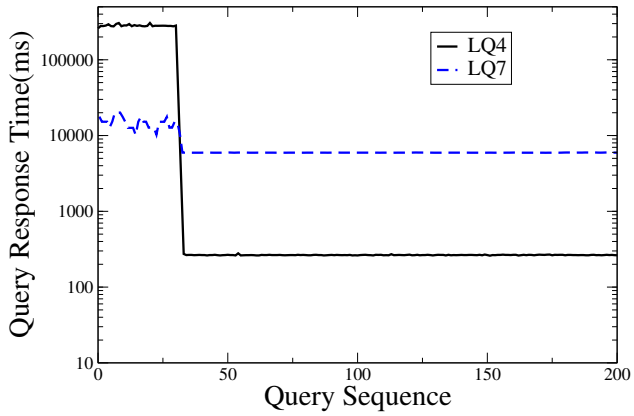
Query Cat.	LQ5	LQ6	LQ7	LQ8	LQ9
Avg. rows	625	956711	44	11532	20



Σχήμα 4.25: *Σύγκριση των χρόνων απόδοσης(ms) για τις κατηγορίες ερωτημάτων 5–9 του LUBM benchmark*

την κατηγορία LQ6, η οποία περιέχει τα απλούστερα ερωτήματα που μπορούν να τεθούν στο σύστημα, δηλαδή τα ερωτήματα που απαντιώνται κατευθείαν από τις δενδρικές δομές.

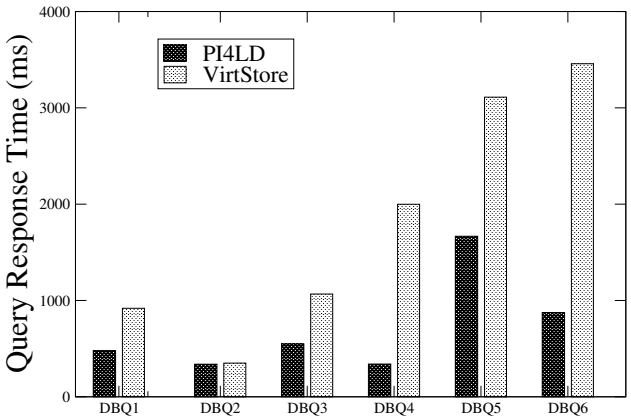
Η επιλογή του αρχικού επίπεδου εισαγωγής επηρεάζει την απόδοση του συστήματος για την επίλυση ενός ερωτήματος μόνο μέχρι να διαδραματιστούν οι απαιτούμενες λειτουργίες re-indexing. Το Σχήμα 4.26 δείχνει το χρόνο που απαιτείται για την απάντηση των LQ4 και LQ7 ως συνάρτηση την σειράς των ερωτημάτων, ενώς το ℓ_1 έχει επιλεχθεί ως αρχικό επίπεδο εισαγωγής. Το σύστημα έχει ρυθμιστεί έτσι ώστε να επιτρέπει την εκτέλεση των λειτουργιών roll-up και drill-down αφού έχουν ολοκληρωθεί τριάντα ερωτήματα. Όπως επιδεικνύεται, οι λειτουργίες roll-up που εκτελούνται στα δέντρα που ρωτιούνται έχουν ως αποτέλεσμα σημαντική επιτάχυνση της



Σχήμα 4.26: *Η επίδραση των προσαρμοστικών λειτουργιών re-indexing στο χρόνο απόκρισης των LQ4 και LQ7, όταν το ℓ_1 επιλέγεται ως αρχικό επίπεδο εισαγωγής*

επίλυσης των ερωτημάτων και για τις δύο κατηγορίες ερωτημάτων. Μετά από τη διαδικασία re-indexing, η απόδοση του συστήματος παραμένει σταθερή για τα υπόλοιπα ερωτήματα.

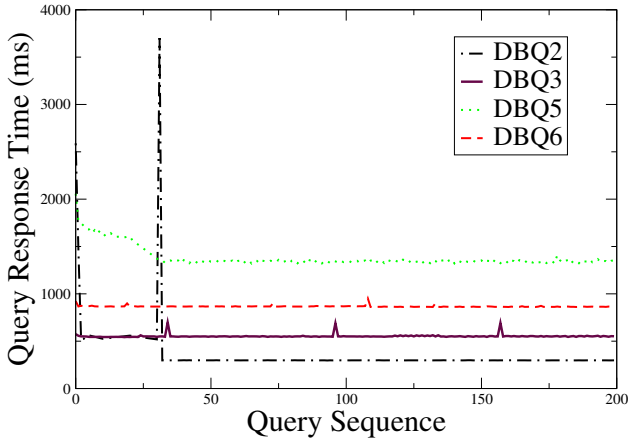
Query Cat.	DBQ1	DBQ2	DBQ3	DBQ4	DBQ5	DBQ6
Avg. rows	2239	13	428	1279	7041	303



Σχήμα 4.27: *Σύγκριση των χρόνων απόδοσης(ms) για κάθε κατηγορία ερωτημάτων του DBpedia dataset*

Στο Σχήμα 4.27 φαίνονται οι χρόνοι απόκρισης για τις κατηγορίες ερωτημάτων που αναφέρονται στα DBpedia δεδομένα. Στο σύστημα έχει οριστεί να επιτρέπεται η εκτέλεση λειτουργιών re-indexing μετά από τριάντα ερωτήματα που έχουν στοχεύσει το ίδιο δέντρο. Εφόσον στη διάταξη του VirtStore η βάση δεδομένων αποθηκεύει προσωρινά τα αποτελέσματα για ένα ερώτημα που έχει απαντηθεί, γίνεται επανεκκίνηση της βάσης δεδομένων πριν την εκτέλεση ενός ερωτήματος. Το PI4LD καταφέρνει να απαντά γρηγορότερα σε σύγκριση με την κεντρικοποιημένη διάταξη σε όλες τις κατηγορίες. Οι χρόνοι απόκρισης που μετρώνται στο PI4LD εξαρτώνται από τον τύπο του ερωτήματος και από το μέγεθος των αποτελεσμάτων που ανακτώνται. Ο μηχανισμός του PI4LD που χρησιμοποιείται για την επίλυση των ερωτημάτων παίζει εξίσου σημαντικό ρόλο στους χρόνους απόκρισης. Η επιρροή αυτού του παράγοντα φαίνεται με περισσότερη λεπτομέρεια στο Σχήμα 4.28, όπου όλα τα ερωτήματα περιλαμβάνουν δύο οντότητες. Οι οντότητες του DBQ2 δεν έχουν δεικτοδοτηθεί κατά την αρχική εισαγωγή των δεδομένων και αυτό έχει ως αποτέλεσμα στο να γίνεται flood του ερωτήματος αρχικά. Στα ακόλουθα ερωτήματα, η οντότητα που αντιστοιχεί στο subject είναι δεικτοδοτημένη μέχρι να εκτελεστεί ένα drill-down προς αυτό το επίπεδο (στο σημείο που εμφανίζεται ο μεγαλύτερος χρόνος απόκρισης). Μετά το drill-down όλα τα ερωτήματα επιλύονται ως exact match ερωτήματα στο primary δακτυλίδι. Εν αντιθέσει, η οντότητα `Politician` στο DBQ3 αντιστοιχεί σε pivot key κατά τις αρχικές εισαγωγές και για αυτό το λόγο ο χρόνος απόκρισης για τα ερωτήματα αυτής της κατηγορίας είναι σταθερός. Τα DBQ5 και DBQ6 μπορούν να θεωρηθούν ως παρεμφερή ερωτήματα, όσον αφορά το ερώτημα για την ίδια οντότητα που εμφανίζεται ως subject. Στο DBQ5, η τιμή `SportsTeam` που προσδιορίζεται ως

object χρησιμοποιείται ως pivot key κατά την αρχική εισαγωγή των δεδομένων και για αυτό το λόγο το DBQ5 επιλύεται πάντα ως exact match ερώτημα σε secondary διάσταση. Αντιθέτως, η τιμή `SportsTeam` εμφανίζεται ως subject στο DBQ6 και επομένως όλα τα ερωτήματα επιλύονται ως exact match στην primary διάσταση.

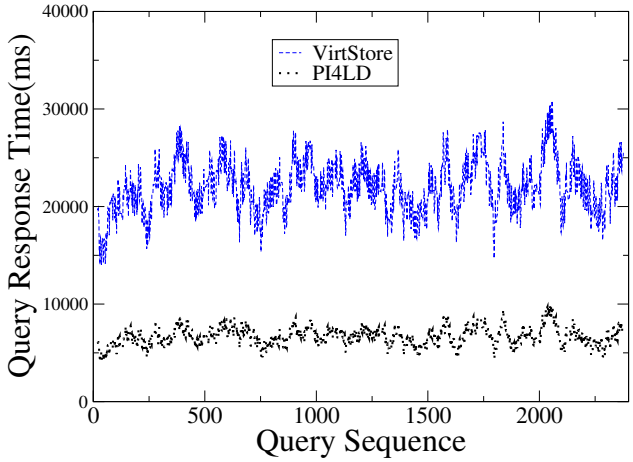


Σχήμα 4.28: *Οι χρόνοι απόκρισης για κάποιες επιλεγμένες κατηγορίες ερωτημάτων του DBpedia*

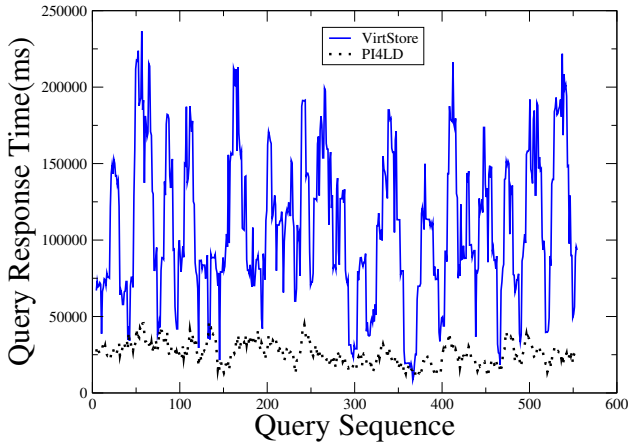
4.12.5 Ταυτόχρονα Ερωτήματα

Η απόδοση και των δύο συστημάτων μελετάται για αυξανόμενο φορτίο ερωτημάτων που θέτουν παράλληλοι χρήστες, όπως συμβαίνει στις διαδικτυακές εφαρμογές που αποθηκεύουν διασυνδεδεμένα δεδομένα. Για την επίτευξη αυτού του στόχου παράγεται ένα φορτίο ερωτημάτων που αποτελείται από 2500 ερωτήματα, τα οποία έχουν επιλεχθεί τυχαία. Αρχικά, τα ερωτήματα αποστέλλονται στα συστήματα μέσω ενός μόνο κόμβου-πελάτη. Οι μέσοι χρόνοι απόκρισης παρουσιάζονται στο Σχήμα 4.29. Το προτεινόμενο σύστημα παρουσιάζει καλύτερη απόδοση σε σχέση με την κεντρικοποιημένη προσέγγιση και αποδεικνύεται γρηγορότερο κατά 5 φορές, ενώ οι χρόνοι απόκρισης συμφωνούν με αυτούς που καταγράφηκαν για κάθε κατηγορία.

Στο Σχήμα 4.30, ο μέσος χρόνος απόκρισης καταγράφεται όταν τα ίδια ερωτήματα αποστέλλονται από πέντε κόμβους-πελάτες ταυτόχρονα. Το PI4LD καταφέρνει να διατηρήσει τους γρηγορότερους ρυθμούς απόκρισης σε επίπεδα σχεδόν ίσα με αυτά που εμφανίζονται στο Σχήμα 4.29. Η κατανομή των δεδομένων ανάμεσα στους κόμβους του PI4LD συνεισφέρει σε αυτό το αποτέλεσμα, δεδομένου ότι η παράλληλη επεξεργασία των ερωτημάτων από διαφορετικούς κόμβους την ίδια χρονική στιγμή (ακόμη και αν κάθε κόμβος επεξεργάζεται μόνο ένα ερώτημα στο σύστημα που υλοποιήθηκε κάθε φορά). Επιπλέον, κάθε κόμβος διατηρεί ένα μικρότερο τμήμα των συνολικών δεδομένων και έτσι απαιτείται λιγότερος χρόνος για την επεξεργασία τους.



Σχήμα 4.29: *Χρόνος απόκρισης για κάθε ερώτημα ενός φορτίου που περιλαμβάνει τυχαίως επιλεγμένα ερωτήματα για τις LUBM κατηγορίες*

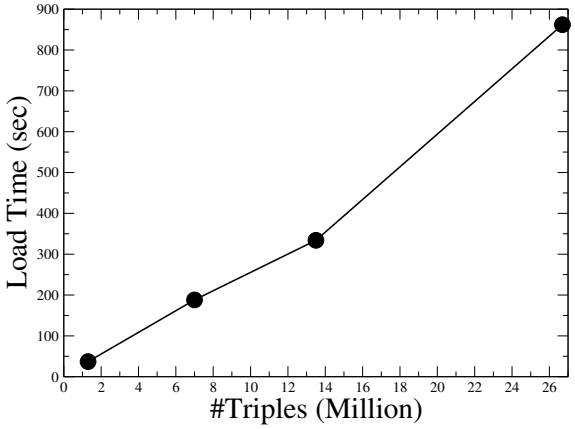


Σχήμα 4.30: *Χρόνοι απόκρισης για ένα φορτίο ερωτημάτων που περιλαμβάνει τυχαίως επιλεγμένα ερωτήματα για τις LUBM κατηγορίες, τα οποία αποστέλλονται από 5 πελάτες*

4.12.6 Επεκτασιμότητα του Συστήματος

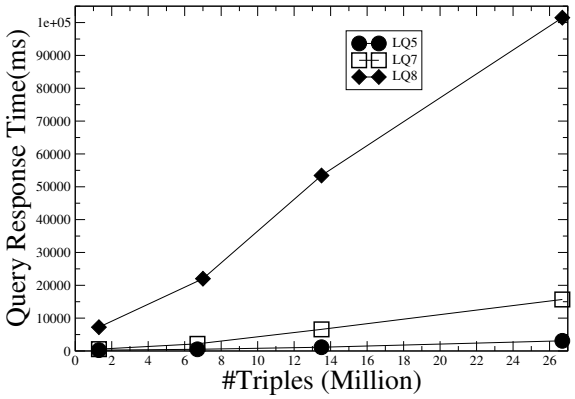
Η επεκτασιμότητα του συστήματος δοκιμάστηκε για μεγαλύτερα σύνολα δεδομένων. Το Σχήμα 4.31 παρουσιάζει τους χρόνους φόρτωσης για την εισαγωγή τεσσάρων διαφορετικών συνόλων δεδομένων που παράχθηκαν για 10, 50, 100 και 200 πανεπιστήμια αντίστοιχα. Τα δεδομένα ταξινομούνται και ομάδες από triples με το ίδιο pivot key στην primary διάσταση εισάγονται στο PI4LD. Ένα χαρακτηριστικό που επιβεβαιώνεται από αυτό το πείραμα είναι ότι ο χρόνος φόρτωσης (load time) αυξάνει γραμμικά με το μέγεθος του συνόλου δεδομένων.

Η επίπτωση του όγκου των δεδομένων μελετάται και για τους χρόνους απόκρισης τριών αντιπροσωπευτικών κατηγοριών ερωτήσεων του LUBM. Οι κατηγορίες των ερωτημάτων που φαίνονται στο Σχήμα 4.32 επιλέχθηκαν επειδή είναι πολυπλοκότερες και απαιτείται κατά την



Σχήμα 4.31: *Χρόνος φόρτωσης για διαφορετικά μεγέθη των LUBM συνόλων δεδομένων (πανεπιστήμια 10, 50, 100 και 200 αντίστοιχα)*

επίλυση τους η πληροφορία που σχετίζεται με τις οντολογίες και βρίσκεται στις ιεραρχικές δομές του PI4LD. Συμπερασματικά, τα ερωτήματα αυτά δε μπορούν να επιλυθούν από απλά DHT lookups για τις hashed τιμές είτε του subject είτε του object. Επίσης, διαφορετικά χαρακτηριστικά του συστήματος μπορούν να χρησιμοποιηθούν κατά την επίλυση των ερωτημάτων κάθε κατηγορίας: το LQ5 μπορεί να χωριστεί σε υποερωτήματα που εκτελούνται παράλληλα, ενώ τα LQ7 και LQ8 είναι path ερωτήματα που απαιτούν πολλαπλά joins. Η αύξηση του μεγέθους του συνόλου δεδομένων προκαλεί κάποια αύξηση στους χρόνους απόκρισης για τα LQ7 και LQ8, αλλά η κλίση είναι μικρή. Οι χρόνοι απόκρισης είναι παρατηρούνται πιο αυξημένοι για το LQ8, δεδομένου ότι το ερώτημα αυτό ανακτά ένα σημαντικά μεγαλύτερο αριθμό από triples σε σύγκριση με το LQ7 και για αυτό ο χρόνος επεξεργασίας του επηρεάζεται από τον όγκο των δεδομένων.



Σχήμα 4.32: *Μέσοι χρόνοι απόκρισης των LUBM ερωτημάτων για διαφορετικά μεγέθη των συνόλων δεδομένων*

ΚΕΦΑΛΑΙΟ 5

Συνολική Επισκόπηση και Μελλοντικές Επεκτάσεις

The world is a fine place, and worth fighting for.

Ernest Hemingway

Πολλές προκλήσεις παραμένουν ανοικτές στη διαχείριση δεδομένων σε κατανεμημένα περιβάλλοντα. Η εξέλιξη των υπολογιστικών συστημάτων καθιστά εφικτή την ανάπτυξη εφαρμογών που δε μπορούσαν καν να υπάρξουν πριν μερικά χρόνια. Επίσης, ανοίγονται νέες προοπτικές για "παραδοσιακές" εφαρμογές με περιορισμένη λειτουργικότητα λόγω της έλλειψης υπολογιστικών πόρων, π.χ. πόρων που προσφέρουν επεξεργαστική ισχύ και αποθηκευτικό χώρο. Η κατανεμημένη διαχείριση δεδομένων είναι μία από τις κύριες, επικρατούσες τάσεις που υπαγορεύεται από την έλευση των διαδικτυακών εφαρμογών, την παγκοσμιοποίηση των αγορών, την αυτοματοποίηση των επιχειρησιακών διεργασιών και την αυξημένη χρήση αισθητήρων.

Η υιοθέτηση των DHT επικαλύψεων συμβάλει στο σχεδιασμό επεκτάσιμων κι ανεκτικών σε σφάλματα υποδομών που μπορούν να αντεπεξέλθουν σε εφαρμογές με αυξημένες απαιτήσεις για διαχείριση δεδομένων με πλήρως κατανεμημένο τρόπο. Ωστόσο, οι DHT επικαλύψεις θέτουν κάποιους περιορισμούς στον τρόπο αναζήτησης σε τέτοιου είδους υποδομές και απαιτούνται επιπρόσθετες μέθοδοι για τη δεικτοδότηση και την αναζήτηση αντικειμένων ώστε να αρθούν αυτοί οι περιορισμοί.

Στην παρούσα διατριβή, η πρόσβαση σε πληροφορία που δημοσιοποιείται από ετερογενείς πηγές και η βελτίωση της ποιότητας των μεθόδων αναζήτησης θεωρούνται σημαντικά προβλήματα για περαιτέρω διερεύνηση. Η οργάνωση, η δεικτοδότηση και η αναζήτηση δεδομένων μελετώνται με στόχο τη δημιουργία τεχνικών που θα επιταχύνουν την επεξεργασία των ερωτημάτων. Για αυτό το λόγο, οι ιδιότητες της δομής των δεδομένων λαμβάνονται υπόψη κατά το σχηματισμό των στρατηγικών για την τοποθέτηση και την οργάνωση των δεδομένων ανάμεσα στους κόμβους που συμμετέχουν στην επικάλυψη. Οι περιγραφόμενοι αλγόριθμοι και τεχνικές επικεντρώνονται σε εννοιολογικές ιεραρχίες. Επίσης, το πολυδιάστατο μοντέλο δεδομένων εξερευνάται στις προτεινόμενες μεθόδους, αλλά η υποστήριξη μερικώς δομημένων δεδομένων διαφοροποιεί τη συγκεκριμένη προσέγγιση από άλλες υπάρχουσες. Ο βασικός τύπος ερωτημάτων που υποστηρίζονται είναι για τον εντοπισμό οποιασδήποτε τιμής σε οποιοδήποτε επίπεδο των αποθηκευμένων ιεραρχιών. Ακόμα, δίνεται έμφαση στην υποστήριξη απλών ερωτημάτων σύνοψης και ερωτημάτων σύνοψης σε πολλαπλές διαστάσεις για διάφορα επίπεδα λεπτομέρειας.

Μία άλλη βασική συμβολή της συγκεκριμένης διατριβής είναι η προσαρμοστική συμπεριφορά των τεχνικών δεικτοδότησης στις τάσεις που παρατηρούνται στα ερωτήματα των χρηστών στο επίπεδο του κόμβου. Όσο το μέγεθος των δεδομένων αυξάνει ραγδαία, η εξαντλητική αναζήτηση των τιμών δε μπορεί να διεκπεραιωθεί κατά τη διάρκεια ενός αποδεκτού χρονικού διαστήματος και οι μηχανισμοί δεικτοδότησης χάνουν την αποτελεσματικότητα τους. Εάν ένα σύστημα έχει την ικανότητα να προσαρμόζεται στις απαιτήσεις των χρηστών δυναμικά, τότε τα οφέλη στη ποιότητα και την ταχύτητα των αναζητήσεων μπορεί να είναι σημαντικά. Επιπλέον, μελέτες σε διαδικτυακές εφαρμογές δείχνουν ότι τα ασύμμετρα φορτία ερωτημάτων (για παράδειγμα αυτά που ακολουθούν τη zipfian κατανομή) είναι μία συνήθης περίπτωση και για αυτό το λόγο ένα ευέλικτο σύστημα που είναι ικανό να οργανώνει τις δομές του στις ξαφνικές αλλαγές των ερωτημάτων παρουσιάζει πλεονεκτήματα.

Ένα από τα αποτελέσματα αυτής της διατριβής είναι μία ιδιαιτέρως προσαρμοστική, επεκτάσιμη και online τεχνική για την οργάνωση και δεικτοδότηση δεδομένων που περιγράφονται από εννοιολογικές ιεραρχίες. Το σύστημα που υλοποιεί τις προτεινόμενες τεχνικές στοχεύει στην κατανομή μεγάλου όγκου δεδομένων σε μία DHT επικάλυψη με τέτοιο τρόπο που να διατηρούνται τα σημασιολογικά στοιχεία που περιέχονται στην ιεραρχία. Ένα σημαντικό στοιχείο της παρουσιαζόμενης μεθόδου είναι η προσαρμοστικότητα της και η έλλειψη συνολικού συντονισμού σε ολόκληρο το σύστημα: Οι αποθηκευμένες ιεραρχίες δεικτοδοτούνται σε διαφορετικά επίπεδα σύμφωνα με το βαθμό λεπτομέρειας των εισερχόμενων ερωτημάτων. Με τη χρήση περιορισμένης γνώσης, οι κόμβοι μπορούν να αποφασίσουν για το επίπεδο δεικτοδότησης των αποθηκευμένων πλειάδων, ώστε να μειωθεί το φαινόμενο προώθησης ενός ερωτήματος σε όλους τους κόμβους του δικτύου. Επιπλέον, δεν υπάρχει ανάγκη για αναβολή της λειτουργίας του συστήματος όταν

ενημερώνονται τα δεδομένα, αφού το σύστημα είναι σε θέση να ενημερώνει διαρκώς τα δεδομένα του με μικρό κόστος. Η πειραματική αξιολόγηση για πολλαπλά, δυναμικά φορτία δεδομένων επιβεβαιώνει τις ιδιότητες των προτεινόμενων μεθόδων: Το τελικό σύστημα καταφέρνει να απαντήσει την πλειοψηφία των ερωτημάτων αποδοτικά με λίγα μηνύματα. Είναι ιδιαίτερα αποδοτικό για πολωμένα φορτία ερωτημάτων, προσαρμόζεται σε ξαφνικές αλλαγές και ενημερώνει τα δεδομένα γρήγορα και αξιόπιστα.

Επιπλέον, μία ενδιαφέρουσα εφαρμογή περιγράφηκε, η οποία μπορεί να εκτελεστεί στο σύστημα που υλοποιεί τις μεθόδους για τη διαχείριση των ιεραρχικών δεδομένων. Ο στόχος προς επίτευξη είναι ο σχεδιασμός ενός πλήρως λειτουργικού, κατανεμημένου Πληροφοριακού Συστήματος για Υποδομές Πλέγματος. Σαν αποτέλεσμα, δημιουργήθηκε ένα σύστημα που παρουσιάζει πλεονεκτήματα σε σχέση με υπάρχουσες προσεγγίσεις. Η λειτουργία της προσαρμογής της δεικτοδότησης μπορεί να αξιοποιηθεί για την αυτόματη σύνοψη παλαιότερων αποτελεσμάτων και την αναλυτικότερη παρουσίαση των πιο πρόσφατων.

Έναν άλλο βασικό άξονα της παρούσας διατριβής αποτέλεσε ο σχεδιασμός και η υλοποίηση του συστήματος *LinkedPeers*: Το σύστημα αυτό αξιοποιεί τις μεθόδους που προαναφέρθηκαν για τη διαχείριση δεδομένων που περιγράφονται από πολλαπλές διαστάσεις, οι οποίες με τη σειρά τους δομούνται με εννοιολογικές ιεραρχίες. Κάθε αντικείμενο περιγράφεται από έναν αυθαίρετο αριθμό διαστάσεων, ενώ δεν απαιτείται ο προσδιορισμός τιμών για όλες τις διαστάσεις. Το *LinkedPeers* αποτελείται από πολλαπλά, "εικονικά" δακτυλίδια (ή πραγματικά με διαφορετικό χώρο αναγνωριστικών για το καθένα), όπου η κάθε διάσταση αντιστοιχίζεται σε ένα ξεχωριστό δακτυλίδι. Η ιδέα πίσω από το σχεδιασμό του *LinkedPeers* είναι η δημιουργία ενός αυτόνομου, "πρωτεύοντος" δακτυλιδιού για την αποθήκευση των δεδομένων, ενώ τα δευτερεύοντα δακτυλίδια συμβάλουν στην αποτελεσματική δημιουργία των απαιτούμενων δεικτών. Μία μηχανή για την υλοποίηση όψεων ανάλογα με τα ερωτήματα υλοποιεί μερικές όψεις για μελλοντική χρήση που αποτελούνται από τους προϋπολογισμένους συνδυασμούς τιμών που περιέχονται σε ένα ερώτημα που διενεργήθηκε, συμβάλλοντας ακόμα περισσότερο στην προσαρμοστική φύση του συστήματος. Επιπλέον, η δεικτοδότηση προσαρμόζεται δυναμικά στα εισερχόμενα ερωτήματα με αποτέλεσμα τη μείωση του κόστους επικοινωνίας κατά την κατανεμημένη επεξεργασία των ερωτημάτων. Η ικανότητα του συστήματος να προσαρμόζει την οργάνωση των δεδομένων του γίνεται ακόμα πιο σημαντική στην περίπτωση των πολυδιάστατων δεδομένων, καθώς δεν απαιτείται η δεικτοδότηση όλων των τιμών της κάθε διάστασης και των μεταξύ τους συνδυασμών; το γεγονός αυτό θα οδηγούσε σε εκθετική αύξηση του αποθηκευτικού χώρου όσο αυξάνει ο αριθμός των διαστάσεων. Εκτεταμένη πειραματική μελέτη του συστήματος δείχνει ότι το σύστημα συμπεριφέρεται με αποδοτικό τρόπο κατά την επίλυση της πλειοψηφίας των ερωτημάτων, μπορεί να χρησιμοποιηθεί για τη δεικτοδότηση τέτοιου είδους δεδομένων και προσαρμόζεται σε μη αναμενόμενες αλλαγές αποφασίζοντας σωστά να εκτελέσει διαδικασίες για προσαρμογή της δεικτοδότησης.

Οι μέθοδοι που προτάθηκαν στο *LinkedPeers* εξερευνώνται περαιτέρω και χρησιμοποιούνται στο σύστημα *PI4LD*, ένα σύστημα που ενοποιεί, αποθηκεύει, δεικτοδοτεί και επερωτά διασυνδεδεμένα δεδομένα. Η προτεινόμενη αρχιτεκτονική στοχεύει προς δύο κατευθύνσεις: είτε να αποτελέσει ένα σύστημα για την πλήρη διαχείριση τέτοιου είδους δεδομένων ή σαν ένα ενδιάμεσο επίπεδο δεικτοδότησης. Η προσαρμοστικότητα του συστήματος εξακολουθεί να διατηρείται σαν ένα από τα βασικά χαρακτηριστικά του και συμβάλλει στην επιτάχυνση του υπολογισμού των αποτελεσμάτων.

Τέλος, η επίλυση των ερωτημάτων εύρους τιμών σε ένα πιο γενικό πλαίσιο που βασίζεται σε DHT επικαλύψεις μπορεί να επιτευχθεί με την εκμετάλλευση των ιδιοτήτων που παρατηρούνται στις Καμπύλες Πλήρωσης του Χώρου. Χωρίς την αλλαγή της τοπολογίας που υπαγορεύεται από το DHT πρωτόκολλο που χρησιμοποιείται, μία συνάρτηση που στηρίζεται στις SFC μπορεί να χρησιμοποιηθεί για την αλλαγή της hash συνάρτησης του πρωτοκόλλου και να συνδυάσει πολλαπλές ιδιότητες σε ένα μόνο κλειδί που θα χρησιμοποιηθεί κατά την εισαγωγή των δεδομένων στην επικάλυψη. Μία τεχνική που βασίζεται στη διάταξη που προσφέρουν τα SFCs για την επίλυση απλών ερωτημάτων και ερωτημάτων εύρους τιμών προτείνεται και χρησιμοποιείται για το σχεδιασμό ενός κατανεμημένου κατάλογου μεταδεδομένων, που χρησιμοποιηθήκε για την αναζήτηση πολυμεσικού περιεχομένου. Η συγκεκριμένη τεχνική μπορεί να επεκταθεί περαιτέρω για την υποστήριξη ερωτημάτων εύρους σε ιεραρχικά δεδομένα παράλληλα με την υποστήριξη των συνοπτικών ερωτημάτων.

5.1 Μελλοντικές Επεκτάσεις

Σε γενικές γραμμές, πιθανές προεκτάσεις μπορούν να στοχεύσουν διάφορα ζητήματα σχετικά με τα προτεινόμενα συστήματα. Μερικά ζητήματα που μπορούν να ερευνηθούν περιγράφονται στη συνέχεια.

Το σύστημα *LinkedPeers* μπορεί να επεκταθεί περαιτέρω για να γίνει αποδοτικότερη η επίλυση άλλων τύπων ερωτημάτων, ειδικά αυτών που απαιτούν το συνδυασμό πληροφορίας που βρίσκεται διασκορπισμένη σε πολλούς κόμβους, όπως top-k ερωτήματα. Επομένως, μία πιθανή επέκταση είναι η ανάπτυξη πιο εξελιγμένων τεχνικών για κατηγορίες ερωτημάτων που δε μελετήθηκαν ήδη. Επίσης, μερικές συναρτήσεις συνάθροισης (π.χ. *SUM* ή *AVG*) μπορούν να υλοποιηθούν από την κατασκευή του συστήματος. Ο στόχος είναι να γίνει διευκολυνθεί ο υπολογισμός τέτοιων συναρτήσεων με αυτόματο τρόπο κατά την εισαγωγή των δεδομένων και τη δημιουργία των δεικτών, καθώς η ενημέρωση των αντικειμένων να γίνεται με τέτοιο τρόπο ώστε να ενημερώνονται κατάλληλα και οι αντίστοιχες υπολογισμένες τιμές. Επιπλέον, η πληροφορία που διατηρείται ήδη στο *LinkedPeers* μπορεί να αξιοποιηθεί για την εξαγωγή χρήσιμης πληροφορίας, για παράδειγμα η γνώση για τον αριθμό των συνδέσεων μπορεί να χρησιμοποιηθεί σαν

μία μετρική για την "ταξινόμηση" των κόμβων, ώστε να αποφευχθούν οι κόμβοι που συνεισφέρουν μικρό αριθμό αποτελεσμάτων κατά την προώθηση των ερωτημάτων. Γενικώς, ο καθορισμός μεθοδολογιών που βασίζονται στην κατηγοριοποίηση των πόρων μπορεί να βοηθήσει για την εκτίμηση των πόρων που είναι πιθανοί να επιστρέψουν αποτελέσματα για μία συγκεκριμένη ερώτηση. Η εκτίμηση αυτή μπορεί να οδηγήσει στον αποκλεισμό κόμβων που δεν αποθηκεύουν σχετικά δεδομένα και στον ορισμό μίας σειράς σύμφωνα με την οποία γίνεται η επίσκεψη των κόμβων, ώστε να επιτευχθούν καλύτεροι χρόνοι απόκρισης και μικρότερη επιβάρυνση για την απαιτούμενη επικοινωνία.

Η μελέτη "συναρτήσεων βαθμολόγησης" (*scoring functions*) για την αξιολόγηση των αντικειμένων που επιστρέφονται μετά την επεξεργασία ενός ερωτήματος φαίνεται πολλά υποσχόμενη για τα προτεινόμενα συστήματα. Η χρήση των αποτελεσμάτων αυτών των συναρτήσεων μπορεί να αποδειχθεί επικερδής για τη δρομολόγηση των ερωτημάτων και την ποιότητα των αποτελεσμάτων. Η υλοποίηση μεθόδων ταξινόμησης μπορεί να χρησιμοποιηθεί για αποτελεσματική top-k επεξεργασία, που κερδίζει το ενδιαφέρον σε πολλές εφαρμογές. Καθώς αυξάνει ο όγκος της πληροφορίας, οι μέθοδοι αναζήτησης πρέπει να λαμβάνουν υπόψη τους τις προτιμήσεις των χρηστών και να "προσωποποιούν" (personalize) τα επιστρεφόμενα αποτελέσματα. Από τη μία πλευρά, αποτελέσματα που είναι πολύ σημαντικά για ένα χρήστη μπορεί να μην είναι σχετικά για άλλον χρήστη, ο οποίος θέτει το ίδιο ερώτημα. Από την άλλη πλευρά, η παρουσίαση μεγάλου αριθμού αποτελεσμάτων που δεν ενδιαφέρουν το χρήστη μπορεί να αποκρίψει αυτά που είναι σημαντικά. Οι μέθοδοι αναζήτησης στο *LinkedPeers* και στο PI4LD μπορούν να χρησιμοποιήσουν την πληροφορία που παρέχεται από το χρήστη για να μειώσουν το χώρο αναζήτησης και να επιλεχθούν τα πιο σχετικά αποτελέσματα ανάλογα με τις προτιμήσεις των χρηστών. Τέτοιου είδους πληροφορία μπορεί να είναι οι τομείς που ενδιαφέρουν τους χρήστες, η χρονική περίοδος που δημοσιεύονται τα δεδομένα, το είδος των πόρων που δημοσιεύουν τα αποτελέσματα, κτλ.

Το *LinkedPeers* έχει τη δυνατότητα να χρησιμοποιηθεί για άλλους τύπους ημι-δομημένων δεδομένων. Δεδομένα που περιγράφονται με τη χρήση XML μπορούν να αποθηκευτούν στο *LinkedPeers* με τις αντίστοιχες τροποποιήσεις. Παρόλα αυτά, υπάρχουν θέματα που πρέπει να μελετηθούν σχετικά με XML εφαρμογές. Για παράδειγμα, ο τρόπος που θα κατανεμηθούν τα έγγραφα στους κόμβους είναι ένα πρόβλημα που χρειάζεται διερεύνηση. Επιπλέον, η πολυπλοκότητα της δομής των XML δεδομένων, του XPath και του XQuery δημιουργεί διάφορες προκλήσεις σχετικά με τη επίδοση. Μία κατεύθυνση που μπορεί να ακολουθηθεί για τη βελτίωση της απόδοσης της αξιολόγησης ενός XML ερωτήματος βασίζεται στη χρησιμοποίηση όψεων που έχουν υλοποιηθεί για να αποθηκεύουν προϋπολογισμένα αποτελέσματα. Οι όψεις αυξάνουν την ταχύτητα για την εύρεση των απαντήσεων σε σύγκριση με το χρόνο που χρειάζεται αν χρησιμοποιηθούν τα XML έγγραφα μόνο. Η υλοποίηση των όψεων παρέχεται από το *LinkedPeers* και χρειάζεται περαιτέρω προσαρμογή για την περίπτωση των XML δεδομένων.

Έχοντας υπόψη ότι τα προτεινόμενα συστήματα στοχεύουν διαδικτυακές εφαρμογές, η ετερογένεια των πόρων που δημοσιεύουν τα δεδομένα είναι ένα πολύ πιθανό ενδεχόμενο. Η ετερογένεια αυτή μπορεί να αποτελέσει αντικείμενο μεγαλύτερης προσοχής τόσο στο *LinkedPeers* όσο και στο PI4LD. Τα δύο αυτά συστήματα μπορούν να γίνουν πιο ευέλικτα και να αντιμετωπίσουν θέματα όπως το ταίριασμα και η αντιστοίχηση σχημάτων. Οι διασυνδέσεις ανάμεσα στις διαστάσεις που διατηρούνται στο *LinkedPeers* μπορούν να επεξεργαστούν για την εξαγωγή σημασιολογικής πληροφορίας και συσχετίσεων μεταξύ τιμών που δεν σχετίζονται άμεσα.

Μία άλλη παράμετρος που μπορεί να διερευνηθεί είναι πως η ικανότητα των SFCs να επιλύουν ερωτήματα εύρους τιμών μπορούν να ενσωματωθεί στο *LinkedPeers*. Αν και τα SFCs απαιτούν ένα σταθερό αριθμό διαστάσεων και δεν ευνοούν τα ερωτήματα που δεν περιλαμβάνουν μία συγκεκριμένη τιμή για κάθε διάσταση, μπορούν να μελετηθούν περαιτέρω ώστε η επίλυση ερωτημάτων εύρους τιμών να γίνει αποτελεσματικότερη στο *LinkedPeers*.

Όσον αφορά το PI4LD, η ενοποίηση δεδομένων που προέρχονται από ετερογενείς πόρους συνεχίζεται. Ιδιαίτερη έμφαση δίνεται σε μηχανισμούς σχεδιασμού της εκτέλεσης των ερωτημάτων και μελετώνται αντίστοιχες τεχνικές που εφαρμόζονται σε διασυνδεδεμένα αποθετήρια, λαμβάνοντας υπόψη ότι το προτεινόμενο σύστημα διατηρεί πληροφορία που μπορεί να χρησιμοποιηθεί αποδοτικότερα για την επίλυση μίας ερώτησης.

Publications

Journals

- *Distributing and searching concept hierarchies: an adaptive DHT-based system.* Athanasia Asiki, Dimitrios Tsoumakos and Nectarios Koziris. Cluster Computing, Volume 13, Issue 3, 2010, Pages 257-276, ISSN 1386-7857, http://dx.doi.org/10.1007/s10586-010-0136-5.

- *Replica-aware, multi-dimensional range queries in Distributed Hash Tables.* Antony Chazapis, Athanasia Asiki, Georgios Tsoukalas, Dimitrios Tsoumakos and Nectarios Koziris. Computer Communications, Volume 33, Issue 8, 17 May 2010, Pages 984-996, ISSN 0140-3664, http://dx.doi.org/10.1016/j.comcom.2010.01.024.

- *A grid middleware for data management exploiting peer-to-peer techniques.* Athanasia Asiki, Katerina Doka, Ioannis Konstantinou, Antonis Zissimos, Dimitrios Tsoumakos, Nectarios Koziris and Panayiotis Tsanakas. Future Generation Computer Systems, Volume 25, Issue 4, April 2009, Pages 426-435, ISSN 0167-739X, http://dx.doi.org/10.1016/j.future.2008.09.005.

Conferences

- *LinkedPeers: A Distributed System for Interlinking Multidimensional Data.* Athanasia Asiki, Dimitrios Tsoumakos and Nectarios Koziris. In Proceedings of the 22nd International Conference on Database and Expert Systems Applications (DEXA 2011), 29 Aug.-2 Sept. 2011, Lecture Notes in Computer Science, Springer, Pages 527-543, http://dx.doi.org/10.1007/978 − 3 − 642 − 23091 − 2_47.

- *An adaptive online system for efficient processing of hierarchical data.* Athanasia Asiki, Dimitrios Tsoumakos and Nectarios Koziris. In Proceedings of the 18th ACM International Symposium on High Performance Distributed Computing (HPDC 2009), Garching, Germany, June 11-13, 2009, ACM, Pages 71 - 80, http://doi.acm.org/10.1145/1551609.1551627.

- *Online Querying of Concept Hierarchies in P2P Systems.* Katerina Doka, Athanasia Asiki, Dimitrios Tsoumakos and Nectarios Koziris. In Proceedings of the On the Move to Meaningful Internet Systems: OTM 2008, Springer, Volume 5331, Pages 212-230, http://dx.doi.org/10.1007/978 − 3 − 540 − 88871 − 0_16.

- *Support for Concept Hierarchies in DHTs.* Athanasia Asiki, Katerina Doka, Dimitrios Tsoumakos and Nectarios Koziris. In Proceedings of the Eighth International Conference on Peer-to-Peer Computing (P2P'08), Aachen, Germany, 8-11 September 2008, IEEE Computer Society, Pages 121-124, http://doi.ieeecomputersociety.org/10.1109/P2P.2008.26.

- *A Distributed Architecture for Multi-Dimensional Indexing and Data Retrieval in Grid Environments.* Athanasia Asiki, Katerina Doka, Ioannis Konstantinou, Antonis Zissimos and Nectarios Koziris. In Proceedings of the Cracow 2007 Grid Workshop (CGW'07), Krakow, Polland, October 16-17, 2007.

- *Gredia Middleware Architecture* Ioannis Konstantinou, Katerina Doka, Athanasia Asiki, Antonis Zissimos and Nectarios Koziris. In Proceedings of the Cracow 2007 Grid Workshop (CGW'07), Krakow, Polland, October 16-17, 2007

Bibliography

[AAB+08] Rakesh Agrawal, Anastasia Ailamaki, Philip A. Bernstein, Eric A. Brewer, Michael J. Carey, Surajit Chaudhuri, AnHai Doan, Daniela Florescu, Michael J. Franklin, Hector Garcia-Molina, Johannes Gehrke, Le Gruenwald, Laura M. Haas, Alon Y. Halevy, Joseph M. Hellerstein, Yannis E. Ioannidis, Hank F. Korth, Donald Kossmann, Samuel Madden, Roger Magoulas, Beng Chin Ooi, Tim O'Reilly, Raghu Ramakrishnan, Sunita Sarawagi, Michael Stonebraker, Alexander S. Szalay, and Gerhard Weikum. The claremont report on database research. *SIGMOD Rec.*, 37:9–19, September 2008.

[AAG+05] Karl Aberer, Luc Onana Alima, Ali Ghodsi, Sarunas Girdzijauskas, Seif Haridi, and Manfred Hauswirth. The essence of p2p: A reference architecture for overlay networks. In *In P2P2005, The 5th IEEE International Conference on Peer-to-Peer Computing*, pages 11–20, 2005.

[Abe01] Karl Aberer. P-grid: A self-organizing access structure for p2p information systems. In *CoopIS*, pages 179–194, 2001.

[ACMHP04] Karl Aberer, Philippe Cudré-Mauroux, Manfred Hauswirth, and Tim Van Pelt. Gridvine: Building internet-scale semantic overlay networks. In *International Semantic Web Conference*, pages 107–121, 2004.

[AHL+11] James Ahrens, Bruce Hendrickson, Gabrielle Long, Steve Miller, Rob Ross, and Dean Williams. Data-intensive science in the us doe: Case studies and future challenges. *Computing in Science and Engineering*, 13:14–24, 2011.

[AMMH07] Daniel J. Abadi, Adam Marcus, Samuel R. Madden, and Kate Hollenbach. Scalable semantic web data management using vertical partitioning. In *Proceedings of the 33rd international conference on Very large data bases*, VLDB '07, pages 411–422. VLDB Endowment, 2007.

[apb] apb. *OLAP Council APB-1 OLAP Benchmark*. http://www.olapcouncil.org/research/resrchly.htm.

[ATS04] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Comput. Surv.*, 36:335–371, December 2004.

[AX02] Artur Andrzejak and Zhichen Xu. Scalable, efficient range queries for grid information services. In *Peer-to-Peer Computing*, pages 33–40, 2002.

[BAS04] Ashwin R. Bharambe, Mukesh Agrawal, and Srinivasan Seshan. Mercury: supporting scalable multi-attribute range queries. *SIGCOMM Computer Communication Review*, 34(4):353–366, 2004.

[Bay97] Rudolf Bayer. The universal B-tree for multidimensional indexing: General concepts. *Lecture Notes in Computer Science*, 1274/1997:198–209, 1997.

[BGS06] Gordon Bell, Jim Gray, and Alex Szalay. Petascale computational systems. *Computer*, 39:110–112, 2006.

[BHBL09] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked Data - The Story So Far. *Int. Journal on Semantic Web and Information Systems (IJSWIS)*, 2009.

[Bit] Bittorent. Bittorent website:. http://www.bittorrent.com/.

[BKK+03] Hari Balakrishnan, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Looking up data in p2p systems. *Commun. ACM*, 46:43–48, February 2003.

[BLK+09] Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. Dbpedia - a crystallization point for the web of data. *Web Semant.*, 7:154–165, September 2009.

[BR99] Kevin Beyer and Raghu Ramakrishnan. Bottom-up computation of sparse and iceberg cube. *SIGMOD Rec.*, 28:359–370, June 1999.

[Bun97] Peter Buneman. Semistructured data. In *PODS*, pages 117–121, 1997.

[But71] A. R. Butz. Alternative algorithm for hilbert's space-filling curve. *IEEE Trans. Comput.*, 20:424–426, April 1971.

[CBB11] Stefano Ceri, Alessandro Bozzon, and Marco Brambilla. The anatomy of a multi-domain search infrastructure. In *Proceedings of the 11th international conference on Web engineering*, ICWE'11, pages 1–12, Berlin, Heidelberg, 2011. Springer-Verlag.

[CDT07] Eddy Caron, Frédéric Desprez, and Cédric Tedeschi. Enhancing computational grids with peer-to-peer technology for large scale service discovery. *Journal of Grid Computing*, 5:337–360, 2007. 10.1007/s10723-006-9058-0.

[CF04] Min Cai and Martin Frank. Rdfpeers: a scalable distributed rdf repository based on a structured peer-to-peer network. In *Proceedings of the 13th international conference on World Wide Web*, WWW '04, pages 650–657, New York, NY, USA, 2004. ACM.

[CFCS04] Min Cai, Martin Frank, Jinbo Chen, and Pedro Szekely. Maan: A multi-attribute addressable network for grid information services. *Journal of Grid Computing*, 2:3–14, 2004. 10.1007/s10723-004-1184-y.

[CFK+00] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke. The data grid: Towards an architecture for the distributed management and analysis of large scale scientific datasets. *Journal of Network and Computer Applications*, 23(3):187–200, 2000.

[CLGS04] Adina Crainiceanu, Prakash Linga, Johannes Gehrke, and Jayavel Shanmugasundaram. Querying peer-to-peer networks using p-trees. In *Proceedings of the 7th International Workshop on the Web and Databases: colocated with ACM SIGMOD/PODS 2004*, WebDB '04, pages 25–30, New York, NY, USA, 2004. ACM.

[Data] Linked Data. Connect Distributed Data across the Web. http://linkeddata.org/.

[Datb] W3C Linked Data. What is Linked Data? http://www.w3.org/standards/semanticweb/data.html.

[DBpa] Querying DBpedia. Public sparql endpoint. http://wiki.dbpedia.org/OnlineAccess.

[DBpb] DBpediaOntology. Dbpedia ontology. http://mappings.dbpedia.org/server/ontology/classes.

[DHJ+05] Anwitaman Datta, Manfred Hauswirth, Renault John, Roman Schmidt, and Karl Aberer. Range queries in trie-structured overlays. In *Proceedings of the Fifth IEEE International Conference on Peer-to-Peer Computing*, pages 57–66, Washington, DC, USA, 2005. IEEE Computer Society.

[ege] egee. Egee accounting portal. http://www3.egee.cesga.es/gridsite/accounting/CESGA/.

[eMu] eMule. emule website:. http://www.emule-project.net/.

[et.05] R. Byrom et.al. Apel: An implementation of grid accounting using r-gma. In *UK e-Science All Hands Conference*, 2005.

[fac] facebook. facebook:. http://www.facebook.com/.

[FK99] Ian Foster and Carl Kesselman, editors. *The grid: Blueprint for a new computing infrastructure*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.

[FK03] Ian Foster and Carl Kesselman. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[FKT01] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *Int. J. High Perform. Comput. Appl.*, 15:200–222, August 2001.

[Fre] FreePastry. http://freepastry.rice.edu/FreePastry.

[GAA03] Abhishek Gupta, Divyakant Agrawal, and Amr El Abbadi. Approximate range selection queries in peer-to-peer systems. In *CIDR*, 2003.

[Gan] Ganglia. Ganglia Monitoring System. http://ganglia.info/.

[GCB+97] Jim Gray, Surajit Chaudhuri, Adam Bosworth, Andrew Layman, Don Reichart, Murali Venkatrao, Frank Pellow, and Hamid Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Mining and Knowledge Discovery*, 1:29–53, 1997. 10.1023/A:1009726021843.

[GL09] Al Geist and Robert Lucas. Major computer science challenges at exascale. *Int. J. High Perform. Comput. Appl.*, 23:427–436, November 2009.

[glo] globus. The Globus Toolkit. http://www.globus.org/.

[Gnu] Gnutella2. Gnutella website:. g2.trillinux.org//.

[Goo] Google. Google search engine. www.google.com.

[GPH04] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. An evaluation of knowledge base systems for large owl datasets. In *International Semantic Web Conference*, pages 274–288, 2004.

[GR11] J. Gantz and D. Reinsel. Extracting value from chaos, June 2011.

[GRC+07] J.F. Gantz, D. Reinsel, C. Chute, W. Schlichting, J. McArthur, S. Minton, I. Xheneti, A. Toncheva, and A. Manfrediz. A forecast of worldwide information growth thorough 2012, March 2007.

[GRE] GREDIA. Grid Enabled access to rich media content (GREDIA) IST project. IST 6th Framework Programme (FP6-34363). Available online: http://www.gredia.eu.

[GS11] Olaf Görlitz and Steffen Staab. Federated data management and query optimization for linked open data, 2011.

[GYGM04] Prasanna Ganesan, Beverly Yang, and Hector Garcia-Molina. One torus to rule them all: multi-dimensional queries in p2p systems. In *Proceedings of the 7th International Workshop on the Web and Databases: colocated with ACM SIGMOD-/PODS 2004*, WebDB '04, pages 19–24, New York, NY, USA, 2004. ACM.

[Hal09] Harry Halpin. A query-driven characterization of linked data. In *LDOW*, 2009.

[Haw] Hawkeye. Hawkeye: A Monitoring and Management Tool for Distributed Systems. http://www.cs.wisc.edu/condor/hawkeye/.

[HCH+05] Ryan Huebsch, Brent Chun, Joseph M. Hellerstein, Boon Thau Loo, Petros Maniatis, Timothy Roscoe, Scott Shenker, Ion Stoica, and Aydan R. Yumerefendi. The architecture of pier: an internet-scale query processor. In *In CIDR*, pages 28–43, 2005.

[HG03] S. Harris and N. Gibbins. 3store: Efficient bulk RDF storage. In *Proceedings of the 1st International Workshop on Practical and Scalable Semantic Systems (PSSS'03)*, pages 1–20. Citeseer, 2003.

[HHB+03] R. Huebsch, J. Hellerstein, N. L. Boon, T. Loo, S. Shenker, and I. Stoica. Querying the Internet with PIER. In *VLDB*, 2003.

[HHK+10] Andreas Harth, Katja Hose, Marcel Karnstedt, Axel Polleres, Kai-Uwe Sattler, and Jürgen Umbrich. Data summaries for on-demand queries over linked data. In

Proceedings of the 19th international conference on World wide web, WWW '10, pages 411–420, New York, NY, USA, 2010. ACM.

[HJS+03] Nicholas J. A. Harvey, Michael B. Jones, Stefan Saroiu, Marvin Theimer, and Alec Wolman. Skipnet: a scalable overlay network with practical locality properties. In *Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems - Volume 4*, USITS'03, pages 9–9, Berkeley, CA, USA, 2003. USENIX Association.

[HK00] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., 2000.

[HMZ10] Peter Haase, Tobias Mathäß, and Michael Ziller. An evaluation of approaches to federated query processing over linked data. In *Proceedings of the 6th International Conference on Semantic Systems*, I-SEMANTICS '10, pages 5:1–5:9, New York, NY, USA, 2010. ACM.

[HSTW11] Katja Hose, Ralf Schenkel, Martin Theobald, and Gerhard Weikum. Database foundations for scalable rdf processing. In Axel Polleres, Claudia d'Amato, Marcelo Arenas, Siegfried Handschuh, Paula Kroner, Sascha Ossowski, and Peter F. Patel-Schneider, editors, *Reasoning Web*, volume 6848 of *Lecture Notes in Computer Science*, pages 202–249. Springer, 2011.

[iMe] iMesh. imesh website:. http://www.imesh.com/.

[Jen] JenaProvider. Virtuoso jena provider. http://www.openlinksw.com/dataspace/dav/wiki/Main/VirtJenaProvider.

[JHS+10] Prateek Jain, Pascal Hitzler, Amit Sheth, Kunal Verma, and Peter Z. Yeh. Ontology alignment for linked open data. In *9th International Semantic Web Conference (ISWC2010)*, November 2010.

[JOV05] H. V. Jagadish, Beng Chin Ooi, and Quang Hieu Vu. Baton: a balanced tree structure for peer-to-peer networks. In *Proceedings of the 31st international conference on Very large data bases*, VLDB '05, pages 661–672. VLDB Endowment, 2005.

[KaZ] KaZaA. Kazaa website:. http://www.kazaa.com/.

[KKK+10] Zoi Kaoudi, Manolis Koubarakis, Kostis Kyzirakos, Iris Miliaraki, Matoula Magiridou, and Antonios Papadakis-Pesaresi. Atlas: Storing, updating and querying rdf(s) data on top of dhts. *Web Semant.*, 8:271–277, November 2010.

[KLL+97] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, STOC '97, pages 654–663, New York, NY, USA, 1997. ACM.

[KP04] Georgia Koloniari and Evaggelia Pitoura. Content-based routing of path queries in peer-to-peer systems. In *EDBT*, pages 29–47, 2004.

[KSHS08] Marcel Karnstedt, Kai-Uwe Sattler, Manfred Hauswirth, and Roman Schmidt. A dht-based infrastructure for ad-hoc integration and querying of semantic data. In *Proceedings of the 2008 international symposium on Database engineering & applications*, IDEAS '08, pages 19–28, New York, NY, USA, 2008. ACM.

[KXZ05] A. Kumar, J. Xu, and E.W. Zegura. Efficient and scalable query routing for unstructured peer-to-peer networks. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 2, pages 1162 – 1173 vol. 2, March 2005.

[LCC+02] Qin Lv, Pei Cao, Edith Cohen, Kai Li, and Scott Shenker. Search and replication in unstructured peer-to-peer networks. In *Proceedings of the 16th international conference on Supercomputing*, ICS '02, pages 84–95, New York, NY, USA, 2002. ACM.

[LCLC09] Dongsheng Li, Jiannong Cao, Xicheng Lu, and Keith C.C. Chen. Efficient Range Query Processing in Peer-to-Peer Systems. *IEEE Transactions on Knowledge and Data Engineering*, 21(1):78–91, Jan. 2009.

[LCP+05] Eng Keong Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim. A survey and comparison of peer-to-peer overlay network schemes. *Communications Surveys Tutorials, IEEE*, 7(2):72 – 93, quarter 2005.

[LK00] J.K. Lawder and P.J.H. King. Using Space-Filling Curves for Multi-dimensional Indexing. *Lecture Notes in Computer Science*, 1832/2000:20–35, 2000.

[LLK+07] Jinwon Lee, Hyonik Lee, Seungwoo Kang, Su Myeon Kim, and Junehwa Song. CISS: An efficient object clustering framework for DHT-based peer-to-peer applications. *Computer Networks*, 51(4):1072–1094, 2007.

[LPZ03] Laks V. S. Lakshmanan, Jian Pei, and Yan Zhao. Qc-trees: an efficient summary structure for semantic olap. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, SIGMOD '03, pages 64–75, New York, NY, USA, 2003. ACM.

[MAK03] Mohamed F. Mokbel, Walid G. Aref, and Ibrahim Kamel. Analysis of Multi-Dimensional Space-Filling Curves. *Geoinformatica*, 7(3):179–209, 2003.

[MAYU05] Akiyoshi Matono, Toshiyuki Amagasa, Masatoshi Yoshikawa, and Shunsuke Uemura. A path-based relational rdf database. In *Proceedings of the 16th Australasian database conference - Volume 39*, ADC '05, pages 95–103, Darlinghurst, Australia, Australia, 2005. Australian Computer Society, Inc.

[MCB+11] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and Byers A. H. Big data: The next frontier for innovation, competition and productivity, May 2011.

[MDS] MDS. GT Information Services: Monitoring and Discovery System (MDS). http://www.globus.org/toolkit/mds/.

[MI06] Konstantinos Morfonios and Yannis Ioannidis. Cure for cubes: cubing using a rolap engine. In *Proceedings of the 32nd international conference on Very large data bases*, VLDB '06, pages 379–390. VLDB Endowment, 2006.

[MJFS01] B. Moon, H.V. Jagadish, C. Faloutsos, and J.H. Saltz. Analysis of the clustering properties of the Hilbert space-filling curve. *IEEE Transactions on Knowledge and Data Engineering*, 13(1):124–141, Jan/Feb 2001.

[MM02] P. Maymounkov and D. Mazières. Kademlia: A peer-to-peer information system based on the XOR metric. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS02)*, pages 53–65, Cambridge, USA, March 2002.

[MPE7] MPEG. MPEG-7 Overview, 7. Available online: http://www.chiariglione.org/mpeg/standards/mpeg-7/mpeg-7.htm.

[NW10] Thomas Neumann and Gerhard Weikum. The rdf-3x engine for scalable management of rdf data. *The VLDB Journal*, 19:91–113, February 2010.

[OTZ+03] Beng Chin Ooi, Kian-Lee Tan, Aoying Zhou, Chin Hong Goh, Yingguang Li, Chu Yee Liau, Bo Ling, Wee Siong Ng, Yanfeng Shu, Xiaoyu Wang, and Ming Zhang. Peerdb: Peering into personal databases. In *SIGMOD Conference*, page 659, 2003.

[OV11] M.T. Ozsu and P. Valduriez. *Principles of Distributed Database Systems*. Springer, 2011.

[QL08] Bastian Quilitz and Ulf Leser. Querying distributed rdf data sources with sparql. In *Proceedings of the 5th European semantic web conference on The semantic web: research and applications*, ESWC'08, pages 524–538, Berlin, Heidelberg, 2008. Springer-Verlag.

[RD01] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, Middleware '01, pages 329–350, London, UK, 2001. Springer-Verlag.

[RDF] RDF. Resource Description Framework(RDF). http://www.w3.org/RDF/.

[RFH+01] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 161–172, San Diego, USA, August 2001.

[RGM] RGMA. R-GMA: Relational Grid Monitoring Architecture. http://www.r-gma.org/.

[RM06] John Risson and Tim Moors. Survey of research towards robust peer-to-peer networks: search methods. *Comput. Netw.*, 50:3485–3521, December 2006.

[RRHS04] Sriram Ramabhadran, Sylvia Ratnasamy, Joseph M. Hellerstein, and Scott Shenker. Brief announcement: Prefix Hash Tree. In *Proceedings of the twenty-third Annual ACM symposium on Principles of distributed computing (PODC '04)*, pages 368–368, New York, NY, USA, 2004.

[SC11] Michael Stonebraker and Rick Cattell. 10 rules for scalable performance in 'simple operation' datastores. *Commun. ACM*, 54:72–80, June 2011.

[SDKR03] Yannis Sismanis, Antonios Deligiannakis, Yannis Kotidis, and Nick Roussopoulos. Hierarchical dwarfs for the rollup cube. In *Proceedings of the 6th ACM international workshop on Data warehousing and OLAP*, DOLAP '03, pages 17–24, New York, NY, USA, 2003. ACM.

[SDRK02] Yannis Sismanis, Antonios Deligiannakis, Nick Roussopoulos, and Yannis Kotidis. Dwarf: shrinking the petacube. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, SIGMOD '02, pages 464–475, New York, NY, USA, 2002. ACM.

[Sem] SemWeb. Semantic web. http://www.w3.org/standards/semanticweb/.

[SGG03] Stefan Saroiu, Krishna P. Gummadi, and Steven D. Gribble. Measuring and analyzing the characteristics of napster and gnutella hosts. *Multimedia Systems*, 9:170–184, 2003. 10.1007/s00530-003-0088-1.

[SGH+11] Michael Schmidt, Olaf Görlitz, Peter Haase, Günter Ladwig, Andreas Schwarte, and Thanh Tran. Fedbench: A benchmark suite for federated semantic data query processing. In *International Semantic Web Conference (1)*, pages 585–600, 2011.

[SHH+11] Andreas Schwarte, Peter Haase, Katja Hose, Ralf Schenkel, and Michael Schmidt. Fedx: optimization techniques for federated query processing on linked data. In *Proceedings of the 10th international conference on The semantic web - Volume Part I*, ISWC'11, pages 601–616, Berlin, Heidelberg, 2011. Springer-Verlag.

[SKW07] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web*, WWW '07, pages 697–706, New York, NY, USA, 2007. ACM.

[Sky] Skype. Skype website:. http://www.skype.com/.

[SMK+01] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, San Diego, USA, August 2001.

[SOTZ05] Yanfeng Shu, Beng Chin Ooi, Kian-Lee Tan, and Aoying Zhou. Supporting multi-dimensional range queries in peer-to-peer systems. In *Proceedings of the Fifth IEEE International Conference on Peer-to-Peer Computing*, pages 173–180, Washington, DC, USA, 2005. IEEE Computer Society.

[SP04] Cristina Schmidt and Manish Parashar. Enabling flexible queries with guarantees in p2p systems. *IEEE Internet Computing*, 8:19–26, May 2004.

[SPA] SPARQL. SPARQL Query Language for RDF. http://www.w3.org/TR/rdf-sparql-query/.

[SQL] SQLite. http://www.sqlite.org/.

[TH81] H. Tropf and H. Herzog. Multidimensional Range Search in Dynamically Balanced Trees. *Applied Informatics*, 2/1981:71–77, 1981.

[TH04] Igor Tatarinov and Alon Halevy. Efficient query reformulation in peer data management systems. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, SIGMOD '04, pages 539–550, New York, NY, USA, 2004. ACM.

[THS07] Egemen Tanin, Aaron Harwood, and Hanan Samet. Using a distributed quadtree index in peer-to-peer networks. *The VLDB Journal*, 16:165–178, April 2007.

[TR06] Dimitrios Tsoumakos and Nick Roussopoulos. Analysis and comparison of p2p search methods. In *Proceedings of the 1st international conference on Scalable information systems*, InfoScale '06, New York, NY, USA, 2006. ACM.

[TS06] Andrew S. Tanenbaum and Maarten van Steen. *Distributed Systems: Principles and Paradigms (2nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006.

[Twi] Twitter. twitter. http://twitter.com/.

[TXD03] Chunqiang Tang, Zhichen Xu, and Sandhya Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overlay networks. In *SIGCOMM*, pages 175–186, 2003.

[Vir] Open-Source Edition Virtuoso. Version 6.1. http://www.openlinksw.com/wiki/main/Main.

[WJOT09] Sai Wu, Shouxu Jiang, Beng Chin Ooi, and Kian-Lee Tan. Distributed online aggregations. *Proc. VLDB Endow.*, 2:443–454, August 2009.

[WKB08] Cathrin Weiss, Panagiotis Karras, and Abraham Bernstein. Hexastore: sextuple indexing for semantic web data management. *Proc. VLDB Endow.*, 1:1008–1019, August 2008.

[WLC] WLCG. Worldwide lhc computing grid. http://lcg.web.cern.ch/LCG/public/default.htm.

[WLOT08] Sai Wu, Jianzhong Li, Beng Chin Ooi, and Kian-Lee Tan. Just-in-time query retrieval over partially indexed data on structured p2p overlays. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 279–290, New York, NY, USA, 2008. ACM.

[ZDN97] Yihong Zhao, Prasad M. Deshpande, and Jeffrey F. Naughton. An array-based algorithm for simultaneous multidimensional aggregates. In *Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, SIGMOD '97, pages 159–170, New York, NY, USA, 1997. ACM.

[ZFS07] X. Zhang, J. Freschl, and J. Schopf. Scalability analysis of three monitoring and information systems: MDS2, R-GMA, and Hawkeye. *J. Parallel Distrib. Comput.*, 67(8):883–902, 2007.

[ZHDR09] Jing Zhou, Wendy Hall, and David De Roure. Building a distributed infrastructure for scalable triple stores. *Journal of Computer Science and Technology*, 24:447–462, 2009. 10.1007/s11390-009-9236-1.

[ZHS+04] B.Y. Zhao, Ling Huang, J. Stribling, S.C. Rhea, A.D. Joseph, and J.D. Kubiatowicz. Tapestry: a resilient global-scale overlay for service deployment. *Selected Areas in Communications, IEEE Journal on*, 22(1):41–53, Jan. 2004.

[ZSL+05] Hai Zhuge, Xiaoping Sun, Jie Liu, Erlin Yao, and Xue Chen. A scalable p2p platform for the knowledge grid. *IEEE Trans. on Knowl. and Data Eng.*, 17:1721–1736, December 2005.