ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Ανωνυμοποίηση σχεσιακών δεδομένων σε κατανεμημένα περιβάλλοντα

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ιωάννης Κ. Γιαννακόπουλος

Επιβλέπων: Κοζύρης Νεκτάριος
Αναπληρωτής Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2012

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Ανωνυμοποίηση σχεσιακών δεδομένων σε κατανεμημένα περιβάλλοντα

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ιωάννης Κ. Γιαννακόπουλος

Επιβλέπων: Κοζύρης Νεκτάριος
Αναπληρωτής Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 22η Οκτωβρίου 2012:

.....	.....	.....
Κοζύρης Νεκτάριος Αν. Καθηγητής Ε.Μ.Π.	Σελλής Τιμολέων Καθηγητής Ε.Μ.Π.	Τσουμάκος Δημήτριος Επίκουρος Καθηγητής Πανεπιστημίου Ιονίου

Αθήνα, Οκτώβριος 2012

.....
Ιωάννης Κ. Γιαννακόπουλος
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών

ΠΕΡΙΛΗΨΗ

Στην παρούσα διπλωματική εργασία εξετάζουμε το πρόβλημα της ανωνυμοποίησης σχεσιακών δεδομένων με χρήση κατανεμημένων τεχνικών. Η ανωνυμοποίηση δεδομένων αποκτά ολοένα και μεγαλύτερη σημασία στις μέρες μας, εξαιτίας της έκρηξης δεδομένων που έχει σημειωθεί τα τελευταία χρόνια και συνεχίζεται ακόμη και σήμερα. Η πρόσβαση σε μεγάλο όγκο δεδομένων που συχνά παράγονται από πολλές διαφορετικές πηγές αν και είναι επιθυμητή, μπορεί να εγείρει σημαντικά ζητήματα για την προστασία και τη διατήρηση της ανωνυμίας των ατόμων και της ιδιωτικότητας των πληροφοριών που τα αφορούν. Η αφαίρεση προσωπικών πληροφοριών από τα δεδομένα (όπως το Όνομα ή το ΑΦΜ) δεν εγγυάται τη διατήρηση της ανωνυμίας, αφού ο συνδυασμός των εναπομεινάντων γνωρισμάτων με εξωτερικά, δημοσίως διαθέσιμα δεδομένα μπορεί να οδηγήσει τελικά στην ταυτοποίηση των ατόμων.

Για την αντιμετώπιση αυτών των κινδύνων έχει προταθεί το μοντέλο k-anonymity. Σκοπός του μοντέλου είναι η γενίκευση των δεδομένων με κατάλληλο τρόπο έτσι ώστε κάθε συνδυασμός των χαρακτηριστικών εκείνων, που αν διασταυρωθούν με εξωτερικές πηγές μπορούν να οδηγήσουν στην ταυτοποίηση του ατόμου, να εμφανίζονται στα δεδομένα τουλάχιστον k φορές. Έχει αναπτυχθεί ένας μεγάλος αριθμός αλγορίθμων που έχουν στόχο την εφαρμογή του μοντέλου σε σχεσιακά δεδομένα. Στην παρούσα εργασία θα συγκρίνουμε δυο αλγορίθμους ανωνυμοποίησης που εκτελούν local recoding και στηρίζονται στη συνεχή διαμέριση των δεδομένων σε υποομάδες.

Παράλληλα, ο συνεχώς αυξανόμενος όγκος των δεδομένων καθιστά αναγκαία την χρησιμοποίηση κατανεμημένων τεχνικών για τη γρήγορη και αποδοτική ανωνυμοποίηση των πληροφοριών. Η κατανεμημένη εκτέλεση θα συμβάλλει στην ταχύτερη ολοκλήρωση της διαδικασίας, στη διαχείριση πολύ μεγάλου όγκου δεδομένων που δεν θα ήταν δυνατή από μια κεντρική εκτέλεση, στην παραλληλοποίηση της διαδικασία και στην διαχείριση των σφαλμάτων με πολύ μικρότερο κόστος σε σχέση με μια κεντρική εκτέλεση.

Στην παρούσα εργασία προτείνουμε ένα τρόπο παραλληλοποίησης του προβλήματος, έτσι ώστε να είναι εφικτή η κατανεμημένη εκτέλεση και στη συνέχεια εκτελούμε τους αλγορίθμους ανωνυμοποίησης συγκρίνοντας την απόδοσή τους για διάφορες περιπτώσεις δεδομένων και χαρακτηριστικών της ανωνυμοποίησης.

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ

Ανωνυμοποίηση, Κατανεμημένα συστήματα, k-anonymity, Mondrian, TopDown

ABSTRACT

In this diploma thesis we consider the problem of anonymising relational data using distributed techniques. The data anonymisation becomes more and more important nowadays due to the explosion of data that has been made in recent years and continues even today. Access to large amounts of data, which are often produced from many different sources, although desirable, may raise important issues about protecting and preserving the anonymity of individuals and privacy of information concerning them. The removal of personal information from data (such as Name or Social Security number) does not guarantee preservation of anonymity because the combination of the remaining attributes with external, publicly available data may eventually lead to the identification of individuals.

To address these risks, the model k-anonymity was proposed. The purpose of the model is the generalization of data in an appropriate manner so that any combination of the characteristics of those which, if crossed by external sources can lead to the identification of the individual, to appear in data at least k times. There have been developed a large number of algorithms that aim to apply the model to relational data. In this thesis we will compare two anonymisation algorithms that perform local recoding and they are based on continuous partitioning data into subgroups.

Meanwhile, the ever-increasing size of data requires the use of distributed techniques for fast and efficient anonymisation of information. The distributed implementation will contribute to faster completion of the procedure, to managing very large volumes of data that would not be managed by a centralized execution, to the parallelization of process and the management of error at a much lower cost than a centralized execution.

In this diploma thesis, we propose a way of parallelizing the problem, so as to enable the distributed execution and then execute anonymisation algorithms, comparing their performance for different situations and characteristics of data anonymisation.

KEY WORDS

anonymization, distributed systems, k-anonymity, Mondrian, TopDown

Ευχαριστίες

Αρχικά θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή, κύριο Νεκτάριο Κοζύρη, αναπληρωτή Καθηγητή ΕΜΠ για την εμπιστοσύνη που μου έδειξε και την δυνατότητα που μου έδωσε να ασχοληθώ με ένα τόσο πρωτοποριακό και ενδιαφέρον θέμα στα πλαίσια της διπλωματικής μου εργασίας.

Επίσης, θα ήθελα να ευχαριστήσω την μεταδιδακτορική ερευνήτρια του εργαστηρίου Υπολογιστικών Συστημάτων, κυρία Κατερίνα Δόκα για την καθοδήγησή της, την πολύτιμη βοήθειά της και την καθοριστική συμβολή της στην εξέλιξη και στην ολοκλήρωση της παρούσας εργασίας.

Τέλος, θα ήθελα να ευχαριστήσω την οικογένειά μου και τους φίλους μου που στάθηκαν δίπλα μου, υλικά και πνευματικά, σε αυτήν την πολυετή πορεία μου στο Εθνικό Μετσόβιο Πολυτεχνείο.

Σας ευχαριστώ.

Περιεχόμενα

Κατάλογος σχημάτων

Κατάλογος πινάκων

Κεφάλαιο 1

Εισαγωγή

1.1 Περιγραφή του προβλήματος

Τα τελευταία χρόνια παρατηρείται μια έκρηξη δεδομένων που συνεχίζεται ακόμη και σήμερα. Καθημερινά χιλιάδες εταιρείες, οργανισμοί, ακαδημαϊκά και ερευνητικά ιδρύματα παράγουν πολύ μεγάλα ποσά δεδομένων και στηρίζουν την επιβίωσή τους στην ανάλυση των δεδομένων αυτών, με στόχο την ανακάλυψη συγκεκριμένων τάσεων των χρηστών, την ανίχνευση ορισμένων μοτίβων συμπεριφοράς ή ακόμα και την παροχή πιο προσωποποιημένων υπηρεσιών στους χρήστες. Για την ανάλυση τόσο μεγάλου όγκου δεδομένων, έχουν προταθεί και χρησιμοποιούνται κατανεμημένα συστήματα που προσφέρουν την δυνατότητα της μεγάλης κλιμακωσιμότητας (scalability) και της ανοχής σε σφάλματα με χαμηλό κόστος (fault tolerance).

Όπως γίνεται σαφές, οι χρήστες αποκτούν καθημερινά πρόσβαση σε ένα ολοένα αυξανόμενο όγκο πληροφορίας. Αν και αυτό το γεγονός είναι συνήθως επιθυμητό, μπορεί να εγείρει σημαντικά ζητήματα στην προστασία της ανωνυμίας των ατόμων καθώς και της ιδιωτικότητας των πληροφοριών που τα αφορούν. Η πρόσβαση σε πολλά δεδομένα που έχουν διαφορετική προέλευση μπορεί να οδηγήσει σε σημαντικούς συμβιβασμούς στην προστασία της ανωνυμίας των χρηστών, ακόμα και αν τα δεδομένα έχουν υποστεί μια πρώτη ανωνυμοποίηση (αν δηλαδή έχουν αποκρυφθεί οι πιο "ευαίσθητες" πληροφορίες, όπως το Όνομα ή το ΑΦΜ).

Ένας τρόπος με τον οποίο μπορούν να ανακτηθούν ευαίσθητες πληροφορίες ακόμα και από "ανωνυμοποιημένα" δεδομένα, όπως αυτά που περιγράφηκαν παραπάνω, αφορά την σύνδεση στοιχείων (linking ή matching) ανάμεσα σε δεδομένα διαφορετικής προέλευσης που έχουν τουλάχιστον n κοινά πεδία. Σε αυτή την περίπτωση, συνδέοντας πληροφορίες από διαφορετικές πηγές, μπορούμε να εξάγουμε συμπεράσματα για την ταυτότητα των ατόμων ενώ σε πολλές περιπτώσεις, μπορούμε να προδιορίσουμε και την ακριβή ταυτότητα του ατόμου. Η αποτελεσματικότητα αυτής της μεθόδου οφείλεται στο γεγονός ότι όλοι οι πιθανοί οι συνδυασμοί των τιμών των n κοινών πεδίων είναι πάρα πολλοί σε αριθμό,

με συνέπεια κάθε πιθανός συνδυασμός να καθορίζει ένα άτομο ή μια μικρή ομάδα ατόμων με μονοσήμαντο τρόπο. Στο παράδειγμα 1.1.1 αναφέρουμε μια περίπτωση linking στην οποία προσδιορίστηκε η ταυτότητα ενός ατόμου με συνδυασμό ιατρικών και εκλογικών καταλόγων (το παράδειγμα που παραθέτουμε βρίσκεται στο άρθρο του L.Sweeney[1] που εισάγει το k-anonymity).

Βλέπουμε λοιπόν ότι η απόκρυψη προσωπικών πληροφοριών δεν αρκεί για την διατήρηση της ανωνυμίας. Μια άλλη λύση που θα μπορούσαμε να εφαρμόσουμε, επομένως είναι η δημοσίευση δεδομένων τα οποία είναι γενικευμένα. Με τον όρο γενικευμένα εννοούμε ότι οι τιμές των n πεδίων δεν θα περιέχουν απαραίτητα ορισμένες τιμές αλλά θα μπορούν να περιέχουν και διαστήματα τιμών. Για παράδειγμα αν υπάρχουν 3 κοινά πεδία μεταξύ δυο πηγών δεδομένων και τα πεδία αυτά είναι: $(Age, Gender, Zipcode)$ και οι τιμές ενός ατόμου στα πεδία αυτά είναι $(22, Male, 55151)$ τότε, αν γενικεύσουμε τα δεδομένα ως προς το Age και το $Zipcode$ τότε το άτομο θα έχει τις εξής τιμές: $([20-25], Male, 5515*)$[1], όπου το $*$ συμβολίζει οποιαδήποτε τιμή στο διάστημα $[0, 9]$[2]. Με αυτόν τον τρόπο προσπαθούμε να αποδώσουμε σε κάθε άτομο πολλούς πιθανούς συνδυασμούς των τιμών των n κοινών πεδίων, έτσι ώστε να γίνει αρκετά δυσκολότερη η άρση της ανωνυμίας με το linking. Είναι σαφές από τα παραπάνω ότι η γενίκευση αποτελεί απώλεια πληροφορίας. Για αυτό το λόγο, αν γενικεύσουμε αυθαίρετα ή αν τα γενικευμένα δεδομένα έχουν μεγάλα διαστήματα τιμών, χάνουμε μεγάλο μέρος της πληροφορίας και τα δεδομένα μπορεί να σταματήσουν να είναι χρήσιμα.

Μια δεύτερη επιδίωξη, πέρα από την επίτευξη υψηλής ποιότητας στην ανωνυμοποίηση (υπό την έννοια της κατά τον δυνατόν μικρότερης απώλειας πληροφορίας) είναι η γρήγορη και άμεση εκτέλεσης της. Μέχρι στιγμής, η ανωνυμοποίηση υλοποιούταν με κεντρικό τρόπο και τις περισσότερες φορές εκτελούταν σε έναν server ο οποίος με κεντρικό τρόπο διάβαζε τα δεδομένα και παρήγαγε τα ανωνυμοποιημένα δεδομένα. Από τη στιγμή όμως που τη σημερινή εποχή όλο και περισσότερες εφαρμογές εκμεταλλεύονται τις δυνατότητες των κατανεμημένων συστημάτων είναι σημαντικό να αναπτυχθούν κατανεμημένες τεχνικές για την ανωνυμοποίηση δεδομένων.

Σκοπός της συγκεκριμένης εργασίας, με βάση τα παραπάνω, είναι η αποδοτική ανωνυμοποίηση δεδομένων, με χρήση κατανεμημένων τεχνικών. Πρέπει δηλαδή η μέθοδος που θα υλοποίησουμε να εξασφαλίζει ότι η ανωνυμία του ατόμου δεν θα μπορεί να αρθεί, αλλά παράλληλα θα πρέπει να υπάρχει η ελάχιστη δυνατή απώλεια πληροφορίας. Επιπρόσθετα, η μέθοδος θα πρέπει να τρέχει με κατανεμημένο τρόπο, έτσι ώστε να μπορεί να χειρίζεται πολύ μεγάλο όγκο δεδομένων χωρίς πρόβλημα, αλλά και για να μπορεί να αξιοποιεί τα χαρακτηριστικά των κατανεμημένων συστημάτων που αναφέρθηκαν και προηγούμενα (scalability, fault-tolerance).

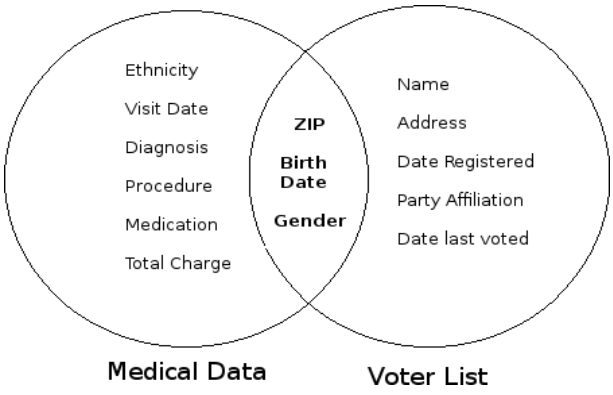
[1] Αυτή αποτελεί μια πιθανή γενίκευση από όλες τις πιθανές.

[2] Το $5515*$ και το $[55150 - 55159]$ είναι ισοδύναμες εκφράσεις

1.1.1 Παράδειγμα: άρση ανωνυμίας με linking

Το 1990 στις ΗΠΑ, πραγματοποιήθηκαν πειράματα με βάση δεδομένα του US Census, με σκοπό να προσδιοριστεί ο αριθμός των ατόμων που έχουν δημογραφικά χαρακτηριστικά που μπορούν να θεωρηθούν αρκετά σπάνια. Ένα εντυπωσιακό αποτέλεσμα της έρευνας ήταν ότι το 87% περίπου των ατόμων που συνιστούν τον πληθυσμό των Ηνωμένων Πολιτειών (216 εκατομμύρια από τα περίπου 248 εκατομμύρια πληθυσμού) έχει δημογραφικά χαρακτηριστικά που καθιστούν κάθε άτομο μοναδικό ή σχεδόν μοναδικό με βάση μόνο τα στοιχεία $(Zipcode, Gender, DateofBirth)$. Όπως είναι εύκολα κατανοητό, τα τρία αυτά στοιχεία εμφανίζονται πολύ συχνά σε διαφορετικά δεδομένα, κάτι που σημαίνει ότι η άρση της ανωνυμίας των ατόμων γίνεται ευκολότερη.

Ένα συγκεκριμένο παράδειγμα linking των 3 στοιχείων που αναφέρθηκαν για την άρση της ανωνυμίας, απεικονίζεται στο Σχήμα 1.1. Στο ίδιο άρθρο[1], συνδέθηκαν ιατρικά δεδομένα με τα δεδομένα εκλογικού καταλόγου. Τα ιατρικά δεδομένα δεν περιέχουν προσωπικά στοιχεία και για αυτό το λόγο θεωρήθηκαν ανωνυμοποιημένα και δημοσιεύτηκαν με τα στοιχεία που φαίνονται στο αριστερό μέρος της εικόνας. Τα δεδομένα του εκλογικού κατάλογου αντίστοιχα, περιέχουν τα στοιχεία που φαίνονται στο δεξιό μέρος του σχήματος. Βλέπουμε ότι αν συνδέσουμε τους δυο καταλόγους ως προς τα στοιχεία $(Zipcode, Gender, DateOfBirth)$ μπορούμε να λάβουμε πληροφορίες για την κατάσταση υγείας ενός ατόμου. Πράγματι, η διασταύρωση που προσπάθησε ο συγγραφέας ξεκίνησε με βάση την ημερομηνία γέννησης του κυβερνήτη της Μασαχουσέτης (William Weld). Τα στοιχεία του Weld βρίσκονταν στα ιατρικά δεδομένα, χωρίς όνομα ή άλλα προσωπικά στοιχεία. Στα ιατρικά δεδομένα υπήρχαν μόνο 6 άτομα με την ημερομηνία γέννησης του Weld, εκ των οποίων μόνο οι 3 ήταν άντρες και από αυτούς μόνο ο Weld είχε το σωστό Zipcode.



Σχήμα 1.1: linking για την άρση της ανωνυμίας

Βλέπουμε λοιπόν πόσο εύκολα μπορούμε με τη βοήθεια του linking να άρουμε την ανωνυμία και να λάβουμε ευαίσθητες πληροφορίες από δεδομένα που δεν περιέχουν προσωπικές πληροφορίες. Κάτι ακόμη που γίνεται σαφές από το παράδειγμα αυτό είναι ότι

δεν μπορούμε να κάνουμε καμία παραδοχή για τα διαφορετικά δεδομένα που θα συνδεθούν. Δεν μπορούμε να ξέρουμε με ποια δεδομένα θα συνδεθούν τα δεδομένα που θέλουμε να ανωνυμοποιήσουμε, αλλά οφείλουμε να εκτελέσουμε την ανωνυμοποίηση με τρόπο τέτοιο έτσι ώστε ανεξάρτητα από τις άλλες πηγές δεδομένων, να μην υπάρχει τρόπος να αρθεί η ανωνυμία.

1.2 Σημασία κατανεμημένης ανωνυμοποίησης

Αναφέραμε προηγουμένως ότι η ανωνυμοποίηση των δεδομένων έχει γίνει απαραίτητη στις μέρες μας, εξαιτίας του τεράστιου όγκου δεδομένων που έχουν στη διάθεσή τους οι χρήστες. Αν και η ανάγκη έχει γίνει πιο επιτακτική τώρα, είναι γεγονός πως η διατήρηση της ανωνυμίας των χρηστών που χρησιμοποιούν μια υπηρεσία ήταν ζητούμενη από τότε που εμφανίστηκαν οι πρώτες πιο προσωποποιημένες υπηρεσίες.

Από την πλευρά των εταιρειών, είναι απαραίτητο να μπορούν να εξασφαλίσουν ότι τα άτομα που χρησιμοποιούν τις υπηρεσίες τους θα μπορούν να διατηρούν την ανωνυμία τους σε κάθε περίπτωση. Στην αντίθετη περίπτωση, οι χρήστες θα χάσουν σταδιακά την εμπιστοσύνη τους στην εταιρεία, με αποτέλεσμα να αναζητήσουν άλλη παρόμοια υπηρεσία σε κάποιον ανταγωνιστή. Παράλληλα όμως, ενώ οι χρήστες θέλουν να διατηρούν την ανωνυμία τους, θέλουν ταυτόχρονα να έχουν πρόσβαση σε μεγάλο όγκο δεδομένων, που όπως είδαμε και προηγουμένως, αυτό ακριβώς το γεγονός αρχίζει να προκαλεί προβλήματα στη διατήρηση της ανωνυμίας των ατόμων. Με την ανωνυμοποίηση των δεδομένων μπορούμε να συμβιβάσουμε τις δυο αυτές αντίρροπες τάσεις. Μπορούμε δηλαδή και να εξασφαλίσουμε τη διατήρηση της ανωνυμίας και την παροχή μεγάλου όγκου δεδομένων.

Βλέπουμε λοιπόν ότι η ανωνυμοποίηση είναι μια απαραίτητη διαδικασία που πρέπει να προηγηθεί πριν την δημοσίευση δεδομένων. Αν συνυπολογίσουμε μάλιστα και το γεγονός ότι ο όγκος των δεδομένων συνεχώς αυξάνεται και ότι διαρκώς γεννιούνται νέα δεδομένα, γίνεται σαφές ότι η ανωνυμοποίηση πρέπει να γίνεται γρήγορα και πρέπει να γίνεται για οσοδήποτε μεγάλα dataset. Για αυτό το λόγο επιδιώκουμε να εκτελέσουμε την ανωνυμοποίηση με κατανεμημένο τρόπο. Η κατανεμημένη αρχιτεκτονική θα μας δώσει πολλούς επεξεργαστικούς πόρους, πολύ μεγάλο αποθηκευτικό χώρο (για να χωράνε οσοδήποτε μεγάλα δεδομένα), μεγάλη κλιμακωσιμότητα (δηλαδή η αύξηση των δεδομένων θα σημαίνει αύξηση των επεξεργαστικών κόμβων που χρησιμοποιούνται) και την ανοχή σε σφάλματα με μικρό κόστος. Τα πλεονεκτήματα της κατανεμημένης σχεδίασης θα αναφερθούν εκτενώς και στα κεφάλαια που ακολουθούν.

Με βάση τα παραπάνω, μπορούμε να καταλάβουμε γιατί η έρευνα επάνω στην ανωνυμοποίηση με κατανεμημένο τρόπο είναι τόσο ενεργή και γιατί είναι τόσο σημαντική η δημιουργία και η βελτίωση των συστημάτων που εξασφαλίζουν την ανωνυμία δεδομένων.

1.3 Γενική Λύση προβλήματος

Σε αυτό το σημείο θα παρουσιάσουμε σε γενικές γραμμές τη λύση που δώσαμε στο πρόβλημα που παρουσιάσαμε στην ενότητα 1.1. Όπως έχουμε αναφέρει και προηγουμένως, ο στόχος μας είναι να υλοποιήσουμε ένα κατανεμημένο σύστημα ανωνυμοποίησης, το οποίο θα εξασφαλίζει ότι δεν μπορεί να αρθεί η ανωνυμία των δεδομένων μέσω του linking, αλλά παράλληλα, η πληροφορία που θα χάνεται θα είναι η λιγότερη δυνατή.

Σε πρώτη φάση, για την ανωνυμοποίηση, χρησιμοποιήσαμε το μοντέλο k-anonymity[3][1] σύμφωνα με το οποίο (τουλάχιστον) k tuples γενικεύονται στην ίδια ομάδα, έτσι ώστε οι τιμές των attributes τους να βρίσκονται στα ίδια διαστήματα τιμών για κάθε attribute. Σύμφωνα με αυτό το μοντέλο, έχουν αναπτυχθεί διάφοροι αλγόριθμοι, οι οποίοι δέχονται σαν είσοδο ένα σύνολο από tuples και έχουν έξοδο μια διαμέριση της εισόδου, έτσι ώστε τα στοιχεία που ανήκουν στην ίδια ομάδα να είναι όσο πλησιέστερα γίνεται. Στην παρούσα εργασία θα χρησιμοποιήσουμε τέσσερις μεθόδους ανωνυμοποίησης τις οποίες και θα συγκρίνουμε για διάφορες περιπτώσεις δεδομένων. Θα κάνουμε μια αναλυτική παρουσίαση των αλγορίθμων που χρησιμοποιήσαμε στην Ενότητα 3.1.

Σε δεύτερη φάση, για να επιτύχουμε την κατανεμημένη εκτέλεση πρέπει να παραλληλοποιήσουμε το πρόβλημα. Αυτό σημαίνει ότι θα πρέπει με κάποιο τρόπο που να εξασφαλίζει όσο το δυνατόν μεγαλύτερη απόδοση, να διαμερίσουμε τα δεδομένα της εισόδου και στη συνέχεια κάθε διαμέριση να δωθεί ως είσοδος σε ένα αλγόριθμο που θα τρέχει σε ξεχωριστό κόμβο του κατανεμημένου συστήματος. Με αυτόν τον τρόπο επιτυγχάνουμε την παράλληλη εκτέλεση και επιταχύνουμε την διαδικασία. Ένα σημαντικό πρόβλημα όμως στην παραπάνω διαδικασία είναι ότι με την αρχική διαμέριση τα δεδομένα χωρίζονται σε αρχικές ομάδες με κάποιο greedy τρόπο και εξασφαλίζεται ότι δυο στοιχεία που ανήκουν σε διαφορετικές (αρχικές) ομάδες δεν θα καταλήξουν στην ίδια τελική ομάδα. Αυτό μπορεί να προκαλέσει σημαντικό πρόβλημα αν τα στοιχεία αυτά βρίσκονται αρκετά κοντά μεταξύ τους και μπορεί να μειώσει κατά πολύ την απόδοση της ανωνυμοποίησης.

Για την επίλυση αυτού του προβλήματος αρχικά ταξινομούμε όλα τα στοιχεία με βάση μια σχέση διάταξης μεταξύ των στοιχείων που στηρίζεται στη "σημαντικότητα" των attributes και στη συνέχεια δημιουργούμε τομές στα ταξινομημένα δεδομένα έτσι ώστε να προκύψουν οι αρχικές ομάδες. Με τον τρόπο αυτό επιτυγχάνουμε τη γρήγορη αρχική διαμέριση των δεδομένων και όπως αποδεικνύεται και πειραματικά (Κεφάλαιο 4) με τη χρήση αυτής της μεθόδου η μείωση της ποιότητας ανωνυμοποίησης είναι σχετικά μικρή ενώ παράλληλα σε κάποιες περιπτώσεις, η εφαρμογή αυτής της αρχικής διαμέρισης των δεδομένων και η παράλληλη εκτέλεση των αλγορίθμων οδηγεί σε υψηλότερη ποιοτικά ανωνυμοποίηση σε σχέση με την κεντρική εκτέλεση ενός ενιαίου συνόλου δεδομένων, συνεπώς καταφέραμε να βρούμε μια αποδοτική μέθοδο για την αρχική διαμέριση των στοιχείων σε ομάδες.
Περιγράφουμε αναλυτικά τη διαδικασία κατανεμημένης ανωνυμοποίησης στην Ενότητα 3.2.

[3]Παρουσιάζουμε αναλυτικά το μοντέλο k-anonymity στο κεφάλαιο 2.1.

1.4 Οργάνωση εγγράφου

Σε αυτό το πρώτο κεφάλαιο, αναλύσαμε το πρόβλημα που επιλύσαμε στην εργασία αυτή, αναφερθήκαμε στη σημασία του προβλήματος, ενώ ταυτόχρονα κάναμε και μια πρώτη περιγραφή του τρόπου με τον οποίο επιλύσαμε το πρόβλημα. Στη συνέχεια:

- Στο Κεφάλαιο 2, θα ορίσουμε τυπικά κάποιες απαραίτητες έννοιες που μας χρειάζονται για τη συνέχεια, θα παρουσιάσουμε το k-anonymity, το οποίο αποτελεί το θεωρητικό μοντέλο της ανωνυμοποίησης, θα αναφερθούμε στο μετασχηματισμό γενίκευσης των δεδομένων (recoding), ενώ παράλληλα θα επιχειρήσουμε μια πρώτη προσέγγιση στην κατανεμημένη σχεδίαση του συστήματος.

- Στο Κεφάλαιο 3, θα αναλύσουμε τις μεθόδους ανωνυμοποίησης που χρησιμοποιήσαμε και θα αναφερθούμε εκτενώς στην κατανεμημένη εκτέλεσή τους.

- Στο Κεφάλαιο 4, θα παραθέσουμε διάφορες πειραματικές τιμές που λάβαμε με την εκτέλεση της ανωνυμοποίησης στα datasets, όπως η ποιότητα της ανωνυμοποίησης (με χρήση ορισμένων μετρικών), ο χρόνος εκτέλεσης, κλπ.

- Στο Κεφάλαιο 5, θα παρουσιάσουμε τα συμπεράσματα που προέκυψαν από την εκτέλεση της ανωνυμοποίησης.

- Τέλος, στις τελευταίες σελίδες παραθέτουμε όλες τις αναφορές που γίνονται από αυτή την εργασία σε διάφορες άλλες δημοσιεύσεις, άρθρα και βιβλία.

Κεφάλαιο 2

Θεωρητικό Μέρος

Στο κεφάλαιο αυτό, θα παρουσιάσουμε το μοντέλο k-anonymity[1], θα παρουσιάσουμε τις κατηγορίες των αλγορίθμων ανωνυμοποίησης με βάση το μετασχηματισμό γενίκευσης που χρησιμοποιείται και τέλος, θα κάνουμε μια μικρή εισαγωγή στο προγραμματιστικό μοντέλο MapReduce και στο Hadoop framework που το υλοποιεί.

2.1 k-anonymity

Έστω ένα σύνολο δεδομένων D. Τα δεδομένα είναι οργανωμένα κατά γραμμές και στήλες έτσι ώστε κάθε γραμμή αφορά διαφορετικό άτομο[1] και κάθε στήλη αφορά διαφορετική ιδιότητα του ατόμου. Το σύνολο των δεδομένων D το αποκαλούμε και πίνακα δεδομένων και κάθε γραμμή του πίνακα θα την αποκαλούμε στο εξής tuple. Για κάθε πίνακα D ορίζουμε:

Ορισμός 1 *Έστω $D(A_1, A_2, ..., A_n)$ πίνακας δεδομένων με πεπερασμένο αριθμό από tuples. Το πεπερασμένο σύνολο $\{A_1, A_2, ..., A_n\}$ είναι το σύνολο των **attributes** του D.*

Για παράδειγμα, έστω ο πίνακας δεδομένων που φαίνεται στον Πίνακα 2.1, στον οποίο απεικονίζονται ιατρικά δεδομένα κάποιων ασθενών[2]. Σε αυτόν τον πίνακα, με βάση των παραπάνω ορισμό το σύνολο των attributes είναι το σύνολο $\{Age, Sex, Zipcode, Disease\}$ ενώ τα tuples είναι οι γραμμές του πίνακα.

Όπως είναι σαφές από τον παραπάνω ορισμό και τον παραπάνω πίνακα, κάθε attribute του πίνακα είναι μοναδικό, αφού αποδίδει μια σημασιολογική ιδιότητα σε κάθε tuple του πίνακα. Πέρα από την ιδιότητα, κάθε attribute συνδέεται άμεσα και με ένα πεδίο τιμών. Για παράδειγμα, το attribute $\{Sex\}$ του πίνακα 2.1, έχει ως πεδίο τιμών το σύνολο $\{Male, Female\}$. Αντίθετα, κάθε tuple στον πίνακα δεν είναι υποχρεωτικά μοναδικό.

Ορίζουμε τώρα την έννοια της όψης πίνακα ή προβολής πίνακα σε άλλον πίνακα που

[1]Επειδή το πρόβλημα που εξετάζουμε αφορά την εξασφάλιση της ανωνυμίας ατόμων, θεωρούμε ότι κάθε σύνολο δεδομένων που εξετάζουμε αφορά πληροφορίες που αφορούν φυσικά πρόσωπα ή άτομα. Αυτή η παραδοχή θα ισχύει στο εξής γενικά εκτός και αν δηλωθεί κάτι διαφορετικό.

[2]Ο συγκεκριμένος πίνακας δεδομένων αναφέρεται στο [2].

Age	**Sex**	**Zipcode**	**Disease**
25	Male	53711	Flu
25	Female	53712	Hepatitis
26	Male	53711	Brochitis
27	Male	53710	Broken Arm
27	Female	53712	AIDS
28	Male	53711	Hang Nail

Πίνακας 2.1: Ένας απλός πίνακας δεδομένων (D)

περιέχει τα ίδια tuples αλλά ένα υποσύνολο από τα attributes του αρχικού πίνακα.

Ορισμός 2 *Έστω πίνακας δεδομένων D με σύνολο attributes A και έστω $A' \subseteq A$. Ο πίνακας D' ο οποίος περιέχει τα ίδια tuples με τον D, με σύνολο attributes το A' θα ονομάζεται* ***προβολή*** *ή* ***όψη*** *του πίνακα D ως προς A' και θα συμβολίζεται με $D[A']$.*

Για παράδειγμα, η προβολή του Πίνακα 2.1 ως προς το σύνολο $A' = \{Age, Zipcode\}$ συμβολίζεται ως $D[Age, Zipcode]$ και είναι ο πίνακας που ακολουθεί. Βλέπουμε ότι οι πίνακες περιέχουν ακριβώς τα ίδια tuples με τη διαφορά ότι στον πίνακα προβολή υπάρχουν λιγότερα attributes.

Age	**Zipcode**
25	53711
25	53712
26	53711
27	53710
27	53712
28	53711

Πίνακας 2.2: Προβολή τού Πίνακα D ως προς το QI_D

.

Συνεχίζοντας, ορίζουμε την έννοια του quasi identifier που παίζει πολύ σημαντικό ρόλο στο μοντέλο k-anonymity.

Ορισμός 3 *Έστω πίνακας D με σύνολο attributes A. Ορίζουμε ως* ***Quasi Identifier*** *ή* ***QI*** *και συμβολίζουμε με QI_D ένα ελάχιστο υποσύνολο του A, τέτοιο ώστε ο πίνακας $D[QI_D]$ αν συνενωθεί με άλλα δεδομένα να μπορεί να οδηγήσει σε κατάργηση της ανωνυμίας ενός ή περισσότερων ατόμων.*

Είναι σαφές ότι το QI δεν είναι μοναδικό και μπορούμε να πούμε ότι η επιλογή των attributes που δημιουργούν το QI σε κάθε περίπτωση εξαρτάται από το είδος των εξωτερικών δεδομένων που έχουμε διαθέσιμα για linking.

Αν για παράδειγμα, ενώσουμε τον πίνακα D με $QI_D = \{Age, Zipcode\}$ με τον πίνακα που ακολουθεί (έστω V) και αφορά δεδομένα που περιέχονται σε εκλογικό κατάλογο τότε μπορούμε να βρούμε ότι ο Ahmed πάσχει από ίωση (Flu).

Name	Age	Sex	Zipcode
Ahmed	25	Male	53711
Brooke	28	Female	55410
Claire	31	Female	90210
Dave	19	Male	02174
Evelyn	40	Female	02237

Πίνακας 2.3: Εκλογικός Κατάλογος (V)

Age	Zipcode
25	53711
28	55410
31	90210
19	02174
40	02237

Πίνακας 2.4: Προβολή του Πίνακα V ως προς το QI_D

Αυτό μπορούμε να το βρούμε αν συνενώσουμε τους πίνακες $D[QI_D]$ και $V[QI_D]$ (Πίνακας 2.2 και Πίνακας 2.4 αντίστοιχα). Από τις δυο όψεις πινάκων βρίσκουμε ότι το tuple $(25, 53711)$ είναι κοινό. Από τις πληροφορίες που μας δίνουν οι πίνακες D και V προκύπτει ότι το άτομο που αναπαριστά το tuple λέγεται Ahmed, πάσχει από ίωση (Flu) και είναι άντρας (Male). Βλέπουμε λοιπόν ότι μπορούμε με κατάλληλη επιλογή του QI να άρουμε την ανωνυμία με τη βοήθεια του linking.

Με βάση τους παραπάνω ορισμούς, μπορούμε τώρα να εκφράσουμε την ιδιότητα του k-anonymity

Ορισμός 4 *Έστω πίνακας $D(A_1, A_2, ..., A_n)$ και QI_D το Quasi Identifier του πίνακα αυτού. Λέμε ότι ο πίνακας D ικανοποιεί το **k-anonymity** αν και μόνο αν κάθε tuple του πίνακα $D[QI_D]$ εμφανίζεται στον πίνακα τουλάχιστον k φορές.*

Ας δούμε πως μπορούμε να μεταβάλουμε τον Πίνακα 2.1 έτσι ώστε να ικανοποιεί το k-anonymity για διάφορες τιμές του k για QI_D=$\{Age, Zipcode\}$.

Age	Sex	Zipcode	Disease
25	Male	[53711-53712]	Flu
25	Female	[53711-53712]	Hepatitis
[26-28]	Male	53711	Brochitis
27	Male	[53710-53712]	Broken Arm
27	Female	[53710-53712]	AIDS
[26-28]	Male	53711	Hang Nail

Πίνακας 2.5: Πίνακας D που ικανοποιεί το k-anonymity (k=2)

Age	Sex	Zipcode	Disease
[25-26]	Male	[53711-53712]	Flu
[25-26]	Female	[53711-53712]	Hepatitis
[25-26]	Male	[53711-53712]	Brochitis
[27-28]	Male	[53710-53712]	Broken Arm
[27-28]	Female	[53710-53712]	AIDS
[27-28]	Male	[53710-53712]	Hang Nail

Πίνακας 2.6: Πίνακας D που ικανοποιεί το k-anonymity (k=3)

Με τη βοήθεια των παραπάνω πινάκων (Πίνακας 2.5 και Πίνακας 2.6) μπορούμε να

κατανοήσουμε γιατί η ικανοποίηση του k-anonymity σημαίνει τη διατήρηση της ανωνυμίας των ατόμων. Αν επιχειρήσουμε linking όπως προηγουμένως, θα δούμε ότι το tuple από το $V[QI_D]$ δεν είναι ίδιο με κάποιο tuple από το $D[QI_D]$ αλλά βρίσκεται εντός των διαστήματων που δείχνουν 2 tuples του $D[QI_D]$ του πίνακα 2.5 και 3 tuples του $D[QI_D]$ του πίνακα 2.6. Δηλαδή, *αν ένας πίνακας D ικανοποιεί το k-anonymity τότε κάθε τυχαίο tuple που έχει ως attributes το QI_D, είτε θα βρίσκεται εντός των διαστημάτων που ορίζουν τουλάχιστον k (ίδια) tuples του D είτε θα βρίσκεται εκτός των διαστημάτων που ορίζουν όλα τα tuples του D.* Με άλλα λόγια, κάθε προσπάθεια linking θα καταλήγει είτε σε k tuples τα οποία δεν μπορούμε να ξεχωρίσουμε είτε σε κανένα.

Από τα παραπάνω βλέπουμε ότι όσο μεγαλύτερο είναι το k τόσο πιο "ισχυρή" είναι η ανωνυμοποίηση[3]. Παράλληλα όμως, όσο μεγαλύτερο είναι το k, τόσο περισσότερη πληροφορία χάνουμε. Αυτό μπορούμε να το καταλάβουμε εποπτικά με μια πρόχειρη σύγκριση[4] των πινάκων 2.5 και 2.6. Βλέπουμε ότι τα tuples του πίνακα 2.5 έχουν attributes με μικρότερα διαστήματα, άρα με μεγαλύτερη ακρίβεια και περισσότερη πληροφορία, ενώ αντίθετα, τα tuples του πίνακα 2.6 έχουν attributes με μεγαλύτερα διαστήματα, άρα είναι πιο γενικά και περιέχουν λιγότερη πληροφορία. Όπως γίνεται εύκολα κατανοητό, όσο πιο "γενικό" είναι ένα tuple τόσο λιγότερη πληροφορία περιέχει.

Στην πράξη, θέλουμε την "ισχυρότερη" δυνατή ανωνυμοποίηση αλλά παράλληλα επιζητούμε και την απώλεια της ελάχιστης δυνατής πληροφορίας. Όπως είναι γνωστό, το πρόβλημα της βέλτιστης k-ανωνυμοποίησης ανήκει στην κλάση προβλημάτων NP-complete, συνεπώς δεν επιζητούμε την τέλεια λύση, αλλά επιζητούμε μια ικανοποιητική λύση που πραγματοποιείται σε σύντομο χρόνο. Μια απόδειξη της πολυπλοκότητας του optimal k-anonymity προβλήματος βρίσκεται στο άρθρο [3].

2.2 recoding

Είδαμε προηγουμένως ότι η τροποποίηση ενός πίνακα δεδομένων έτσι ώστε να ικανοποιεί το k-anonymity είναι άρρηκτα συνδεδεμένη με τη γενίκευση των tuples που τον συνιστούν. Αυτή η διαδικασία γενίκευσης των tuples ενός πίνακα δεδομένων, μπορεί να αναπαρασταθεί με τη βοήθεια ενός μετασχηματισμού γενίκευσης. Ο μετασχηματισμός αυτός ονομάζεται **recoding**. Μαθηματικά, κάθε μετασχηματισμός γενίκευσης μπορεί να εκφραστεί με τον ακόλουθο τρόπο:

$$\phi : T \rightarrow T' \tag{2.1}$$

[3]Από την άποψη ότι όσο περισσότερα είναι τα tuples στα οποία έχουμε καταλήξει από το linking με άλλα δεδομένα, τόσο πιο δύσκολο είναι να κάνουμε σωστή ταυτοποίηση του ατόμου, το οποίο αναπαρίσταται από ένα από τα k tuples.

[4]Στη συνέχεια θα δώσουμε ακριβής τρόπους με τους οποίους μπορούμε να υπολογίσουμε το ποσοστό της πληροφορίας που χάνεται.

όπου T είναι ο χώρος των tuples και ισχύει $T = X_1 \times X_2 \times ...X_n$, όπου X_i είναι το πεδίο τιμών του i-οστού attribute και T' είναι ο χώρος των γενικευμένων tuples για τον οποίο ισχύει αντίστοιχα $T' = X'_1 \times X'_2 \times ...X'_n$.

Στην πιο απλή περίπτωση η γενίκευση ενός tuple πραγματοποιείται με την γενίκευση κάποιων ή όλων των attributes του tuple ξεχωριστά. Πιο συγκεκριμένα, η τιμή κάθε attribute ενός tuple αντιστοιχίζεται σε ένα ευρύτερο διάστημα που την περιέχει (πχ, το attribute ηλικία με τιμή 23 μπορεί να αντιστοιχίζεται στο πεδίο τιμών $[20-25]$). Αυτή η διαδικασία επαναλαμβάνεται για όλα τα attributes όλων των tuples. Τελικά, αυτό που έχουμε κάνει είναι να δημιουργήσουμε έναν κανόνα με τον οποίο το attribute κάθε tuple γενικεύεται σε ένα συγκεκριμένο πεδίο τιμών με κριτήριο μόνο την τιμή του (και όχι για παράδειγμα τις τιμές άλλων attributes του ίδιου tuple). Για αυτό το λόγο λέμε ότι το συγκεκριμένο είδος recoding λειτουργεί καθολικά, αφού ανεξάρτητα από το tuple οι τιμές των ίδιων attributes γενικεύονται με τον ίδιο τρόπο και για αυτό το λόγο ονομάζεται **global recoding**[4]. Η μαθηματική έκφραση του μετασχηματισμού σε αυτήν την περίπτωση θα έχει την εξής μορφή:

$$\phi_i^g : X_i \to X'_i \qquad i = 1..n \tag{2.2}$$

$$\phi^g(t) = (\phi_1^g(x_1), \phi_2^g(x_2), ..., \phi_n^g(x_n)) \qquad t = (x_1, x_2, ..., x_n) \in T \tag{2.3}$$

όπου με ϕ_i^g συμβολίζουμε την τιμή που λαμβάνει ο μετασχηματισμός στην i-οστή του διάσταση, κάθε tuple έχει n attributes (τόσες προφανώς είναι και οι διαστάσεις του μετασχηματισμού), X_i είναι το πεδίο τιμών του i-οστού attribute και X'_i είναι το γενικευμένο πεδίο τιμών και $t = (x_1, x_2, ..., x_n)$.

Στη γενικότερη περίπτωση όμως, η γενικευμένη τιμή ενός attribute δεν καθορίζεται μόνο από την αντίστοιχη αρχική τιμή του tuple, αλλά μπορεί να καθορίζεται και από τις τιμές άλλων attributes του ίδιου tuple. Σε αυτήν την περίπτωση μπορεί να υπάρχουν 2 tuples που έχουν ίδιες τιμές στο ίδιο attribute, αλλά τελικά μπορεί να γενικεύονται σε διαφορετικά διαστήματα, για παράδειγμα δυο tuples που έχουν στο attribute ηλικία την τιμή 23, μπορεί να γενικεύονται στα διαστήματα $[21-24]$ και $[22-25]$ αντίστοιχα. Σε αυτόν τον μετασχηματισμό εξετάζουμε την τοποθεσία του tuple και με βάση αυτή αποφασίζουμε την γενίκευση που θα εφαρμοστεί. Ο μετασχηματισμός αυτός ονομάζεται **local recoding** ([2],[5],[6]) και περιγράφεται ως εξής:

$$\phi_i^l : T \to X'_i \qquad i = 1..n \tag{2.4}$$

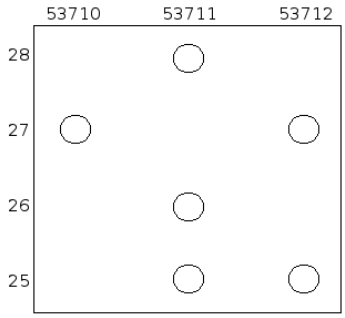
$$\phi^l(t) = (\phi_1^l(t), \phi_2^l(t), ..., \phi_n^l(t)) \qquad t \in T \tag{2.5}$$

όπου όλα τα σύμβολα που χρησιμοποιούνται έχουν παρόμοια σημασία με πριν.

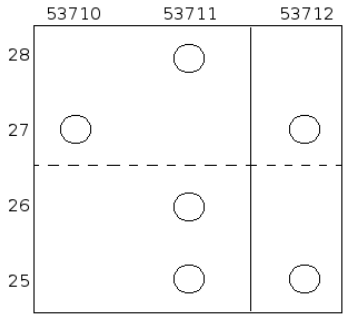
Αν συγκρίνουμε το global και το local recoding, θα δούμε ότι ενώ με το global recoding απεικονίζεται το πεδίο τιμών ενός attribute σε ένα γενικευμένο πεδίο τιμών, με το local recoding απεικονίζεται ένα tuple σε ένα γενικευμένο tuple. Το global recoding αποτελεί

μια ειδική περίπτωση του local recoding και από άποψη απόδοσης, το global recoding οδηγεί σε γενίκευση που στην καλύτερη περίπτωση είναι όσο καλή είναι και η γενίκευση μέσω του local recoding. Γενικά δηλαδή, το local recoding μπορεί να πετύχει λιγότερη απώλεια πληροφορίας σε σχέση με το global recoding και για αυτό προτιμάται. Στη συνέχεια θα παραθέσουμε ένα παράδειγμα στο οποίο θα συγκρίνουμε τους δυο μετασχηματισμούς και θα φανεί ο λόγος για τον οποίο θεωρείται αποδοτικότερο το local recoding.

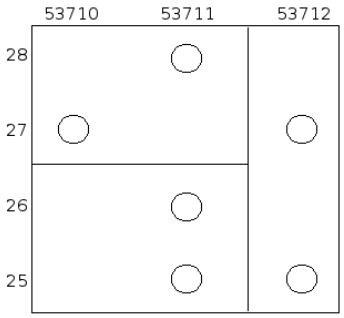
Ένας άλλος τρόπος να κατηγοριοποιήσουμε τον μετασχηματισμό γενίκευσης των δεδομένων είναι με βάση τις διαστάσεις που εξετάζουμε κάθε φορά προς γενίκευση. Στην απλή περίπτωση η γενίκευση γίνεται για κάθε διάσταση ξεχωριστά και το recoding χαρακτηρίζεται ως **single dimensional recoding**. Στην γενική περίπτωση, η γενίκευση απεικονίζει όχι ένα attribute (ή ισοδύναμα μια διάσταση κάθε φορά), αλλά απεικονίζει το καρτεσιανό γινόμενο πολλών attributes ή διαστάσεων. Σε αυτήν την περίπτωση έχουμε **multi dimensional recoding**. Στις εικόνες που ακολουθούν παραθέτουμε τα δεδομένα του Πίνακα 2.1 ως προς τα πεδία Age και Zipcode (η αναπαράσταση γίνεται στο επίπεδο) και παρουσιάζουμε μια διαμέριση των σημείων χρησιμοποιώντας single dimensional recoding και multi dimensional recoding.



Σχήμα 2.1: data



Σχήμα 2.2: single



Σχήμα 2.3: multi

Στο Σχήμα 2.1 απεικονίζονται, όπως αναφέραμε, τα δεδομένα του Πίνακα 2.1 στο δισδιάστατο επίπεδο με κάθετο άξονα Age και οριζόντιο Zipcode. Στο Σχήμα 2.2 απεικονίζεται μια διαμέριση των δεδομένων με single dimensional τρόπο. Θεωρούμε ότι το k=2. Η διακεκομμένη γραμμή δείχνει πως ακριβώς θα γινόταν μια δεύτερη διαμέριση των δεδομένων αν ήταν επιτρεπτή. Επειδή αν γίνει η δεύτερη διαμέριση καταλήγουμε σε ομάδες που περιέχουν δεδομένα που έχουν λιγότερα από k στοιχεία, η διαμέριση δεν είναι επιτρεπτή. Τέλος, στο Σχήμα 2.3 απεικονίζεται μια διαμέριση των στοιχείων με multi dimensional τρόπο. Εδώ ακριβώς βλέπουμε και τις διαφορές που έχει η multi dimensional προσέγγιση από τη single dimensional. Κάθε διαμέριση του χώρου στη single dimensional δημιουργεί δυο ξεχωριστές ομάδες δεδομένων αλλά η επόμενη διαμέριση επηρεάζει με ομοιόμορφο τρόπο και τις δυο ομάδες. Αντίθετα, στη multi dimensional προσέγγιση μια διαμέριση του χώρου αφήνει δυο ανεξάρτητους υποχώρους εκ των οποίων ο καθένας μπορεί να διαμεριστεί εκ νέου

ανεξάρτητα από το αν μια πιθανή τομή επιτρέπεται στον άλλο ή όχι. Καταλαβαίνουμε δηλαδή ότι το multi dimensional recoding είναι αρκετά πιο ακριβές ενώ παράλληλα το single dimensional recoding είναι αρκετά πιο απλό.

Συνδυάζοντας τώρα τις παραπάνω κατηγορίες, δημιουργούμε ορισμένες παραλλαγές στους παραπάνω μετασχηματισμούς. Αρχικά, το local recoding εκτελείται εξ ορισμού πολυδιάστατα. Το global recoding, όμως, μπορεί να εκτελεστεί είτε με μονοδιάστατο είτε με πολυδιάστατο τρόπο. Στο παράδειγμα που παραθέτουμε[5], δίνουμε έναν τυχαίο πίνακα δεδομένων και εκτελούμε recoding σε αυτόν με τρεις διαφορετικούς τρόπους: global recoding, multi dimensional global recoding και local recoding. Τα δεδομένα που χρησιμοποιήθηκαν παρατίθενται στο γράφημα (a). Στο επάνω μέρος του γραφήματος βλέπουμε την οργάνωση των δεδομένων στο δισδιάστατο επίπεδο και στο κάτω μέρος παρουσιάζουμε τα δεδομένα σε μορφή πίνακα.

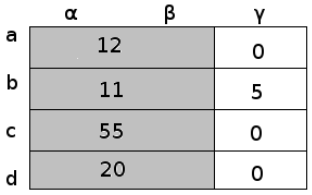
	α	β	γ
a	2	10	0
b	8	3	5
c	30	25	0
d	0	20	0

Att1	**Att2**	**Freq**
a	α	2
a	β	10
b	α	8
b	β	3
b	γ	5
c	α	30
c	β	25
d	β	20

(a)

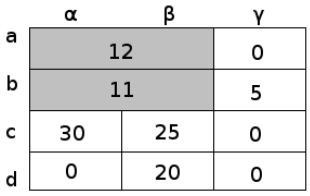
Εξετάζουμε τα δεδομένα ως προς 2 attributes. Το Att1 που έχει πεδίο τιμών το $\{a, b, c, d\}$ και το Att2 που έχει πεδίο τιμών το $\{\alpha, \beta, \gamma\}$. Θεωρούμε ότι τα πεδία τιμών είναι διακριτά και μπορούν να υπάρχουν πολλά σημεία-tuples στις ίδιες περιοχές. Το πεδίο Freq στον πίνακα δείχνει πόσα σημεία υπάρχουν με τα αντίστοιχα attributes και ο ίδιος αριθμός παρουσιάζεται στη δισδιάστατη απεικόνιση και δείχνει πόσα σημεία βρίσκονται στο αντίστοιχο τμήμα. Επάνω στα δεδομένα εφαρμόζουμε recoding θεωρώντας ότι k=5.

[5]Το παράδειγμα που παραθέτουμε βρίσκεται στο άρθρο [5].



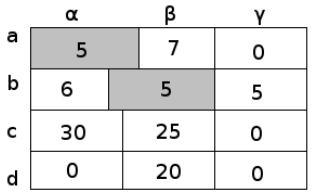
Att1	Att2	Freq
a	(α,β)	12
b	(α,β)	11
b	γ	5
c	(α,β)	55
d	(α,β)	20

(b)



Att1	Att2	Freq
a	(α,β)	12
b	(α,β)	11
b	γ	5
c	α	30
c	β	25
d	β	20

(c)



Att1	Att2	Freq
a	(α,β)	5
a	β	7
b	α	6
b	(α,β)	5
b	γ	5
c	α	30
c	β	25
d	β	20

(d)

Στα γραφήματα (b), (c), (d) φαίνεται η ομαδοποίηση των δεδομένων μετά το recoding. Στο (b) εφαρμόσαμε global recoding, στο (c) global multi dimensional recoding και στο (d) εφαρμόσαμε local recoding. Βλέπουμε ότι:

- Στο (b), υπάρχουν τέσσερις τομές όπως φαίνεται και από τις γραμμές που οριοθετούν τα τμήματα (τρεις ως προς τον άξονα Att1 και μία ως προς το Attr2) εκ των οποίων, τα τρία τμήματα δεν περιέχουν δεδομένα. Συνεπώς, σε αυτήν την περίπτωση έχουμε 5 ομάδες δεδομένων στις οποίες περιέχονται πολλά tuples.

- Στο (c), εισάγουμε μια multi dimensional προσέγγιση. Ο αριθμός των τομών αυξάνεται κατά πολύ (συνολικά έχουμε 8 τομές και οι πιθανοί συνδυασμοί των τομών είναι αρκετοί) πράγμα που σημαίνει ότι οι ομάδες αυξάνονται σε αριθμό. Πράγματι βλέπουμε ότι σε αυτήν την περίπτωση δημιουργούνται δέκα ομάδες δεδομένων εκ των οποίων οι τέσσερις δεν περιέχουν δεδομένα. Βλέπουμε επίσης ότι συγκριτικά με το (b) κάποιες ομάδες που περιείχαν πολλά tuples έχουν διαιρεθεί σε αυτήν την περίπτωση. Οι μόνες ομάδες που πρέπει αναγκαστικά να συνενωθούν για να ικανοποιούν τον περιορισμό k=5 σημειώνονται με γκρι χρώμα.

- Τέλος στο (d), εφαρμόζουμε local recoding. Σε αυτήν την προσέγγιση δεν ακολουθούμε τη λογική των τομών όπως στις δυο προηγούμενες περιπτώσεις, αλλά στοχεύουμε στη διαίρεση ακόμα και τις ίδιας ομάδας έτσι ώστε να δημιουργήσουμε όσες περισσότερες ομάδες μπορούμε. Πράγματι, αν συγκρίνουμε με το (c), η ομάδα {a,{α,β}} που απαρτιζόταν από 12 tuples διαιρέθηκε έτσι ώστε να δημιουργηθούν δύο ομάδες με την καθεμία να έχει τουλάχιστον k tuples. Επειδή όμως τα {a,α} στοιχεία ήταν μόνο 2 όπως μπορούμε να δούμε και από το (a), "μεταφέραμε" 3 στοιχεία από

το {a,β} και δημιουργήθηκε η ομάδα {a,{α,β}}. Με ακριβώς παρόμοιο τρόπο διαχειριστήκαμε την ομάδα {b,{α,β}} της περίπτωσης (c). Τελικά, βλέπουμε ότι με αυτόν τον τρόπο έχουμε το μεγαλύτερο αριθμό ομάδων (12 συνολικά, εκ των οποίων οι 8 περιέχουν tuples) και κάθε ομάδα περιέχει το μικρότερο δυνατό αριθμό από tuples.

Συμπερασματικά, από το παραπάνω παράδειγμα βλέπουμε για ποιο λόγο το local recoding θεωρείται αποδοτικότερο από το global recoding. Εποπτικά, μπορούμε να πούμε ότι ενώ με το global recoding στόχος μας είναι να δημιουργήσουμε όσο μικρότερες διαμερίσεις του χώρου μπορούμε, με το local recoding μπορούμε να "μετονομάσουμε" ορισμένα tuples έτσι ώστε να συμμετέχουν σε διαφορετικές ομάδες. Δεδομένου ότι στόχος μας είναι η δημιουργία πολλών (αριθμητικά) και μικρών (ως προς τα στοιχεία που περιέχουν) ομάδων καταλαβαίνουμε ότι το local recoding προτιμάται από το global recoding ενώ ακόμη, η multi dimensional προσέγγιση προτιμάται της single dimensional.

Στην παρούσα εργασία θα χρησιμοποιήσουμε αλγορίθμους που εκτελούν local recoding. Στο επόμενο κεφάλαιο θα αναλύσουμε τη λειτουργία τους και στο Κεφάλαιο 4 θα παραθέσουμε τα αποτελέσματα από τις εκτελέσεις των αλγορίθμων αυτών επάνω σε πραγματικά και τεχνητά δεδομένα.

2.3 Κατανεμημένη Αρχιτεκτονική

Όπως αναφέραμε και στο εισαγωγικό κεφάλαιο, σκοπός της εργασίας είναι η ανάπτυξη μιας κατανεμημένης μεθόδου για την ανωνυμοποίηση μεγάλου όγκου δεδομένων. Στις προηγούμενες ενότητες κάναμε μια εισαγωγή στην έννοια της ανωνυμοποίησης και στο μετασχηματισμό γενίκευσης με τον οποίο επιτυγχάνεται. Στις ενότητες που ακολουθούν θα παρουσιάσουμε το προγραμματιστικό μοντέλο MapReduce[7] που μας επιτρέπει την εκτέλεση κατανεμημένων προγραμμάτων σε ένα cluster υπολογιστών και το Hadoop[8], που αποτελεί μια open source υλοποίηση του MapReduce από την εταιρεία Apache Foundation.

2.3.1 MapReduce

Το MapReduce είναι ένα προγραμματιστικό μοντέλο και το όνομα μιας υλοποίησης του μοντέλου αυτού με στόχο την επεξεργασία αλλά και τη δημιουργία μεγάλων data sets. Με τον όρο data set, αναφερόμαστε σε ένα σύνολο δεδομένων το οποίο θεωρούμε ότι είναι αρκετά μεγάλο για να μπορεί να επεξεργαστεί αποτελεσματικά από έναν υπολογιστή. Το σύνολο αυτό θα έχει στοιχεία (tuples) και κάθε στοιχείο θα έχει γνωρίσματα (attributes). Θα μπορούσαμε να παρομοιάσουμε ένα data set σαν ένα πολύ μεγάλο πίνακα δεδομένων όπως αυτός του Πίνακα 2.1. Επίσης έχει πάρει το όνομά του από τις συναρτήσεις map και reduce που συναντώνται στον συναρτησιακό προγραμματισμό, παρόλο που ο σκοπός των λειτουργιών map και reduce στο προγραμματιστικό αυτό μοντέλο είναι διαφορετικός από το σκοπό των αντίστοιχων συναρτήσεων στο συναρτησιακό μοντέλο.

Το MapReduce είναι ένα προγραμματιστικό μοντέλο με το οποίο μπορούμε να επιλύσουμε embarrassingly parallel προβλήματα με είσοδο πολύ μεγάλα data sets με τη βοήθεια ενός μεγάλου αριθμού υπολογιστών[6]. Με τον όρο embarrassingly parallel πρόβλημα εννοούμε κάποιο πρόβλημα το οποίο μπορεί εύκολα να χωριστεί σε ένα σύνολο παράλληλων εργασιών για την επίλυσή του. Τέτοιες περιπτώσεις προβλημάτων έχουμε όταν αυτές οι παράλληλες εργασίες δεν έχουν καμία σχέση εξάρτησης ή επικοινωνίας μεταξύ τους. Χαρακτηριστικά παραδείγματα embarrassingly parallel προβλημάτων είναι: η απεικόνιση (rendering) των γραφικών των υπολογιστών (κάθε pixel μπορεί να απεικονιστεί ξεχωριστά), η εξυπηρέτηση στατικών σελίδων από ένα web server (αναφέρουμε μόνο τη στατική γιατί στη δυναμική εξυπηρέτηση μπορεί να υπάρχει αλληλεπίδραση μεταξύ διαφορετικών σελίδων), κοκ.

Η υπολογιστική διαδικασία δέχεται σαν είσοδο ένα σύνολο ζευγαριών κλειδιών/τιμών (key/value pairs) και έχει σαν έξοδο ένα σύνολο key/values ζευγάρια. Ο προγραμματιστής μπορεί να επέμβει στη διαδικασία μέσω των συναρτήσεων Map και Reduce. Συγκεκριμένα:

- Η συνάρτηση **Map** δέχεται ως είσοδο ένα ζευγάρι key/value και παράγει ένα ζευγάρι από ένα ενδιάμεσο κλειδί και μια τιμή. Η βιβλιοθήκη του MapReduce συγκεντρώνει όλες τις τιμές που έχουν το ίδιο ενδιάμεσο κλειδί και τις προωθεί στη συνάρτηση Reduce.

- Η συνάρτηση **Reduce** δέχεται ως είσοδο ένα ενδιάμεσο κλειδί που παράχθηκε από τη συνάρτηση Map και ένα σύνολο από τιμές (values) αυτού του κλειδιού. Στη συνέχεια αφού γίνουν οι απαραίτητοι υπολογισμοί, επιστρέφει συνήθως μια ή καμία έξοδο (όπου έξοδος είναι ένα ζευγάρι key/value).

Ο προγραμματιστής, ανάλογα με τον υπολογισμό που θέλει να εκτελέσει γράφει τις παραπάνω συναρτήσεις, τηρώντας όμως το interface των συναρτήσεων:

$$\begin{aligned} map \quad & (k1, v1) \quad \rightarrow list(k2, v2) \\ reduce \quad & (k2, list(v2)) \rightarrow list(v2) \end{aligned}$$

Στην πράξη, το προγραμματιστικό μοντέλο MapReduce έχει μεγάλη εκφραστικότητα, αφού ένα μεγάλο πλήθος προβλημάτων μπορεί να επιλυθεί πολύ εύκολα με τη βοήθειά του. Ακόμη, πέρα από εκφραστικότητα το MapReduce παρέχει αρκετά μεγάλο fault-tolerance ενώ παράλληλα παρέχει μεγάλη κλιμακωσιμότητα (scalability). Στην επόμενη ενότητα θα εξετάσουμε τον ακριβή τρόπο λειτουργίας του Hadoop, που αποτελεί την υλοποίηση του MapReduce που χρησιμοποιήσαμε στην παρούσα εργασία.

2.3.2 Hadoop

Το Hadoop είναι ένα από τα μεγαλύτερα open source frameworks που υποστηρίζουν την εκτέλεση κατανεμημένων εφαρμογών. Στην ουσία, αποτελείται από ένα σύνολο υποσυ-

[6]Στο εξής, το σύνολο των υπολογιστών τους οποίους χρησιμοποιούμε με τη βοήθεια του MapReduce θα τους αποκαλούμε cluster υπολογιστών.

στημάτων που συνδυάζουν τη λειτουργία τους αρμονικά για την εκτέλεση κατανεμημένων διεργασιών. Πυρήνας του Hadoop είναι το Hadoop MapReduce, το υποσύστημα δηλαδή εκείνο που υλοποιεί το προγραμματιστικό μοντέλο που περιγράφηκε στην προηγούμενη ενότητα. Προϋπόθεση για την λειτουργία του MapReduce είναι η ύπαρξη ενός κατανεμημένου συστήματος αρχείων. Για το λόγο αυτό, χρησιμοποιείται το Hadoop Distributed File System (HDFS).

Το HDFS έχει αρχιτεκτονική master/slave. Στο HDFS cluster υπάρχει ένας μοναδικός NameNode, ένας master κόμβος δηλαδή που διαχειρίζεται το namespace του filesystem και δίνει πρόσβαση στους clients. Παράλληλα, στο cluster υπάρχουν και τα DataNodes, συνήθως ένα DataNode ανά κόμβο (συμπεριλαμβανομένου και του master κόμβου), που είναι υπεύθυνοι για την αποθήκευση των δεδομένων στα συστήματα στα οποία τρέχουν. Οι DataNodes δηλαδή είναι υπεύθυνοι για την τοπική αποθήκευση δεδομένων στον υπολογιστή που τρέχουν και ο NameNode είναι αυτός που επικοινωνεί με τους DataNodes, διαβάζει τα δεδομένα από αυτούς, δημιουργεί την εντύπωση ότι ο αποθηκευτικός χώρος του cluster είναι ενιαίος, είναι υπεύθυνος για τη δημιουργία αντιγράφων και ελέγχει σε πολύ τακτά χρονικά διαστήματα αν οι DataNodes εκτελούνται.

Το HDFS γράφτηκε για να υποστηρίζει πολύ μεγάλα αρχεία και κατ' επέκταση, γράφτηκε για να τρέχει σε πολύ μεγάλα cluster. Οι υπολογιστές σε πολύ μεγάλα cluster, οργανώνονται σε racks. Αυτό σημαίνει ότι δυο υπολογιστές σε διαφορετικά racks για να επικοινωνήσουν, πρέπει να μεταδώσουν πληροφορίες μέσω ενός switch. Αυτό προφανώς εισάγει καθυστέρηση στην επικοινωνία μεταξύ υπολογιστών και κατ' επέκταση, επιβραδύνει την λειτουργία του filesystem. Παράλληλα όμως, σε ένα μεγάλο cluster υπάρχει πάντα πιθανότητα ένας κόμβος να σταματήσει κάποια στιγμή να δουλεύει. Για το λόγο αυτό, το HDFS δημιουργεί αντίγραφα, τα οποία αποθηκεύονται, ανάλογα με τον αριθμό των αντιγράφων, είτε σε ένα είτε σε διαφορετικά racks[7]. Αν και αυτό εισάγει καθυστέρηση στην αποθήκευση των δεδομένων, εξασφαλίζει ότι σε περίπτωση που κάποιο rack βγει εκτός λειτουργίας (γιατί για παράδειγμα, κάτι συνέβη στο switch) τότε και πάλι δεν θα χάσουμε τα δεδομένα, αφού μπορούμε να τα ανακτήσουμε από κόμβο που ανήκει σε άλλο rack. Η ιδιότητα αυτή του HDFS, η ικανότητα δηλαδή του filesystem να γνωρίζει σε ποιο rack ανήκει ο υπολογιστής στον οποίο έχει αποθηκευτεί ένα block ή ένα αρχείο δεδομένων (η ιδιότητα αυτή καλείται και rack awareness), παίζει σημαντικό ρόλο στη λειτουργία του MapReduce όπως θα δούμε και στη συνέχεια.

Πάνω ακριβώς από το HDFS, λειτουργεί η MapReduce Engine. Όπως το HDFS, έτσι και το MapReduce έχει αρχιτεκτονική master/slave. Στον master κόμβο του cluster εκτελείται ο JobTracker (μοναδικός σε όλο το cluster), στον οποίο υποβάλλονται τα MapReduce jobs από τον client. Σε όλους τους υπολογιστές του cluster τρέχουν πα-

[7]Ο προεπιλεγμένος αριθμός αντιγράφων που δημιουργεί το HDFS είναι 3. Σε αυτήν την περίπτωση, 2 αντίγραφα του αρχείου αποθηκεύονται σε υπολογιστές του ίδιου rack και το 3ο αντίγραφο αποθηκεύεται σε υπολογιστή άλλου rack.

ράλληλα και οι TaskTrackers (συμπεριλαμβανομένου και το master) οι οποίοι είναι υπεύθυνοι για την εκτέλεση συγκεκριμένων εργασιών (tasks). Ως MapReduce job μπορούμε να θεωρήσουμε ένα ολοκληρωμένο κύκλο εκτέλεσης της συνάρτησης[8] map (ενδεχομένως περισσότερες από μια εκτελέσεις) και στη συνέχεια της reduce (ενδεχομένως περισσότερες από μια εκτελέσεις, επίσης) όπως αυτές αναλύθηκαν στην προηγούμενη ενότητα. Αντίστοιχα, μια εκτέλεση της map ή της reduce θεωρούμε ότι είναι ένα task. Κάθε διαφορετική εκτέλεση της map ή της reduce γίνεται για διαφορετικά δεδομένα.

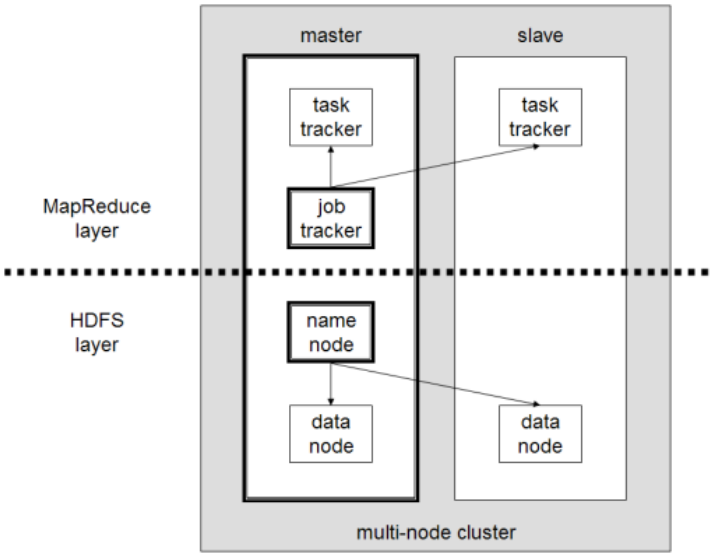
Ο JobTracker είναι υπεύθυνος για την εκτέλεση ενός job. Επικοινωνεί με τους Task Trackers και στέλνει τα tasks που πρέπει να γίνουν για την εκτέλεση του job. Ο τρόπος με τον οποίο οι slaves αναλαμβάνουν tasks είναι πολύ απλός: κάθε TaskTracker έχει ένα συγκεκριμένο αριθμό από διαθέσιμα slots. Κάθε task καταλαμβάνει ένα slot. Σε περίπτωση που γεμίσουν όλα τα slots όλων των TaskTrackers, υπάρχει μια ουρά αναμονής (την οποία τηρεί ο JobTracker) στην οποία τοποθετούνται τα tasks που περιμένουν να εκτελεστούν (με προτεραιότητα First In First Out). Με την ολοκλήρωση όλων των tasks, ο JobTracker εκτελεί κάποιες τελευταίες ενέργειες και στη συνέχεια το job ολοκληρώνεται.

Ένα πολύ σημαντικό σημείο στην επικοινωνία μεταξύ JobTracker και TaskTrackers αφορά τον τρόπο με τον οποίο ο JobTracker αποφασίζει ποιο task θα αναλάβει κάποιος TaskTracker. Επειδή κάθε task εκτελείται για διαφορετικά δεδομένα, ο JobTracker πρέπει να φροντίσει έτσι ώστε το task που θα αναθέσει σε κάποιο κόμβο να εκτελείται για δεδομένα που υπάρχουν είτε στον τοπικό σκληρό δίσκου του κόμβου είτε στον σκληρό δίσκο κόμβου που βρίσκεται στο ίδιο rack με τον κόμβο αυτό. Σε περίπτωση που τα tasks κατανέμονταν τυχαία μέσα στο cluster, θα είχαμε μεγάλες καθυστερήσεις εξαιτίας του δικτύου. Εδώ ακριβώς φαίνεται και η σημασία του rack-awareness του HDFS που είδαμε προηγουμένως. Ο JobTracker μπορεί να κατανέμει τα tasks στο cluster με γνώμονα τη φυσική θέση των δεδομένων στα nodes ελαχιστοποιώντας έτσι τις καθυστερήσεις εξαιτίας των δικτυακών συσκευών.

Συνοψίζοντας τις λειτουργίες των HDFS και MapReduce, παραθέτουμε ένα πολύ απλό Σχήμα στο οποίο φαίνεται το στιγμιότυπο ενός μικρού cluster που αποτελείται από έναν slave και έναν master (Σχήμα 2.4). Στο σχήμα αυτό φαίνεται η ομοιότητα που υπάρχει στην αρχιτεκτονική του HDFS με το MapReduce (master/slave όπως είδαμε). Επίσης, μπορούμε να δούμε πως επικοινωνεί ο JobTracker με τους TaskTrackers και ο NameNode με τους DataNodes.

Μια παρατήρηση επάνω στο Σχήμα 2.4 αφορά την ύπαρξη μοναδικού "καθολικού" master στο cluster. Στο σχήμα αυτό, φαίνεται ότι ένας κόμβος του cluster αναλαμβάνει ρόλο NameNode και JobTracker ταυτόχρονα. Στην πράξη, κάτι τέτοιο αν και συνηθίζεται δεν είναι απαραίτητο. Σε πολλές περιπτώσεις, για λόγους απόδοσης ο JobTracker τρέχει σε διαφορετικό κόμβο από τον NameNode. Για παράδειγμα, σε περίπτωση που το cluster είναι

[8]Ο όρος συνάρτηση εδώ, δεν χρησιμοποιείται με την προγραμματιστική του έννοια, αλλά με την έννοια που χρησιμοποιήθηκε στην ενότητα 2.3.1.



Σχήμα 2.4: Hadoop cluster

αρκετά μεγάλο και αποτελείται από χιλιάδες HDFS αρχεία, ο NameNode για να εκτελεστεί χρειάζεται πολλούς πόρους. Σε αυτήν την περίπτωση, είναι προτιμότερο να εκτελούνται σε διαφορετικούς κόμβους γιατί με αυτό τον τρόπο, απορροφούν του πόρους που χρειάζονται χωρίς να τους διεκδικούν.

2.4 Υπάρχουσες κατανεμημένες μέθοδοι ανωνυμοποίησης

Στην ενότητα αυτή, θα κάνουμε μια πολύ σύντομη ανασκόπηση σε προσπάθειες που έχουν γίνει στο παρελθόν για την επίτευξη μια κατανεμημένης μεθόδου ανωνυμοποίησης. Αρχικά, η ανωνυμοποίηση των δεδομένων μπορούσε να επιτευχθεί μόνο κεντρικό τρόπο. Ένας server ήταν υπεύθυνος για τη συλλογή των δεδομένων, την ανωνυμοποίησή τους και στη συνέχεια την αποστολή των δεδομένων στους χρήστες. Στη συνέχεια, κάποιες εργασίες που πρότειναν ένα κατανεμημένο μοντέλο για την ανωνυμοποίηση των δεδομένων ([9], [10]) δεν μπόρεσαν να διαχειριστούν δεδομένα τα οποία ήταν κατανεμημένα σε ένα σύνολο από κόμβους. Τέλος, ένα πλήρως κατανεμημένο σύστημα το οποίο στηρίζεται στις ιεραρχίες των δεδομένων για την επίτευξη του k-anonymity είναι το σύστημα KANIS [11].

Το KANIS είναι ένα κατανεμημένο σύστημα το οποίο στηρίζεται στο πρωτόκολλο peer to peer και στο DHT (Distributed Hash Table) για την κατανομή των δεδομένων στους κόμβους. Μέρος του συστήματος είναι η εκτέλεση του αλγορίθμου Incognito [4], ένας αλγόριθμος ο οποίος για την εκτέλεση της ανωνυμοποίησης στηρίζεται σε ιεραρχίες δεδομένων και εκτελεί global recoding, σε αντίθεση με την παρούσα εργασία στην οποία χρησιμοποιήθηκε το σύστημα Hadoop, οι αλγόριθμοι ανωνυμοποίησης εκτελούν local recoding και στηρίζονται στη διαμέριση των δεδομένων.

Κεφάλαιο 3

Υλοποίηση κατανεμημένης εφαρμογής

Στο κεφάλαιο αυτό, θα αναλύσουμε τις τεχνικές ανωνυμοποίησης που χρησιμοποιήσαμε στην παρούσα εργασία και θα μελετήσουμε τον τρόπο με τον οποίο μετατρέψαμε τις μεθόδους ώστε να τρέχουν με κατανεμημένο τρόπο. Θα παρουσιάσουμε τους αλγορίθμους σε επίπεδο ψευδοκώδικα, θα αναπαραστήσουμε με γραφικό τρόπο την εκτέλεσή τους, θα μελετήσουμε την πολυπλοκότητά τους και στο τέλος θα παρουσιάσουμε και τους τρόπους με τους οποίους μπορούμε να μετατρέψουμε την εκτέλεση τους από κεντρική σε κατανεμημένη.

3.1 Μέθοδοι ανωνυμοποίησης

Όπως αναφέραμε και στις προηγούμενες ενότητες, η βέλτιστη ανωνυμοποίηση ενός πίνακα δεδομένων ως προς κάποια συνάρτηση κόστους ανήκει στην κατηγορία των NP-complete προβλημάτων, δηλαδή δεν έχει βρεθεί κάποιος αλγόριθμος πολυωνυμικού χρόνου που να επιλύει το πρόβλημα βέλτιστα. Παρ'όλα αυτά, έχουν αναπτυχθεί διάφοροι greedy αλγόριθμοι πολυωνυμικού χρόνου που επιλύουν το πρόβλημα της ανωνυμοποίησης με ικανοποιητικό τρόπο. Μια υποκατηγορία των άπληστων αυτών αλγορίθμων, είναι οι αλγόριθμοι που στηρίζονται στη συνεχή διαμέριση του χώρου του προβλήματος. Οι αλγόριθμοι αυτοί σε κάθε βήμα εκτέλεσης τους διαιρούν το σύνολο των tuples σε 2 ή περισσότερα υποσύνολα. Στη συνέχεια, εκτελούν την ίδια διαδικασία αναδρομικά μέχρι να εκπληρωθεί κάποιο κριτήριο τερματισμού[1]. Στο τέλος, ο αλγόριθμος έχει διαμερίσει το αρχικό σύνολο σε πολλά υποσύνολα που το καθένα έχει τουλάχιστον k tuples (έτσι ώστε να ικανοποιείται το k-anonymity). Ο τελικός πίνακας δεδομένων θα περιέχει τιμές που θα είναι ίδιες για tuples που ανήκουν στο ίδιο υποσύνολο και ίσες με το εύρος των τιμών του υποσυνόλου

[1]Συνήθως, το κριτήριο τερματισμού έχει άμεση σχέση με τον αριθμό των tuples που βρίσκονται στο σύνολο που είναι προς διαμέριση.

στο αντίστοιχο attribute. Σε μορφή ψευδοκώδικα, η παραπάνω διαδικασία θα μπορούσε να εκφραστεί ως εξής:

```
1 k        //αριθμός του k−anonymity
2 partition(Set data)
3   if (|data|<2k)
4     return ∅
5   else
6     part1, part2 = split(data)
7     return partition(part1) ∪ partition(part2)
```

Η συνάρτηση split θεωρούμε ότι είναι μια συνάρτηση που δέχεται ως παράμετρο το αρχικό σύνολο (πίνακα δεδομένων - data) και επιστρέφει δυο υποσύνολα (part1, part2) που η ένωσή τους είναι το αρχικό σύνολο ενώ παράλληλα τα part1 και part2 είναι ξένα μεταξύ τους. Δεν διευκρινίζουμε περαιτέρω τη λειτουργία της split γιατί κάθε αλγόριθμος διαμερίζει με διαφορετικό τρόπο το σύνολο. Επίσης, η συνθήκη τερματισμού της αναδρομής είναι η συνθήκη $|data| < 2k$. Η συνθήκη αυτή είναι λογική, αφού αν ένα σύνολο διαμεριστεί θα δημιουργηθούν δύο περίπου ίσα υποσύνολα, τα οποία πρέπει να αποτελούνται από τουλάχιστον k στοιχεία το καθένα. Συνεπώς, το σύνολο από το οποίο θα προκύψουν πρέπει να περιέχει τουλάχιστον 2k στοιχεία.

Από άποψη χρονικής πολυπλοκότητας, η συνεχής διαμέριση μας εξασφαλίζει ότι σε κάθε βήμα του αλγορίθμου το νέο σύνολο που παράγεται θα είναι περίπου το μισό σε σχέση με το σύνολο από το οποίο προήλθε. Δηλαδή, κάθε νέα κλήση της partition θα διαχειρίζεται ένα σύνολο data που θα μειώνεται με εκθετικά (αφού συνεχώς θα υποδιπλασιάζεται). Ισοδύναμα, μπορούμε να πούμε ότι η πολυπλοκότητα του αλγορίθμου θα αυξάνεται λογαριθμικά ως προς το μέγεθος του συνόλου. Η τελική συνολική πολυπλοκότητα του αλγορίθμου θα δίνεται από την πολυπλοκότητα της split επί εναν παράγοντα $\log n - \log k$ (αν θεωρήσουμε ότι ο αρχικός πίνακας δεδομένων περιέχει n tuples) ή ισοδύναμα $\log \frac{n}{k}$. Στη συνέχεια θα μελετήσουμε με μεγαλύτερη ακρίβεια την πολυπλοκότητα των αλγορίθμων αφού θα ορίσουμε ακριβώς και τη συνάρτηση split σε κάθε αλγόριθμο.

3.1.1 Mondrian

Ο αλγόριθμος Mondrian[2] είναι ένας greedy αλγόριθμος που η λειτουργία του στηρίζεται στη συνεχή διαμέριση του χώρου του προβλήματος, όπως περιγράψαμε παραπάνω. Ο αλγόριθμος, σε μορφή ψευδοκώδικα έχει την ακόλουθη μορφή:

```
1 k        //αριθμός k−anonymity
2 mondrian(partition)
3   if (|partition|<2k)
4     return ∅
5   else
6     dim = chooseDimension(partition)
7     splitValue = findMedian(partition,dim)
```

```
8       lPart = {t ∈ partition : t.dim ≤ splitValue}
9       rPart = {t ∈ partition : t.dim > splitValue}
10      return mondrian(lPart) ∪ mondrian(rPart)
```

Οι γραμμές 6-10 του ψευδοκώδικα αποτελούν τη συνάρτηση split που περιγράψαμε στο γενικό αλγόριθμο διαμέρισης. Όπως βλέπουμε, στην αρχή επιλέγεται μια διάσταση του πίνακα. Η διάσταση αυτή ισοδυναμεί με κάποιο attribute των tuples και ανήκει στο QID βάση του οποίου εκτελείται η ανωνυμοποίηση. Στη συνέχεια, εξετάζουμε όλα τα tuples στο attribute που επιλέχθηκε και επιλέγουμε την τιμή του ενδιάμεσου attribute (splitValue). Για παράδειγμα, αν τα tuples στο attribute που είναι προς εξέταση έχουν τις τιμές $\{10, 11, 9, 10, 2\}$, η ενδιάμεση τιμή είναι το 10[2].

Στη συνέχεια, χωρίζουμε τα tuples με βάση την τιμή που έχουν στο υποεξέταση attribute και το splitValue. Τα tuples που είναι μεγαλύτερα από το ενδιάμεσο στοιχείο στην επιλεγμένη διάσταση, τοποθετούνται στο rPart ενώ όλα τα υπόλοιπα τοποθετούνται στο lPart. Τέλος, η διαδικασία συνεχίζεται αναδρομικά, μέχρι όλα τα υποσύνολα να παραμείνουν με λιγότερα από $2k$ στοιχεία (έτσι ώστε να μην διαμερίζονται).

Ένα πολύ σημαντικό τμήμα του αλγορίθμου αφορά τον τρόπο με τον οποίο επιλέγεται η διάσταση ως προς την οποία θα γίνει η διαμέριση. Γενικά, μπορούν να χρησιμοποιηθούν πολλοί ευριστικοί τρόποι για την επιλογή της διάστασης. Ένας απλός και αποδοτικός τρόπος που επιλέχθηκε στην εργασία αυτή, είναι η επιλογή της διάστασης που έχει το μέγιστο κανονικοποιημένο εύρος τιμών. Για κάθε διάσταση (ή αλλιώς, για κάθε attribute που ανήκει στο QID) υπολογίζεται το εύρος τιμών του εξεταζόμενου partition και στη συνέχεια, το εύρος αυτό διαιρείται με τη μέγιστη τιμή που εμφανίζει το attribute σε όλα τα tuples του πίνακα δεδομένων (έτσι τα εύρη τιμών όλων των διαστάσεων κανονικοποιούνται και έχουν μέγιστη τιμή ίση με 1). Από όλα τα attributes επιλέγεται αυτό που έχει το μεγαλύτερο κανονικοποιημένο εύρος. Σε περίπτωση που όλα τα attributes έχουν το ίδιο κανονικοποιημένο εύρος (πχ, στο πρώτο βήμα εκτέλεσης του αλγορίθμου) τότε επιλέγεται το attribute που έχει το μικρότερο απόλυτο εύρος.

Μετά την επιλογή της διάστασης ακολουθεί η εύρεση του ενδιάμεσου στοιχείου και στη συνέχεια γίνεται η διαμέριση των tuples με βάση το στοιχείο αυτό. Γενικά, ένας αλγόριθμος που στηρίζεται στη διαμέριση όπως ο Mondrian μπορεί να ακολουθήσει ορισμένες πολιτικές κατά τη διαμέριση ενός συνόλου. Μια πιθανή πολιτική είναι η διαμέριση ενός συνόλου με στόχο τη δημιουργία δυο συνόλων που περιέχουν περίπου τον ίδιο αριθμό στοιχείων, ακόμη και αν αυτό σημαίνει ότι πολλά σημεία που βρίσκονται εκατέρωθεν του ενδιάμεσου στοιχείου τοποθετούνται στο ίδιο υποσύνολο. Μια δεύτερη πιθανή πολιτική είναι η ακριβώς αντίθετη της πρώτης: κατά τη διαμέριση τα στοιχεία τοποθετούνται "αυστηρά" εκεί που θα έπρεπε να ανήκουν με βάση την θέση τους ως προς το ενδιάμεσο στοιχείο ακόμα και αν αυτό σημαίνει ότι θα δημιουργηθούν δυο σύνολα που θα έχουν μεγάλη διαφορά στο αριθμό των

[2]Αν δηλαδή, ταξινομήσουμε τα tuples ως προς το attribute που εξετάζουμε, το ενδιάμεσο στοιχείο είναι το στοιχείο που βρίσκεται στη μέση του ταξινομημένου συνόλου. Στο παράδειγμα που αναφέρουμε το ταξινομημένο σύνολο θα είναι το $\{2, 9, 10, 10, 11\}$ και η μεσαία τιμή είναι το 10.

στοιχείων τους. Η πρώτη πολιτική διαμέρισης είναι γνωστή ως relaxed parititioning και η δεύτερη είναι γνωστή ως strict partitioning.

Το relaxed partitioning σε σύγκριση με το strict partitioning, πλεονεκτεί στον ταχύτερο χρόνο εκτέλεσης και στην ισοκατανομή του υπολογιστικού φόρτου με ομοιόμορφο τρόπο. Ως προς το χρόνο εκτέλεσης, με το relaxed partitioning γνωρίζουμε ότι το δέντρο των αναδρομών θα έχει ακριβώς $\log n$ επίπεδα, αφού σε κάθε βήμα του αλγορίθμου τα εξεταζόμενα σύνολα υποδιαιρούνται σε δυο ίσα υποσύνολα. Αντίθετα, το strict partitioning δεν προσφέρει καμία εγγύηση ότι τα παραγόμενα υποσύνολα θα είναι ίσα, συνεπώς το δέντρο των αναδρομών δεν θα έχει απαραίτητα $\log n$ επίπεδα, αλλά θα έχει στη μέση περίπτωση $\log_a n$ όπου $a \in (1, 2]$ και στη χειρότερη περίπτωση θα έχει n επίπεδα [3]. Επίσης, με το relaxed partitioning γνωρίζουμε ότι κάθε αναδρομική κλήση του ίδιου επιπέδου θα κάνει περίπου τον ίδιο χρόνο να εκτελεστεί διότι έχει το ίδιο μέγεθος εισόδου. Αντίθετα, με το strict partitioning δεν ισχύει κάτι τέτοιο, αφού οι εκτελέσεις στο ίδιο επίπεδο (στο δέντρο αναδρομών) μπορεί να έχουν πολύ μεγάλη διαφορά στο μέγεθος εισόδου.

Το strict partitioning παρόλα αυτά, υπερτερεί του relaxed σε θέματα απόδοσης. Αυτό είναι λογικό, από τη στιγμή που με το strict partitioning η διαμέριση γίνεται με στόχο τα tuples της ίδιας υποομάδας να είναι αρκετά κοντινά στην επιλεγόμενη διάσταση, ενώ με το relaxed ο πρωταρχικός στόχος είναι η ισοκατανομή των tuples και δευτερευόντως η μείωση της απόστασης των tuples της ίδιας ομάδας. Στη συνέχεια, θα εξετάσουμε την απόδοση του αλγορίθμου Mondrian όταν χρησιμοποιεί και τις δυο πολιτικές διαμέρισης.

Τέλος, ένα σημαντικό ζήτημα στη λειτουργία του αλγορίθμου όταν εκτελεί strict partitioning, αφορά την πιθανότητα να παραχθούν σύνολα με λιγότερα από k στοιχεία. Όπως είδαμε στο προηγούμενο κεφάλαιο, σε αυτήν την περίπτωση τα παραγόμενα δεδομένα δεν θα ικανοποιούν την ιδιότητα k-anonymity. Για το λόγο αυτό, κατά τον "αυστηρό" διαχωρισμό των tuples και πριν την αναδρομική κλήση του αλγορίθμου θα ελέγχουμε αν τα παραγόμενα σύνολα περιέχουν λιγότερα από k tuples. Σε αυτήν την περίπτωση, το σύνολο που έχει τα λιγότερα tuples θα απορροφά από το μεγαλύτερο σύνολο τα tuples εκείνα που στο εξεταζόμενο attribute έχουν τιμή που είναι η πλησιέστερη δυνατή στη διάμεσο[4]. Με αυτόν τον τρόπο "επεμβαίνουμε" στην πολιτική διαμέρισης του αλγορίθμου, αλλά παράλληλα, εγγυόμαστε την ορθή λειτουργία του strict partitioning. Παρακάτω παραθέτουμε τον αλγόριθμο Mondrian σε μορφή ψευδοκώδικα όταν εκτελεί relaxed και strict partitioning.

```
1 k          //αριθμός k−anonymity
2 mondrian(partition)          //relaxed partitioning
3   if (|partition|<2k)
4     return ∅
```

[3]Επειδή κάτι τέτοιο όμως είναι εξαιρετικά απίθανο, στο εξής θα συμβολίζουμε με log n τον αριθμό των βημάτων εκτέλεσης χωρίς να αναφέρουμε εκ νέου ότι η βάση του λογαρίθμου είναι ελαφρώς μικρότερη από 2.

[4]Με αυτόν τον τρόπο, αν το rPart έχει λιγότερα από k στοιχεία, θα απορροφά tuples που από το lPart με τη μεγαλύτερη δυνατή τιμή στο εξεταζόμενο attribute, ενώ στην αντίθετη περίπτωση θα απορροφούνται tuples με τη μικρότερη δυνατή τιμή.

```
5     else
6       dim = chooseDimension(partition)
7       splitValue = findMedian(partition,dim)
8       cPart = {t ∈ partition : t.dim = splitValue}
9       lPart = {t ∈ partition : t.dim < splitValue}
10      rPart = {t ∈ partition : t.dim > splitValue}
11      for(t ∈ cPart)
12        if(|lPart|>|rPart|)
13          rPart.add(t)
14        else
15          lPart.add(t)
16      return mondrian(lPart) ∪ mondrian(rPart)
```

```
1  k         //αριθμός k−anonymity
2  mondrian(partition)         //strict partitioning
3    if (|partition|<2k)
4      return ∅
5    else
6      dim = chooseDimension(partition)
7      splitValue = findMedian(partition,dim)
8      lPart = {t ∈ partition : t.dim =< splitValue}
9      rPart = {t ∈ partition : t.dim > splitValue}
10     while(lPart.length<k)
11       lPart = lPart + rPart.getClosestTupleToMedian()
12     while(rPart.length<k)
13       rPart = rPart + lPart.getClosestTupleToMedian()
14     return mondrian(lPart) ∪ mondrian(rPart)
```

Χρονικά, η συνάρτηση chooseDimension εκτελείται σε $q \cdot n$ βήματα, όπου n είναι το μέγεθος του partition και q είναι ο αριθμός των attributes που ανήκουν στο QID. Όλες οι υπόλοιπες διαδικασίες εκτελούνται σε γραμμικό χρόνο ως προς την είσοδο με εξαίρεση την getClosestTupleToMedian που χρειάζεται $k \cdot n$ βήματα για να εκτελεστεί. Συνολικά, η χρονική πολυπλοκότητα του αλγορίθμου Mondrian που εκτελεί relaxed partitioning είναι:

$$\mathcal{O}(q \cdot n \cdot \log \frac{n}{k})$$

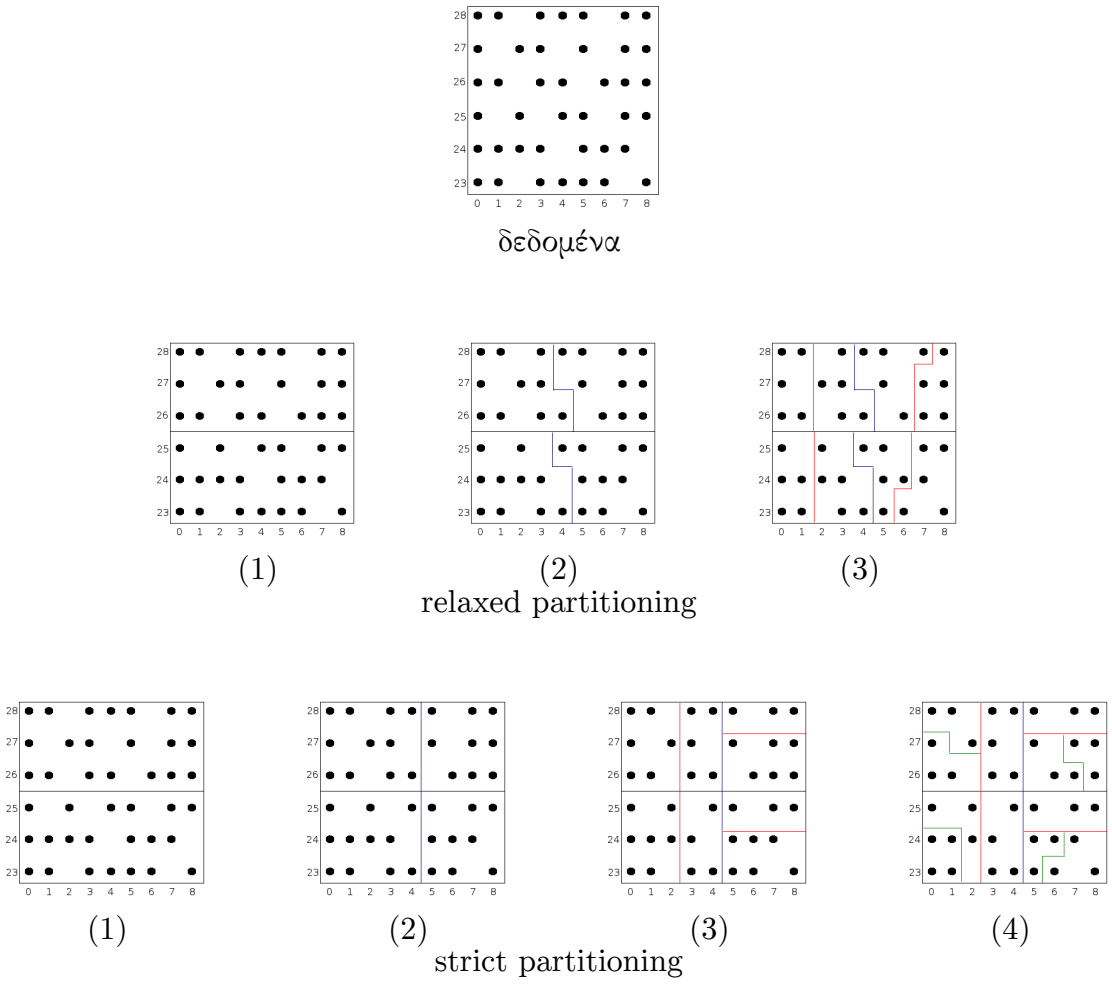
ενώ όταν εκτελείται strict partitioning η πολυπλοκότητα χειρότερης περίπτωση είναι ίση με:

$$\mathcal{O}((q+k) \cdot n \cdot \log \frac{n}{k})$$

Να σημειώσουμε ότι επειδή συνήθως τα μεγέθη q και k είναι πολύ μικρότερα του n στη βιβλιογραφία η πολυπλοκότητα του αλγορίθμου Mondrian αναφέρεται ως $\mathcal{O}(n \cdot \log n)$.

Τέλος, παραθέτουμε ένα μικρό παράδειγμα εκτέλεσης του αλγορίθμου Mondrian στην Εικόνα 3.1 για k=3 Στο επάνω μέρος του Σχήματος βλέπουμε τα δεδομένα στο δισδιάστατο επίπεδο. Στις 3 εικόνες που ακολουθούν βλέπουμε τα βήματα που θα εκτελέσει

ο αλγόριθμος αν εκτελέσει relaxed partitioning. Στις 4 τελευταίες εικόνες βλέπουμε τα βήματα που θα εκτελέσεις ο αλγόριθμος αν εκτελέσει strict partitioning.



Σχήμα 3.1: Παράδειγμα εκτέλεσης Mondrian για k=3

Με τη βοήθεια του παραπάνω σχήματος μπορούμε να εντοπίσουμε τις κυριότερες διαφορές που αναφέραμε και προηγουμένως για τους δυο τύπους partitioning. Στην περίπτωση του relaxed partitioning ο αλγόριθμος διαχωρίζει πάντα τα δεδομένα σε δυο ίσα κομμάτια. Αυτό έχει ως αποτέλεσμα την γρήγορη εκτέλεση του αλγορίθμου (3 βήματα εκτέλεσης έναντι 4 του strict partitioning), την ισοκατανομή του φορτίου, αλλά ταυτόχρονα την ύπαρξη μεγάλων κλάσεων ισοδυναμίας στο τέλος (κάθε κλάση ισοδυναμίας έχει 5 tuples ενώ k=3). Αντίθετα, στο strict partitioning έχουμε περισσότερα βήματα εκτέλεσης, οι κλάσεις που δημιουργούνται έχουν διαφορετικό αριθμό tuples, αλλά τελικά οι κλάσεις ισοδυναμίας είναι περισσότερες και μικρότερες. Αυτό συμβάλει στην καλύτερη ποιότητα ανωνυμοποίησης αφού η γενίκευση των tuples είναι μικρότερη και οι κλάσεις ισοδυναμίας, από τη στιγμή που περιέχουν λίγα σε αριθμό tuples, θα έχουν μικρό εύρος τιμών σε όλες τις διαστάσεις και τελικά, η απώλεια πληροφορίας είναι μειωμένη σε σχέση με την relaxed περίπτωση. Στο Κεφάλαιο 4 θα παραθέσουμε διάφορες εκτελέσεις για να εντοπίσουμε την ακριβή διαφορά στη λειτουργία του relaxed και του strict partitioning.

3.1.2 TopDown

Ο αλγόριθμος TopDown είναι ένας greedy αλγόριθμος που η λειτουργία του στηρίζεται στη συνεχή διαμέριση του χώρου του προβλήματος, όπως περιγράφηκε στην ενότητα 3.1. Ο αλγόριθμος παρουσιάζει πολλές ομοιότητες στη λειτουργία του με τον αλγόριθμο Mondrian που εξετάσαμε νωρίτερα, αλλά παρουσιάζει και αρκετές διαφορές. Αρχικά, παραθέτουμε τον αλγόριθμο σε μορφή ψευδοκώδικα:

```
1 k          //αριθμός του k−anonymity
2 topDown(partition, point1)
3   if (|partition|<2k)
4     return ∅
5   else
6     point2 = getMostDinstantTuple(point1)
7     part1 = {t ∈ partition : distance(t,point1)≤distance(t,point2)}
8     part2 = {t ∈ partition : distance(t,point1)>distance(t,point2)}
9     return topDown(part1) ∪ topDown(part2)
```

Ο αλγόριθμος αυτός παρουσιάζει πολλές ομοιότητες με την top down μέθοδο που περιγράφεται στο [6] (από εκεί ακριβώς έχει πάρει και το όνομά του). Η βασική ιδέα πίσω από τον αλγόριθμο είναι ότι δυο tuples θα καταλήξουν στην ίδια ομάδα σε κάποιο βήμα του αλγορίθμου αν και μόνο αν είναι αρκετά τοπικά. Η έννοια της τοπικότητας έχει άμεση σχέση με τη θέση των tuples στο q-διάστατο χώρο. Αν δυο tuples είναι αρκετά κοντά, τότε αναμένουμε να έχουν μικρές αποκλίσεις στα attributes τους και θέλουμε τα tuples αυτά να καταλήξουν στην ίδια ομάδα. Σε αντίθετη περίπτωση, αν δυο tuples βρίσκονται μακριά το ένα από το άλλο, τότε αναμένουμε να έχουν μεγάλες διαφορές στις τιμές των attributes τους και θέλουμε τα tuples αυτά να τοποθετηθούν σε διαφορετικές ομάδες.

Για να προσδιορίσουμε τις αποστάσεις ανάμεσα στα tuples υπάρχουν πολλοί τρόποι. Η δική μας προσέγγιση εξετάζει την απόσταση ενός σημείου σε σχέση με δυο σημεία που βρίσκονται πολύ μακριά μεταξύ τους (point1, point2). Μπορούμε δηλαδή να θεωρήσουμε ότι τα σημεία αυτά αποτελούν ακριανά σημεία του χωρίου το οποίο εξετάζουμε. Στη συνέχεια, για κάθε ενδιάμεσο σημείο, εξετάζουμε την απόσταση που έχει με τα δυο αυτά ακριανά σημεία. Με το σημείο που βρίσκεται πλησιέστερα σε αυτό, τοποθετείται στην ίδια ομάδα. Με την διαδικασία αυτή, προκύπτουν δυο σύνολα: ένα σύνολο που περιέχει σημεία που βρίσκονται πλησιέστερα στο ακρότατο point1 (σύνολο part1) και ένα σύνολο που περιέχει σημεία που βρίσκονται πλησιέστερα στο point2 (σύνολο part2). Συνεχίζουμε τη διαδικασία επαγωγικά μέχρι τα χωρία που έχουν προκύψει να μην μπορούν να διαιρεθούν περαιτέρω.

Κομβικό σημείο στην παραπάνω ανάλυση αποτελεί η απόσταση. Μπορούμε να χρησιμοποιήσουμε πολλές ευριστικές συναρτήσεις που να μας δίνουν την απόσταση δυο σημείων (για παράδειγμα, θα μπορούσαμε να χρησιμοποιήσουμε την ευκλείδια απόσταση, ή την απόσταση Manhattan). Επειδή όμως τα tuples εκτός από σημεία σε κάποιο n-διάστατο χώρο έχουν φυσική πληροφορία, θέλουμε να χρησιμοποιήσουμε μια έκφραση που να έχει

άμεση σχέση με το ποσό της πληροφορίας που χάνουμε όταν ομαδοποιούμε δυο μεμονωμένα tuples. Για το λόγο αυτό, χρησιμοποιούμε το Normalized Certainty Penalty (NCP) [12]. Το NCP υπολογίζεται για μια ομάδα από tuples (ισοδύναμα η ομάδα αυτή μπορεί να ονομάζεται και κλάση ισοδυναμίας) και αποτελεί μια έκφραση σύγκρισης ανάμεσα στο εύρος τιμών των attributes της ομάδας των tuples και το εύρος τιμών των attributes του πίνακα δεδομένων. Για μια διάσταση (ένα attribute) το NCP υπολογίζεται για την κλάση ισοδύναμιας G ως εξής:

$$NCP^{(i)}(G) = \frac{\max_G^{(i)} - \min_G^{(i)}}{\max^{(i)} - \min^{(i)}}$$

δηλαδή, είναι το πηλίκο ανάμεσα στο εύρος τιμών της κλάσης ισοδυναμίας στη διάσταση i ως προς το εύρος τιμών του πίνακα δεδομένων στη διάσταση i. Σε όλες τις διαστάσεις, το NCP υπολογίζεται ως εξής:

$$NCP(G) = \sum_{i \in QID} w_i NCP^{(i)}(G)$$

όπου QID είναι το Quasi Identifier και w_i είναι βάρη για τα οποία ισχύει $\sum_i w_i = 1$.[5] Στο επόμενο κεφάλαιο, θα δούμε πως μπορούμε να χρησιμοποιήσουμε το NCP για να υπολογίσουμε την απόδοση των αλγορίθμων ανωνυμοποίησης που κατασκευάσαμε.

Ένα δεύτερο κομβικό σημείο του παραπάνω αλγορίθμου, είναι επίσης η παραδοχή ότι τα σημεία point1 και point2 με τα οποία συγκρίνονται όλα τα υπόλοιπα σημεία, αποτελούν ακριανά σημεία του χωρίου στην αντίστοιχη n-διάστατη απεικόνιση. Η ιδιότητα αυτή είναι σημαντική γιατί σε περίπτωση που τα σημεία δεν ήταν ακριανά τότε δεν θα μπορούσαμε να εγγυηθούμε ότι η μεταξύ τους απόσταση θα ήταν η μέγιστη δυνατή, πράγμα που θα σήμαινε ότι τα δυο σημεία που εξ ορισμού θα καταλήξουν σε διαφορετικές ομάδες ίσως θα έπρεπε να καταλήξουν στην ίδια ομάδα με βάση τη μεταξύ τους απόσταση. Για να εξασφαλίσουμε, λοιπόν, ότι τα σημεία είναι πράγματι ακριανά σημεία του χωρίου που ορίζουν τα δεδομένα, παίρνουμε το σημείο εκείνο το οποίο έχει τη μικρότερη απόσταση από το σημείο με μηδενικές τιμές. Το σημείο αυτό ενδεχομένως δεν θα είναι ένα υπαρκτό σημείο, αλλά αποτελεί σημείο αναφοράς όλου του n-διάστατου χώρου. Στις περιπτώσεις που εξετάζουμε στην παρούσα εργασία, το σημείο που βρίσκεται πλησιέστερα στο σημείο αναφοράς είναι ακριανό σημείο του χωρίου[6].

Με βάση λοιπόν το σημείο που έχουμε βρει (το σημείο αυτό αντιστοιχεί στο point1 και δίνεται ως παράμετρος στον αλγόριθμο, έχει δηλαδή ήδη υπολογιστεί πριν την πρώτη εκτέλεση) βρίσκουμε το σημείο εκείνο το οποίο αν τοποθετηθεί στην ίδια κλάση ισοδυναμίας

[5] Τα βάρη χρησιμοποιούνται για να αυξομειώσουν τη σημασία κάθε attribute στον υπολογισμό του NCP. Επειδή στη δική μας ανάλυση θεωρούμε όλες τις διαστάσεις εξίσου σημαντικές, όλα τα βάρη έχουν την τιμή $w_i = \frac{1}{|QID|}$.

[6] Αυτό ισχύει σε περιπτώσεις που οι τιμές των attributes είναι μη αρνητικές -όπως στην εργασία αυτή-. Στη γενική περίπτωση, που κάποια ή και όλα τα attributes μπορούν να έχουν και αρνητικές τιμές μπορούμε να πάρουμε ως σημείο αναφοράς το σημείο εκείνο που έχει τιμές τις μικρότερες τιμές για κάθε attribute του πίνακα δεδομένων.

με το point 1 θα δώσει τη μεγαλύτερη τιμή για το NCP. Αυτό το σημείο είναι το point 2. Έχοντας ήδη ορίσει το NCP, μπορούμε εύκολα να αποδείξουμε ότι τα σημεία 1 και 2 στο πρώτο βήμα του αλγορίθμου αν τοποθετηθούν στην ίδια κλάση ισοδυναμίας θα έχουμε τιμή 1 για το NCP (τα σημεία αυτά δηλαδή ορίζουν διαστήματα attributes με μέγιστο εύρος). Στο δεύτερο βήμα του αλγορίθμου, η ομάδα που δημιουργήθηκε με σημείο αναφοράς το point 1 θα έχει ως σημείο αναφοράς και πάλι το point 1 (αφού ήταν ακρότατο πριν θα εξακολουθήσει να είναι και τώρα) με τη διαφορά ότι τώρα αναζητείται το νέο σημείο αναφοράς του νέου χωρίου. Αντίστοιχη λογική υπάρχει για το σύνολο part 2 που δημιουργήθηκε με σημείο αναφοράς το point 2. Με τον τρόπο αυτό, σε κάθε βήμα του αλγορίθμου το ένα σημείο αναφοράς είναι το ίδιο με το σημείο αναφοράς που χρησιμοποιήθηκε στο προηγούμενο βήμα και το άλλο υπολογίζεται εκ νέου με τον τρόπο που αναφέραμε. Στο παράδειγμα που ακολουθεί στο τέλος της ενότητας, θα δείξουμε σε κάθε βήμα της εκτέλεσης τα σημεία που χρησιμοποιούνται ως σημεία αναφοράς.

Τέλος, πριν παραθέσουμε την τελική μορφή του αλγορίθμου TopDown, πρέπει να αναφέρουμε μια πολύ μεγάλη ομοιότητα που υπάρχει ανάμεσα στον τρόπο που εκτελείται η διαμέριση στον αλγόριθμο TopDown και το strict partitioning που είδαμε στον αλγόριθμο Mondrian. Αν δούμε προσεκτικά τις γραμμές 8 και 9 του αλγορίθμου TopDown θα δούμε ότι ο αλγόριθμος εκτελεί strict partitioning, πράγμα που προκαλεί αρκετά προβλήματα απόδοσης όπως είδαμε αναλυτικά στην προηγούμενη ενότητα. Ένα πιθανό πρόβλημα λόγω strict partitioning είναι όπως είδαμε η αναδρομική κλήση της συνάρτησης topDown με την ίδια είσοδο, με συνέπεια την υπερχείλιση μνήμης. Εδώ όμως, δεν υπάρχει τέτοια περίπτωση, αφού σε κάθε βήμα του αλγορίθμου τουλάχιστον 1 στοιχείο τοποθετείται σε διαφορετική ομάδα, άρα ο χώρος του προβλήματος μειώνεται τουλάχιστον κατά 1 σε κάθε αναδρομή. Ένα δεύτερο πιθανό πρόβλημα είναι η δημιουργία συνόλων που έχουν μεγάλη διαφορά στο μέγεθος. Το πρόβλημα αυτό δεν αντιμετωπίζεται από τον TopDown αλγόριθμο, διότι θέλουμε η δημιουργία συνόλων να έχει άμεση σχέση με τη θέση των δεδομένων και όχι με τον αριθμό των στοιχείων των συνόλων. Τέλος, το τρίτο πρόβλημα που συνοδεύει το strict partitioning είναι η πιθανή δημιουργία συνόλων με λιγότερα από k στοιχεία. Το πρόβλημα αυτό αντιμετωπίζεται από τον αλγόριθμο με τον εξής τρόπο: αφού έχουν δημιουργηθεί τα σύνολα part1 και part2, αν κάποιο από τα 2 έχει λιγότερα από k στοιχεία τότε "απορροφά" από το άλλο σύνολο τα πιο μακρινά από το σημείο αναφοράς του άλλου συνόλου στοιχεία μέχρι να αποκτήσει k στοιχεία. Η τελική μορφή του TopDown αλγορίθμου είναι η ακόλουθη:

```
1 k          //αριθμός του k−anonymity
2 topDown(partition, point1)
3   if (|partition|<2k)
4     return ∅
5   else
6     point2 = getMostDinstantTuple(point1)
7     part1 = {t ∈ partition : NCP(part1,t)≤NCP(part2,t)}
```

```
8       part2 = {t ∈ partition : NCP(part1,t)>NCP(part2,t)}
9       while(part1.length<k)
10        part1 = part1 + part2.getFurthestTuple()
11      while(part2.length<k)
12        part2 = part2 + part1.getFurthestTuple()
13      return topDown(part1) ∪ topDown(part2)
```

Όπως βλέπουμε, η συνάρτηση distance αντικαταστάθηκε με την τιμή του NCP[7] και προσθέσαμε δυο while loops στο τέλος που ελέγχουν αν τα παραγόμενα σύνολα έχουν τουλάχιστον k στοιχεία.

Για τον υπολογισμό της πολυπλοκότητας, πρέπει να αναλύσουμε το κόστος των επιμέρους εργασιών που εκτελεί ο αλγόριθμος. Θα χρησιμοποιήσουμε τους ίδιους συμβολισμούς με προηγουμένως: n είναι ο αριθμός των tuples αρχικά, q είναι ο αριθμός των διαστάσεων του QID. Αρχικά, η getMostDinstantTuple απαιτεί $q \cdot n$ βήματα αφού βρίσκει την ελάχιστη απόσταση (άρα εξετάζει όλες τις διαστάσεις) για όλα τα tuples. Στη συνέχεια, το NCP υπολογίζεται σε χρόνο q και απαιτούνται $4 \cdot n$ υπολογισμοί του NCP. Η for-loop δηλαδή εκτελείται σε (ασυμπτωτικό) χρόνο $\mathcal{O}(q \cdot n)$. Τέλος, οι δυο while-loops έχουν χρόνο χειρότερης περίπτωση επίσης $\mathcal{O}(k \cdot q \cdot n)$. Ο συνολικός αριθμός βημάτων εκτέλεσης όπως είδαμε και για τον αλγόριθμο Mondrian είναι $\mathcal{O}(\log \frac{n}{k})$[8]. Τελικά, ο χρόνος εκτέλεσης της χειρότερης περίπτωσης του αλγορίθμου TopDown είναι ίσος με:

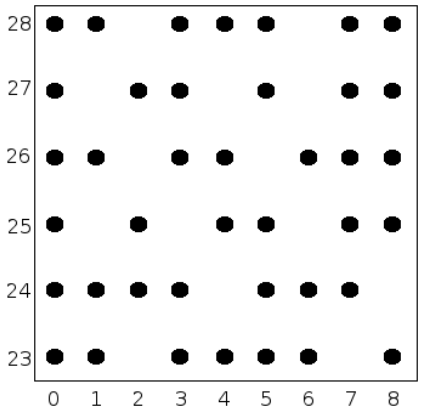
$$\mathcal{O}(k \cdot q \cdot n \cdot \log \frac{n}{k})$$

Παρατηρούμε ότι η χρονική πολυπλοκότητα του αλγορίθμου TopDown είναι λίγο χειρότερη από του αλγορίθμου Mondrian που εκτελεί strict partitioning. Η διαφορά είναι ότι οι μεταβλητές k, q που θα μπορούσαν να αγνοηθούν στην προηγούμενη περίπτωση, εδώ έχουν μεγαλύτερη επίδραση στο χρόνο εκτέλεσης του αλγορίθμου. Αυτό σημαίνει ότι μεταβολή των k και q θα επηρεάζουν πιο έντονα το χρόνο εκτέλεσης του αλγορίθμου TopDown παρά τον αλγόριθμο Mondrian.

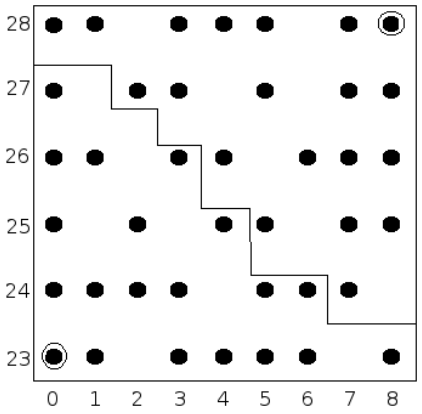
Στο Σχήμα 3.2 παραθέτουμε ένα παράδειγμα εκτέλεσης της TopDown στα δεδομένα που συναντήσαμε και στο σχετικό παράδειγμα εκτέλεσης του αλγορίθμου Mondrian. Και σε αυτό το παράδειγμα απεικονίζουμε τα δεδομένα στο δισδιάστατο επίπεδο για να δείξουμε τον τρόπο με τον οποίο πραγματοποιούνται οι διαμερίσεις. Θεωρούμε ότι το QID έχει δύο διαστάσεις και ότι k=3. Στις εικόνες (a)-(e) βλέπουμε τα βήματα εκτέλεσης του αλγορίθμου. Στο βήμα (a) βλέπουμε τα δεδομένα πριν την εκτέλεση του πρώτου βήματος. Στην εικόνα (b) έχουμε επιλέξει το πρώτο σημείο αναφοράς (το σημείο με τις συντεταγ-

[7] Η γραφή NCP(part1,t) υποδηλώνει την τιμή του NCP για την κλάση ισοδυναμίας που προκύπτει αν στο part1 προσθέσουμε το tuple t.

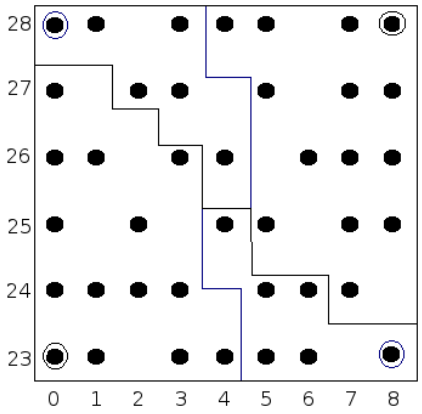
[8] Στην πραγματικότητα, από τη στιγμή που τα παραγόμενα σύνολα σε κάθε βήμα του αλγορίθμου δεν είναι απόλυτα ίσα (η είσοδος δεν υποδιπλασιάζεται δηλαδή) ο αριθμός των βημάτων δεν θα είναι ακριβώς ο λογάριθμος με βάση το 2 της εισόδου, αλλά η βάση του λογαρίθμου θα είναι λίγο μικρότερη. Επειδή όμως η λογαριθμική σχέση με την είσοδο εξακολουθεί να υφίσταται, μπορούμε να ισχυριστούμε ότι ο συνολικός αριθμός βημάτων είναι $\mathcal{O}(\log n)$.



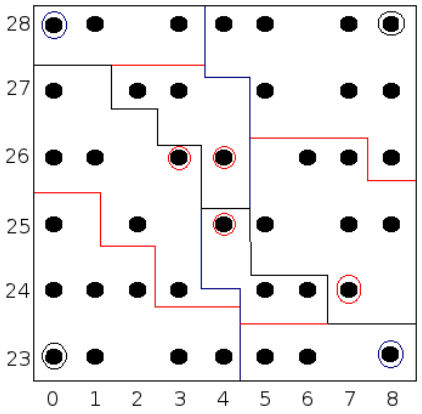
(a)



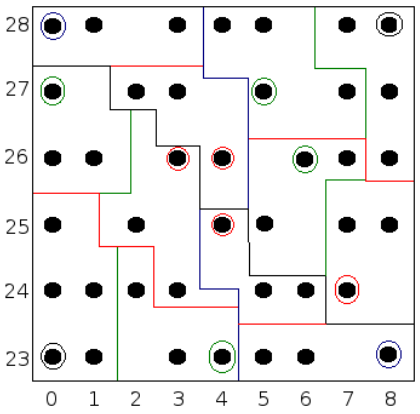
(b)



(c)



(d)



(e)

Σχήμα 3.2: Παράδειγμα εκτέλεσης TopDown για k=3

μένες $(23, 0)$ επειδή βρίσκεται πιο κοντά στο σημείο $(0, 0)$) και με βάση αυτό, βρίσκουμε το δεύτερο σημείο αναφοράς και δημιουργούμε μια διαμέριση του χώρου. Η τομή, καθώς και τα σημεία αναφοράς απεικονίζονται με μαύρο χρώμα. Στη συνέχεια, στην εικόνα (c), δουλεύουμε για κάθε σύνολο ξεχωριστά και με βάση το ήδη υπάρχον σημείο αναφοράς ενός συνόλου βρίσκουμε και πάλι το δεύτερο σημείο και με βάση τα δύο σημεία δημιουργούμε με τον ίδιο τρόπο τη διαμέριση των συνόλων. Τα νέα σημεία αναφοράς καθώς και οι νέες τομές στα σύνολα συμβολίζονται με μπλε χρώμα. Στην εικόνα (d) επαναλαμβάνουμε την ίδια διαδικασία για κάθε ένα από τα 4 σύνολα. Τα νέα σημεία αναφοράς και οι τομές έχουν κόκκινο χρώμα. Τέλος, στην εικόνα (e) βλέπουμε το τέλος εκτέλεσης του αλγορίθμου. Τα 4 από τα 8 σύνολα της εικόνας (d) μπορούν να διαμεριστούν εκ νέου. Πράγματι, στην εικόνα (e) φαίνονται οι διαμερίσεις και τα σημεία αναφοράς με πράσινο χρώμα. Παρατηρούμε ότι στην εικόνα (e) όλα τα σύνολα έχουν λιγότερα από $2k$ στοιχεία, άρα ο αλγόριθμος ολοκληρώνει την εκτέλεσή του.

Αν συγκρίνουμε τον αλγόριθμο TopDown με τον αλγόριθμο Mondrian (όταν εκτελεί relaxed partitioning) θα εντοπίσουμε και πάλι τις διαφορές που οφείλονται στο διαφορετικό είδος partitioning που εκτελεί ο κάθε αλγόριθμος και αναλύσαμε στην προηγούμενη ενότητα. Στο Κεφάλαιο 4 θα παρουσιάσουμε διάφορες συγκρίσεις μεταξύ της relaxed και strict πολιτικής διαμέρισης, ενώ παράλληλα θα συγκρίνουμε και τους αλγορίθμους Mondrian και TopDown.

3.2 Κατανεμημένη εκτέλεση

Στην προηγούμενη ενότητα αναφερθήκαμε στους αλγορίθμους ανωνυμοποίησης που θα χρησιμοποιήσουμε. Στην ενότητα αυτή, θα δούμε πως ακριβώς μπορούμε να χρησιμοποιήσουμε τους αλγορίθμους αυτούς με τέτοιο τρόπο έτσι ώστε να εκτελούνται με κατανεμημένο τρόπο σε ένα cluster υπολογιστών.

Σκοπός μας είναι να εφαρμόσουμε τους δυο αλγορίθμους με ένα όσο πιο γενικό τρόπο γίνεται, έτσι ώστε να μπορούν να εκτελούνται με υψηλή απόδοση ανεξάρτητα από το είδος και την κατανομή των δεδομένων. Όπως είδαμε και στην Ενότητα 2.3.1 το προγραμματιστικό μοντέλο MapReduce χρησιμοποιείται για τη συγγραφή κατανεμημένων προγραμμάτων που δίνουν λύση σε προβλήματα που μπορούν εύκολα να παραλληλοποιηθούν (embarrassingly parallel problems). Στη συγκεκριμένη περίπτωση, παρατηρούμε ότι το πρόβλημα της ανωνυμοποίησης δεν μπορεί να παραλληλοποιηθεί με κάποιο εύκολο τρόπο, αφού οι αλγόριθμοι που εξετάσαμε πρέπει σε κάθε χρονική στιγμή να γνωρίζουν τα attributes όλων των tuples που εξετάζουν.

Σε περιπτώσεις που έχουμε πολύ μεγάλα datasets και για μικρές τιμές του k[9] μπορούμε να ισχυριστούμε ότι τα tuples μπορούν να χωριστούν σε ένα σύνολο υποομάδων (κάθε μια από τις οποίες θα ανωνυμοποιηθεί ξεχωριστά) χωρίς να έχουμε μεγάλη μείωση

[9]Με τον όρο μικρές τιμές εννοείται το k πρέπει να είναι πολύ μικρότερο από τον αριθμό των tuples.

στην απόδοση. Αν, για παράδειγμα, χωρίσουμε το σύνολο των n tuples σε m ομάδες, τότε μπορούμε να εκτελέσουμε τους αλγορίθμους ανωνυμοποίησης ταυτόχρονα σε m υπολογιστές. Το μειονέκτημα αυτής της μεθόδου είναι ότι τα tuples που έχουν χωριστεί σε διαφορετικές υποομάδες εξαρχής, δεν θα καταλήξουν ποτέ στην ίδια τελική κλάση ισοδυναμίας (μετά την ανωνυμοποίηση) ακόμα και αν θα έπρεπε με βάση τη μεταξύ τους απόσταση. Το πρόβλημα δηλαδή εντοπίζεται μεταξύ συνοριακών tuples γειτονικών ομάδων. Παρ'όλα αυτά, αν ο αριθμός των διαμερίσεων είναι μικρός σε σχέση με το n (το συνολικό αριθμό των tuples δηλαδή) το μειονέκτημα αυτό δεν επηρεάζει σε μεγάλο βαθμό την τελική απόδοση και μπορεί να θεωρηθεί αμελητέο. Στις υποενότητες που ακολουθούν, θα περιγράψουμε τη διαδικασία με την οποία διαμερίζουμε τον αρχικό πίνακα δεδομένων.

3.2.1 Εύρεση πεδίων τιμών διαστάσεων

Αρχικά, διατρέχουμε όλα τα tuples του πίνακα δεδομένων με τη βοήθεια ενός Map Reduce job και υπολογίζουμε το μέγιστο και το ελάχιστο στοιχείο κάθε διάστασης. Τα δεδομένα εισόδου θεωρούμε ότι είναι οργανωμένα σε αρχεία που περιέχουν τα tuples (1 tuple/γραμμή). Για κάθε αρχείο δημιουργείται ένας mapper ο οποίος διατρέχει το αρχείο και για κάθε attribute των tuples κρατάει τη μέγιστη και την ελάχιστη τιμή που έχει παρουσιαστεί. Μόλις το αρχείο διαβαστεί πλήρως, ο mapper γράφει στην έξοδο σαν κλειδί το όνομα της διάστασης (ένας αριθμός από το 1 μέχρι το πλήθος των διαστάσεων του QID) και σαν τιμή το ζεύγος μέγιστης και ελάχιστης τιμής που βρήκε από το συγκεκριμένο αρχείο. Κάθε mapper δηλαδή διαβάζει ένα αρχείο και έχει σαν έξοδο τόσα key/values όσες είναι και διαστάσεις που ανήκουν στο QID.

Στη συνέχεια, τα key/values όλων των mappers (όλων των αρχείων δηλαδή) διαβάζονται από έναν reducer και για κάθε κλειδί (για κάθε attribute δηλαδή) υπολογίζεται το γενικό μέγιστο και το γενικό ελάχιστο όλων των αρχείων. Με τον τρόπο αυτό γνωρίζει το range (το διάστημα στο οποίο παίρνει τιμές ένα attribute) κάθε attribute. Στο τέλος, ο reducer ταξινομεί τις διαστάσεις σε αύξουσα σειρά range και γράφει στην έξοδο ένα key/value για κάθε διάσταση. Ως κλειδί γράφεται το όνομα της διάστασης και ως value γράφεται το range της διάστασης.

Η εύρεση των ranges των διαστάσεων είναι μια πολύ σημαντική εργασία, γιατί με αυτόν τον τρόπο κάθε attribute αποκτά βαρύτητα ανάλογα με το range που του αντιστοιχεί. Συγκεκριμένα, όσο μικρότερο range έχει ένα attribute τόσο σημαντικότερο θεωρείται. Αυτό είναι λογικό, αφού θέλουμε τα attributes που είναι σημαντικά να γενικεύονται πιο "δύσκολα" από τα attributes που είναι λιγότερο σημαντικά. Για παράδειγμα, ένα attribute που συναντάται συχνά σε δεδομένα που έχουν σχέση με άτομα είναι το φύλο και ο ταχυδρομικός κώδικας. Το φύλο έχει range 2 (Male, Female) ενώ ο ταχυδρομικός κώδικας μπορεί να έχει range που ξεπερνά το 100. Όταν λοιπόν ανωνυμοποιήσουμε δεδομένα που περιέχουν αυτά τα 2 attributes, λαμβάνουμε περισσότερη πληροφορία αν γνωρίζουμε επα-

κριβώς το φύλο, αλλά δεν γνωρίζουμε με μεγάλη ακρίβεια τον ταχυδρομικό κώδικα, παρά το αντίστροφο[10]. Με αντίστοιχη λογική, προτιμούμε να δώσουμε βαρύτητα σε attributes με μικρό range σε σχέση με attributes που έχουν μεγαλύτερο range.

Έχοντας λοιπόν ορίσει μια ιεραρχία ανάμεσα στα attributes του QID, μπορούμε να ορίσουμε και μια σχέση διάταξης ανάμεσα στα tuples που θα μας φανεί πολύ χρήσιμη στη συνέχεια. Θα λέμε δηλαδή ότι ένα tuple A είναι μεγαλύτερο από ένα tuple B αν και μόνο αν το attribute A έχει μεγαλύτερη τιμή από το B στο πιο σημαντικό attribute. Σε περίπτωση που η τιμή των δυο tuples είναι ίδια στο πιο σημαντικό attribute θα ελέγχουμε το δεύτερο σημαντικότερο attribute, κοκ. Αν τα δυο tuples έχουν τις ίδιες τιμές σε όλα τα attributes τότε θα λέμε ότι είναι ίσα. Σε περίπτωση που το B έχει μεγαλύτερη τιμή από το A στο πιο σημαντικό attribute (ή στο πρώτο attribute που το A και B δεν έχουν ίδια τιμή), τότε το A θα είναι μικρότερο του B. Η παραπάνω σχέση διάταξης που ορίσαμε μπορεί να αναπαρασταθεί με τη βοήθεια της συνάρτητης compare ως εξής:

```
1 compare(Tuple a, Tuple B)
2   for(att ∈ QID)
3     if(a.att > b.att)
4       return BIGGER
5     else if (a.att < b.att)
6       return SMALLER
7   return EQUAL
```

όπου θεωρούμε ότι το QID περιέχει τα attributes ταξινομημένα σε αύξουσα σειρά range (ή ισοδύναμα ταξινομημένα κατά φθίνουσα σειρά σπουδαιότητας). Στο εξής, το QID θα θεωρούμε ότι περιέχει ταξινομημένα attributes ως προς τη σειρά σπουδαιότητας, χωρίς να αναφέρεται εκ νέου.

3.2.2 Sampling-Εύρεση τομών

Έχοντας ορίσει μια σχέση διάταξης για τα tuples όπως είδαμε προηγουμένως, θέλουμε με βάση αυτή τη σχέση να τα ταξινομήσουμε και στη συνέχεια να τα χωρίσουμε σε υποομάδες. Θέλουμε οι ομάδες που θα δημιουργηθούν να περιέχουν τον ίδιο ή περίπου τον ίδιο αριθμό από tuples. Σε αντίθετη περίπτωση, αν κάποια ομάδα περιέχει πολλά περισσότερα στοιχεία από τις υπόλοιπες τότε η διεργασία που θα τρέξει για τα συγκεκριμένα δεδομένα θα καθυστερήσει σε σχέση με τις διεργασίες που θα αναλάβουν τις υπόλοιπες ομάδες, ενώ παράλληλα υπάρχει και ο κίνδυνος η διεργασία να μην τερματίσει αν τα δεδομένα είναι εξαιρετικά πολλά. Γενικά, σε ένα cluster είναι επιθυμητό μια εργασία που εκτελείται ταυτόχρονα σε πολλούς υπολογιστές να καταναλώνει περίπου τον ίδιο χρόνο σε κάθε υπολογιστή και να μην "επιβαρύνει" κάποιο κόμβο του cluster με πολύ μεγάλο υπολογιστικό φορτίο. Ο όρος αυτός αναφέρεται ως load balancing (εξισορρόπηση φορτίου όπως είδαμε

[10] Δηλαδή, θυσιάζουμε πληροφορία στο tuple που μπορεί να λάβει πολλές τιμές για να κερδίσουμε περισσότερη πληροφορία στο tuple που μπορεί να λάβει λιγότερες τιμές.

και σε προηγούμενη ενότητα) και είναι μια πολύ σημαντική παράμετρος στην αποδοτική εκτέλεση της κατανεμημένης εφαρμογής.

Ο προφανής τρόπος με τον οποίο θα μπορούσαμε να χωρίσουμε τα δεδομένα σε m ομάδες είναι ο εξής: αρχικά θα διαβάζαμε όλα τα αρχεία εισόδου (1 mapper για κάθε αρχείο εισόδου). Κάθε mapper θα έγραφε στην έξοδο κάθε tuple που διάβαζε (το Tuple θα εγγραφόταν ως κλειδί και το value δεν θα είχε τιμή), έτσι ώστε όλα τα tuples όλων των αρχείων να καταλήξουν ταξινομημένα στην είσοδο του reducer και στη συνέχεια, ο reducer θα δημιουργούσε m αρχεία στα οποία θα τοποθετούσε στο πρώτο αρχείο τα πρώτα $\frac{n}{m}$ tuples, στο δεύτερο αρχείο τα επόμενα $\frac{n}{m}$ tuples, κοκ.

Αν και με τη μέθοδο αυτή δημιουργούνται ακριβώς m αρχεία ακριβώς ίδιου μεγέθους το καθένα, υπάρχει ένα σοβαρό μειονέκτημα. Κατά την εκτέλεση της εργασίας τα δεδομένα διαβάζονται αρχικά από έναν αριθμό από mappers αλλά στη συνέχεια όλα τα (ταξινομημένα πλέον) δεδομένα διαβάζονται από 1 reducer. Αυτό σημαίνει ότι μία μόνο διεργασία (ο reducer) πρέπει να εξετάσει όλα τα δεδομένα, πράγμα που σημαίνει ότι όσο περισσότερα είναι τα δεδομένα τόσο περισσότερο θα καθυστερήσει η εκτέλεση[11] ενώ παράλληλα, επειδή ο reducer θα εκτελεστεί σε ένα συγκεκριμένο κόμβο του cluster, ο κόμβος αυτός θα επιβαρυνθεί με πολύ μεγάλο υπολογιστικό φορτίο.

Για να μειώσουμε το υπολογιστικό φορτίο στον κόμβο που θα εκτελεστεί ο reducer μπορούμε να εκτελέσουμε δειγματοληψία στα tuples που θα στείλουν οι mappers στον reducer. Αυτό σημαίνει ότι ενώ οι mappers θα διαβάσουν όλα τα tuples των αρχείων, δεν θα στείλουν όλα τα tuples που διαβάζουν στον reducer, αλλά θα στείλουν μόνο ένα μέρος από τα tuples αυτά. Αν, για παράδειγμα, θέλουμε ο reducer να διαβάσει το 10% του συνόλου των tuples τότε κάθε mapper θα στείλει ένα tuple στον reducer με πιθανότητα 0.1[12]. Τελικά, όταν οι mappers έχουν ολοκληρώσει το διάβασμα όλων των αρχείων, ο reducer θα έχει ως είσοδο το 10% του συνόλου των tuples. Στη συνέχεια, ο reducer θα εξετάσει τα tuples που βρίσκονται στις θέσεις $\frac{0.1 \cdot n}{m} \cdot i$ για $i = 1, .., m-1$ των ταξινομημένων tuples. Τα tuples που βρίσκονται στις θέσεις αυτές θα λειτουργήσουν σαν τομές μεταξύ δυο διαφορετικών υποσυνόλων, θα είναι δηλαδή τα όρια των υποσυνόλων.

Στη γενική περίπτωση, αν θεωρήσουμε ότι το ποσοστό δειγματοληψίας συμβολίζεται με s, ο αριθμός όλων των tuples είναι n και θέλουμε να δημιουργήσουμε m ομάδες από tuples, ο reducer θα γράφει στην έξοδο 1 tuple για κάθε $\frac{s \cdot n}{m}$ tuples που διαβάζει. Για μεγάλες τιμές του s (τιμές κοντά στο 1), τα tuples που τυπώνει στην έξοδο ο reducer θα είναι πολύ κοντά στις πραγματικές τομές του συνόλου των δεδομένων, αλλά παράλληλα θα εξετάζει πολλά tuples με τις συνέπειες που είδαμε προηγουμένως. Για μικρές τιμές του s,

[11] Πέρα από την καθυστέρηση της ολοκλήρωσης της εργασίας του reducer, όταν ένα task έχει πολλά δεδομένα να προσπελάσει, αυξάνεται η πιθανότητα να παρουσιαστεί σφάλμα εκτέλεσης και να επανεκκινήσει το task ή να αποτύχει, κάτι που θα δημιουργήσει επιπλέον καθυστέρηση.

[12] Από άποψη υλοποίησης, αυτό μπορεί να γίνει με τη χρήση μιας γεννήτριας τυχαίων αριθμών. Σε κάθε εκτέλεση της map (συνάρτηση που καλεί ο mapper για κάθε γραμμή εισόδου) επιλέγεται ένα αριθμός από το 1 ως το 100. Αν ο αριθμός αυτός είναι μικρότερος ή ίσος του 10 τότε ο mapper θα γράψει το συγκεκριμένο tuple που εξετάζει στην έξοδο ενώ σε διαφορετική περίπτωση το αγνοεί.

ο χρόνος εκτέλεση βελτιώνεται πολύ αλλά παράλληλα αυξάνεται η πιθανότητα τα tuples-τομές που τυπώνει ο reducer να βρίσκονται αρκετά μακριά από τα πραγματικά. Για την εύρεση μια καλής συχνότητας δειγματοληψίας απαιτείται ένας συμβιβασμός ανάμεσα στην μεγάλη ακρίβεια εύρεσης των tuples-τομών και στην ταχύτητα εύρεσής τους.

3.2.3 Ταξινόμηση και διαμέριση

Έχοντας βρει τις τομές μεταξύ των υποσυνόλων των δεδομένων μπορούμε να δημιουργήσουμε τελικά τις ομάδες των δεδομένων. Οι τομές των ομάδων έχουν δημιουργηθεί από προηγούμενο MapReduce job και βρίσκονται αποθηκευμένα σε ένα γνωστό αρχείο στο HDFS. Αρχικά λοιπόν, κατά την αρχικοποίηση του MapReduce job πρέπει να ανοίξουμε το αρχείο αυτό και να διαβάσουμε τα tuples-τομές των ομάδων.

Στη συνέχεια, δημιουργούνται οι mappers που θα διαβάσουν εκ νέου τα αρχεία εισόδου. Οι mappers για κάθε γραμμή κειμένου που διαβάζουν δημιουργούν ένα tuple και στη συνέχεια το γράφουν στην έξοδο σαν κλειδί. Το πεδίο value παραμένει χωρίς τιμή. Κανονικά, σε περίπτωση που χρησιμοποιούμε ένα reducer (όπως στις περιπτώσεις που έχουμε συναντήσει μέχρι τώρα), τα tuples που έχουν γράψει όλοι οι mappers ταξινομούνται και στέλνονται στο μοναδικό reducer. Σε αυτήν την περίπτωση όμως, η χρήση μοναδικού reducer δεν είναι επιθυμητή για τον ίδιο λόγο που αναφέραμε στην προηγούμενη υποενότητα. Αν χρησιμοποιήσουμε μοναδικό reducer, τότε όλα τα δεδομένα θα σταλούν σε έναν κόμβο πράγμα που θα δημιουργήσει υπερφόρτωση ενός κόμβου. Από τη στιγμή κιόλας που η έξοδος του MapReduce job θέλουμε να είναι κάποιες υποομάδες από tuples (το οποίο θα αντιστοιχεί σε ένα αρχείο για κάθε υποομάδα), μπορούμε να δημιουργήσουμε m reducers, κάθε ένας από τους οποίους θα είναι υπεύθυνος για μια υποομάδα δεδομένων. Δηλαδή θα χρησιμοποιήσουμε τόσους reducers όσες είναι και οι ομάδες που θέλουμε να δημιουργήσουμε και κάθε reducer θα είναι υπεύθυνος για την εγγραφή ενός αρχείου που αντιστοιχεί στην ομάδα αυτή.

Από άποψη υλοποίησης μπορούμε να χρησιμοποιήσουμε έναν μηχανισμό του MapReduce για να στείλουμε τα tuples στους reducers που τους αντιστοιχούν. Ο μηχανισμός αυτός ονομάζεται partitioner, εκτελείται σε κάθε MapReduce job και είναι υπεύθυνος για την αποστολή ενός key/value pair από την έξοδο του mapper στην είσοδο του σωστού reducer. Μέχρι στιγμής, τα MapReduce jobs που έχουμε χρησιμοποιήσει είχαν μόνο 1 reducer και για αυτό χρησιμοποιήθηκε ο default Partitioner του MapReduce που απλά προωθεί όλα τα key/value pairs όλων των mappers στο έναν και μοναδικό reducer. Στην περίπτωση αυτή όμως, που χρησιμοποιούνται πολλοί reducers, πρέπει να χρησιμοποιήσουμε έναν κατάλληλο partitioner ο οποίος να στέλνει με κατάλληλο τρόπο τα tuples στον reducer που τους αντιστοιχεί.

Η υλοποίηση ενός τέτοιου partitioner είναι απλή. Για κάθε tuple που γράφει στην έξοδό του ένας mapper, ο partitioner θα ελέγχει ως προς ποια τομή είναι μικρότερο και

μόλις βρεθεί η τομή ως προς την οποία είναι μικρότερο, θα το τοποθετεί στο προηγούμενο σύνολο. Κομβικό ρόλο στην παραπάνω διαδικασία έχει η ταξινομημένη λίστα των τομών (από το μικρότερο στο μεγαλύτερο ως προς τη σχέση compare που είδαμε στην ενότητα 3.2.1). Σε μορφή κώδικα ο partitioner θα έχει ως εξής:

```
1 cuts       //sorted list of tuples/cuts
2 partitioner(Tuple tuple)
3   partition=0
4   for(Tuple cut:cuts)
5     if compare(tuple,cut)==SMALLER
6       return partition
7     else
8       partition=partition+1
9   return partition
```

όπου cuts θεωρούμε ότι είναι η ταξινομημένη λίστα των τομών και η συνάρτηση partitioner επιστρέφει τον αριθμό του reducer στο οποίο θα καταλήξει το tuple.

Ένα βασικό πλεονέκτημα της παραπάνω διαδικασίας είναι ότι ο partitioner στέλνει τα tuples στους κατάλληλους reducers αμέσως μετά την ολοκλήρωση της map συνάρτησης, πράγμα που σημαίνει ότι κάθε mapper έχει τον δικό της partitioner. Η παραπάνω διαδικασία δηλαδή εκτελείται κατανεμημένα σε κάθε κόμβο που υπάρχει ένα mapper.

Αφού ολοκληρώσουν την εκτέλεσή τους όλοι οι mappers τα tuples έχουν καταλήξει στα splits που τους αντιστοιχούν. Στην συνέχεια, ταξινομούνται και τελικά καταλήγουν στους reducers οι οποίοι με τη σειρά τους γράφουν τα tuples στην έξοδο. Τελικά, δημιουργούνται αρχεία κειμένου που περιέχουν τα tuples ταξινομημένα και κάθε αρχείο είναι μια υποομάδα δεδομένων. Τα δεδομένα δηλαδή όλων των αρχείων είναι ακριβώς τα ίδια με τον αρχικό πίνακα δεδομένων.

3.2.4 Εκτέλεση Ανωνυμοποίησης

Τελικά, μετά την δημιουργία των υποομάδων, μπορούμε να εκτελέσουμε τους αλγορίθμους ανωνυμοποίησης που αναλύσαμε στην Ενότητα 3.1. Για κάθε αρχείο που περιέχει τα tuples μια ομάδας, θα δημιουργηθεί ένα mapper που θα διαβάσει όλα τα tuples. Μόλις τελειώσει η ανάγνωση των αρχείων, μπορούμε να εκτελέσουμε τους αλγορίθμους ανωνυμοποίησης[13]. Η μέθοδος map σε αυτήν την περίπτωση, κάθε φορά που διαβάζει μια γραμμή κειμένου, δημιουργεί ένα tuple και τα εισάγει σε μια λίστα. Μόλις διαβαστούν όλα τα tuples η λίστα αυτή δίνεται σαν είσοδος στους αλγορίθμους ανωνυμοποίησης. Μετά την ολοκλήρωση των αλγορίθμων, τα σύνολα των κλάσεων ισοδυναμίας που παράχθηκαν από τους αλγορίθμους γράφονται σε αρχεία στο HDFS.

[13]Οι περισσότερες υλοποιήσεις του MapReduce δίνουν τη δυνατότητα στον προγραμματιστή να προσθέσει κώδικα αμέσως μετά την τελευταία κλήση της συνάρτησης map, που καλείται όπως είδαμε για κάθε γραμμή του αρχείου εισόδου. Στο Hadoop, προσφέρεται αυτή η δυνατότητα μέσω της μεθόδου close που μπορεί να γίνει override από τον προγραμματιστή.

Ένα σημαντικό ζήτημα που προκύπτει με την ολοκλήρωση της εκτέλεσης, είναι η απόδοση των αλγορίθμων. Είδαμε στην Ενότητα 3.1.2 ότι ένας δείκτης που έχει άμεση σχέση με την απώλεια πληροφορίας είναι ο δείκτης NCP. Για την συνολική αξιολόγηση του αλγορίθμου, μπορούμε να χρησιμοποιήσουμε τον δείκτη Global Certainty Penalty (GCP) [12] ο οποίος ορίζεται ως εξής:

$$GCP(P) = \frac{\sum_{G \in P} |G| \cdot NCP(G)}{d \cdot N}$$

όπου P είναι το σύνολο που περιέχει τις κλάσεις ισοδυναμίας που παρήγαγε ο αλγόριθμος, με G συμβολίζεται η κλάση ισοδυναμίας, d είναι ο αριθμός των attributes που περιέχει του QID και N είναι ο αριθμός των tuples στον αρχικό πίνακα δεδομένων.

Για τον υπολογισμό του GCP, μόλις ολοκληρωθεί η εκτέλεση των αλγορίθμων, υπολογίζεται από κάθε mapper ο όρος $\sum_{G \in P_i} |G| \cdot NCP(G)$, όπου i είναι ο αριθμός του συνόλου των κλάσεων ισοδυναμίας (συνολικά θα υπάρχουν m σύνολα που θα περιέχουν κλάσεις ισοδυναμίας του ίδιου αλγορίθμου). Στη συνέχεια, ο mapper γράφει στην έξοδο ένα key/value pair: ένα κλειδί με τίτλο το όνομα του αλγορίθμου που εκτελέστηκε και value ίσο με το παραπάνω άθροισμα για το σύνολο που παρήγαγε ο αλγόριθμος που εκτελέστηκε. Μόλις όλοι οι mappers ολοκληρώσουν την εκτέλεσή τους, τα key/values καταλήγουν σε ένα reducer ο οποίος αθροίζει όλα τα values που έχουν το ίδιο κλειδί. Τέλος, ο reducer μετά την άθροιση διαιρεί τα αποτελέσματα με τον όρο $d \cdot N$ και τυπώνει στην έξοδο ένα key/values που αντιστοιχεί στο όνομα του αλγορίθμου που εκτελέστηκε[14] και έναν αριθμό που αντιστοιχεί στο GCP του αλγορίθμου.

[14]Πχ, TopDown, Mondrian (strict), Mondrian (relaxed) και άλλα.

Κεφάλαιο 4

Εκτελέσεις

Στο κεφάλαιο αυτό θα παρουσιάσουμε τα αποτελέσματα που λάβαμε από την εκτέλεση των αλγορίθμων ανωνυμοποίησης που παρουσιάσαμε στο Κεφάλαιο 3, με κατανεμημένο τρόπο. Θα παρουσιάσουμε τα δεδομένα που χρησιμοποιήσαμε ως είσοδο των αλγορίθμων, θα παραθέσουμε τη συνάρτηση με την οποία αξιολογούμε την απόδοση των αλγορίθμων και, τέλος, θα παρουσιάσουμε τα αποτελέσματα που λάβαμε τόσο ως προς το χρόνο όσο και ως προς την ποιότητα της ανωνυμοποίησης σε κάθε περίπτωση εκτέλεσης.

4.1 Δεδομένα εισόδου

Στην ενότητα αυτή θα περιγράψουμε τα χαρακτηρστικά των δεδομένων που θα χρησιμοποιήσουμε ως είσοδο στο κατανεμημένο σύστημα. Χρειαζόμαστε κάποια δεδομένα που θα χρησιμοποιηθούν για να συγκρίνουμε την απόδοση του κεντρικοποιημένου αλγορίθμου με την κατανεμημένη εκτέλεση (προφανώς τα δεδομένα αυτά πρέπει να είναι μικρού μεγέθους για να μπορούν να εκτελεστούν κεντρικά) και χρειαζόμαστε και δεδομένα τα οποία πρέπει να είναι πολύ μεγάλα έτσι ώστε να ελέγξουμε αν η κατανεμημένη εκτέλεση επιτρέπει την ανωνυμοποίηση ενός πολύ μεγάλου dataset. Στις επόμενες ενότητες περιγράφονται τα χαρακτηριστικά των δυο κατηγοριών δεδομένων.

4.1.1 Δεδομένα για κεντρική εκτέλεση

Αρχικά, θέλουμε να χρησιμοποιήσουμε κάποια datasets για να συγκρίνουμε την κατανεμημένη με την κεντρική εκτέλεση των αλγορίθμων. Η σύγκριση αυτή είναι πολύ χρήσιμη για να εντοπίσουμε πως μεταβάλλεται η απόδοση των αλγορίθμων συναρτήσει του αριθμού των partitions. Για να μπορέσουμε όμως να τρέξουμε με κεντρικό τρόπο έναν αλγόριθμο, πρέπει το dataset να είναι αρκετά μικρό. Για το λόγο αυτό χρησιμοποιούμε τα εξής datasets:

- Το πραγματικό dataset Adults [13] το οποίο έχει 15 attributes και αποτελείται από περίπου 32000 tuples,

- και ένα απλό συνθετικό dataset που κατασκευάσαμε για την κεντρική εκτέλεση. Το δεύτερο dataset αποτελείται από 35000 tuples και κάθε tuple έχει 5 attributes, κάθε ένα από τα οποία παίρνει τιμές στο διάστημα $[0 - 100]$ με ομοιόμορφη κατανομή (περισσότερα για τις ομοιόμορφες κατανομές στην επόμενη ενότητα).

4.1.2 Συνθετικά δεδομένα για κατανεμημένη εκτέλεση

Τα δεδομένα που θα χρησιμοποιήσουμε ως είσοδο στην κατανεμημένη μέθοδο ανωνυμοποίησης αποτελούνται από tuples που περιέχουν 10 attributes. Τα attributes έχουν διαφορετικά εύρη τιμών και όλα τα attribute του ίδιου tuple ακολουθούν την ίδια κατανομή. Τα εύρη τιμών των tuples παρατίθενται στον παρακάτω πίνακα (Πίνακας 4.1).

Διαστάσεις	1	2	3	4	5	6	7	8	9	10
Μέγιστο	1	80	5	1000	1000	100	100	51000	1000	10
Ελάχιστο	0	20	1	800	0	0	1	50000	100	0
Εύρος	2	61	5	201	1001	101	100	1001	901	11

Πίνακας 4.1: Πίνακας εύρους τιμών των attributes των δεδομένων

Σημειώνουμε ότι το εύρος των διαστημάτων δεν είναι απλά η διαφορά μεταξύ μέγιστου και ελάχιστου στοιχείου, αφού τα παραπάνω διαστήματα είναι κλειστά (μια τιμή μπορεί να ισούται με το μέγιστο ή το ελάχιστο του αντίστοιχου διαστήματος). Για το λόγο αυτό, το εύρος είναι προσαυξημένο κατά ένα σε σχέση με τη διαφορά του μεγαλύτερου από το ελάχιστο στοιχείο.

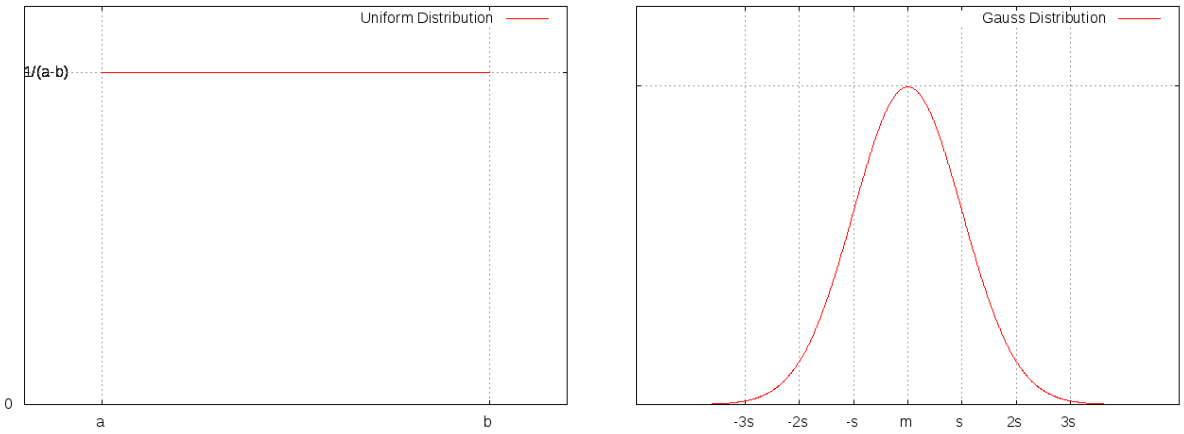
Ως προς την κατανομή που ακολουθούν οι τιμές των attributes, μας ενδιαφέρει να δούμε πως επηρεάζει την ποιότητα της ανωνυμοποίησης η πόλωση των δεδομένων γύρω από συγκεκριμένες τιμές. Για το λόγο αυτό, τα δεδομένα που χρησιμοποιούμε ακολουθούν μια από τις ακόλουθες κατανομές:

- *Uniform* κατανομή, στην οποία όλες οι τιμές στο διάστημα $[a, b]$ έχουν την ίδια πιθανότητα εμφάνισης. Η συνάρτηση πυκνότητας πιθανότητας της καμπύλης είναι η

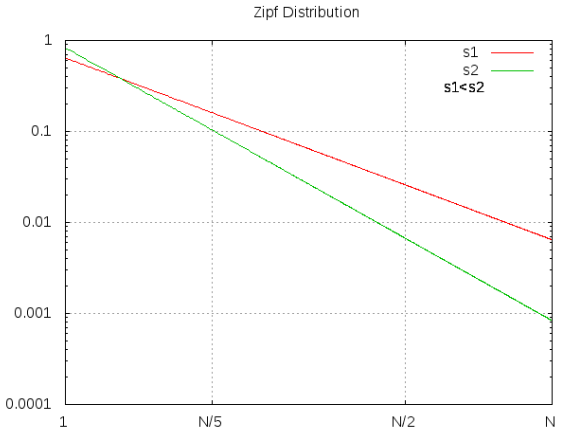
$$F_{UNIFORM}(x) = \begin{cases} \frac{1}{b-a}, & x \in [a, b] \\ 0 & x \notin [a, b] \end{cases}$$

  Η καμπύλη της $F_{UNIFORM}$ απεικονίζεται στο Σχήμα 4.1 (a). Τα σημεία a και b είναι για κάθε attribute τα άκρα του διαστήματος τιμών και δίνονται από τον Πίνακα 4.1.

- *Gauss* (ή κανονική) κατανομή, στην οποία οι τιμές γύρω από τη μέση τιμή της κατανομής εμφανίζονται με μεγαλύτερη πιθανότητα ενώ όσο οι τιμές απομακρύνονται από τη μέση τιμή εμφανίζονται με μικρότερη πιθανότητα. Βλέπουμε δηλαδή ότι στην κατανομή αυτή τα δεδομένα παρουσιάζουν μεγαλύτερη πόλωση σε σχέση με την προηγούμενη κατανομή και αναμένουμε το μεγαλύτερο πλήθος των τιμών κάθε



(a) (b)



(c)

Σχήμα 4.1: Συναρτήσεις Πυκνότητας Πιθανότητας των 3 κατανομών

attribute να βρίσκεται κοντά στο μέσο κάθε διαστήματος. Η συνάρτηση πυκνότητας πιθανότητας της κανονικής κατανομής είναι η

$$F_{GAUSS}(x) = \frac{1}{\sigma \cdot \sqrt{2\pi}} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

και η καμπύλη της απεικονίζεται στο Σχήμα 4.1 (b). Ο όρος μ είναι η μέση τιμή της κατανομής (συμβολίζεται με m στο Σχήμα) και ο όρος σ είναι η τυπική απόκλιση (συμβολίζεται με s στο Σχήμα).

- *Zipf* κατανομή, στην οποία οι τιμές γύρω από το 0 παρουσιάζουν τη μεγαλύτερη πιθανότητα εμφάνισης ενώ όσο οι τιμές βρίσκονται μακρύτερα από το 0 η πιθανότητα εμφάνισης μειώνεται πολύ έντονα. Με την κατανομή αυτή τα δεδομένα αποκτούν μια πολύ εντονότερη πόλωση σε σχέση με τις δυο προηγούμενες κατανομές. Η συνάρτηση

πυκνότητας πιθανότητας της Zipf κατανομής είναι η

$$F_{ZIPF}(k) = \frac{1/k^s}{\sum_{i=1}^{N} 1/n^s}$$

και η καμπύλη της απεικονίζεται στο Σχήμα 4.1 (c) σε λογαριθμικούς άξονες. Ο συντελεστής N είναι το πλήθος των δεδομένων ενώ ο συντελεστής s είναι η τιμή του εκθετικού που χαρακτηρίζει την κατανομή. Στο σχήμα βλέπουμε πως μετακινείται η καμπύλη για διαφορετικές τιμές του s. Επειδή μεγαλύτερες τιμές του s συνεπάγονται μεγαλύτερη πόλωση της κατανομής, χρησιμοποιούμε την τιμή 2 για να επιτύχουμε έντονη πόλωση (σε σχέση με την κανονική κατανομή) αλλά ταυτόχρονα πόλωση που δίνει μεγαλύτερες πιθανότητες εμφάνισης σε περισσότερες τιμές στα attributes των tuples.

Να σημειώσουμε ότι αν και στα παραπάνω σχήματα απεικονίζονται συνεχείς γραφικές παραστάσεις, οι τιμές των attributes είναι διακριτές.

Τέλος, ένα εξίσου σημαντικό χαρακτηριστικό των δεδομένων είναι το μέγεθός τους. Επειδή θέλουμε η κατανεμημένη ανωνυμοποίηση να είναι αποτελεσματική για ένα οσοδήποτε μεγάλο dataset, τα δεδομένα που θα χρησιμοποιηθούν θέλουμε να είναι αρκετά πολλά έτσι ώστε να ελέγξουμε αν το σύστημα μπορεί να τα διαχειριστεί. Έτσι λοιπόν, θα χρησιμοποιήσουμε δεδομένα διαφορετικών μεγεθών που περιέχουν από 1 εκατομμύριο tuples μέχρι 100 εκατομμύρια tuples. Στον παρακάτω πίνακα παραθέτουμε τα μεγέθη των δεδομένων που χρησιμοποιήθηκαν σε εκατομμύρια tuples και σε bytes.

Μέγεθος	Μέγεθος (MB)		
(millions of tuples)	Uniform	Gauss	Zipf
1	40	40.5	32.7
5	200	202.5	163.5
10	400	405	327
50	2000	2025	1635
100	4000	4050	3270

Πίνακας 4.2: Συγκεντρωτικός πίνακας μεγέθους δεδομένων

Το πιο μικρό dataset (1 million tuples) έχει μέγεθος 40 MB και το πιο μεγάλο dataset (100 million tuples) έχει μέγεθος περίπου 4GB. Με τη βοήθεια των δεδομένων αυτών θα μελετήσουμε τον τρόπο με τον οποίο επηρεάζεται ο χρόνος εκτέλεσης αλλά και η απόδοση της κατανεμημένης ανωνυμοποίησης με την μεταβολή κάποιων παραμέτρων του συστήματος.

4.2 Αξιολόγηση απόδοσης αλγορίθμων

Πριν παρουσιάσουμε τις εκτελέσεις των κατανεμημένων αλγορίθμων, εισάγουμε μια συνάρτηση μέτρησης με την οποία μπορούμε να μετρήσουμε την ποιότητα της ανωνυμοποίησης, μετρώντας ουσιαστικά το ποσοστό της πληροφορίας που χάθηκε λόγω της γενίκευσης των tuples. Η μετρική αυτή, στην οποία αναφερθήκαμε και σε προηγούμενα κεφάλαια (Κεφάλαιο 3.1.2, Κεφάλαιο 3.2.4) ονομάζεται Global Certainty Penalty (GCP) και υπολογίζεται ως εξής:

$$GCP(P) = \frac{\sum_{G \in P} |G| \cdot NCP(G)}{d \cdot N}$$

όπου NCP είναι το Normalized Certainty Penalty και ισχύει:

$$NCP^{(i)}(G) = \frac{\max_G^{(i)} - \min_G^{(i)}}{\max^{(i)} - \min^{(i)}}$$

$$NCP(G) = \sum_{i \in QID} w_i NCP^{(i)}(G)$$

Το NCP εκφράζει τη σχέση που έχει το εύρος τιμών μιας κλάσης ισοδυναμίας (G) με το εύρος τιμών του συνολικού πίνακα δεδομένων, σε όλες τις διαστάσεις για τις οποίες εκτελείται η ανωνυμοποίηση. Στην ουσία, το NCP αποτελεί μια έκφραση της πληροφορίας που εμπεριέχει μια κλάση ισοδυναμίας σε σχέση πάντα με το σύνολο των δεδομένων: όσο πιο διάσπαρτα είναι τα tuples μέσα στην κλάση ισοδυναμίας, τα ranges των attributes μεγαλώνουν με αποτέλεσμα η κλάση ισοδυναμίας να γενικεύεται όλο και περισσότερο. Στην ακραία περίπτωση που τα όρια της κλάσης ισοδυναμίας συμπίπτουν με τα όρια των δεδομένων, η κλάση αυτή έχει γενικευθεί πλήρως, αφού δεν μπορεί να μας δώσει καμία συγκεκριμένη πληροφορίας για τα μέλη της. Έτσι λοιπόν, γίνεται σαφές ότι όσο αποδοτικότερη είναι η ανωνυμοποίηση, τόσο πλησιέστερα βρίσκονται και τα tuples που ανήκουν σε κάθε κλάση ισοδυναμίας και τελικά, τόσο χαμηλότερες είναι και οι ενδείξεις NCP για τις κλάσεις.

Το GCP είναι ο δείκτης που τελικά μας δείχνει την ποιότητα της ανωνυμοποίησης, αφού για τον υπολογισμό του χρειάζονται οι τιμές NCP όλων των κλάσεων ισοδυναμίας. Όσο υψηλότερη είναι η τιμή του GCP τόσο περισσότερη πληροφορία έχει χαθεί σε σχέση με τον αρχικό πίνακα δεδομένων και συνεπώς, τόσο χειρότερη είναι η ανωνυμοποίηση που εφαρμόστηκε. Σε αντίθετη περίπτωση, όσο χαμηλότερη είναι η τιμή του GCP, τόσο πιο αποδοτική είναι η ανωνυμοποίηση που εφαρμόστηκε.

Ένα πολύ λεπτό σημείο στην χρησιμοποίηση του GCP ως δείκτη για την ποιότητα της ανωνυμοποίησης είναι η χρήση του δείκτη για έναν αλγόριθμο συγκριτικά με κάποιον άλλο. Αυτό σημαίνει ότι μια τιμή GCP δεν μπορεί να μας πληροφορήσει αν ο αλγόριθμος ανωνυμοποίησης που εφαρμόστηκε είναι αποδοτικός ή όχι, αφού σε κάθε περίπτωση θα ανα-

μένουμε να έχουμε απώλεια πληροφορίας κατά την ανωνυμοποίηση. Αν όμως συγκρίνουμε τους δείκτες GCP δυο διαφορετικών αλγορίθμων, μπορούμε να αποφανθούμε ποιος είναι ο αποδοτικότερος, αφού σε αυτήν την περίπτωση θα προτιμήσουμε εκείνον με τον οποίο χάθηκε λιγότερη πληροφορία.

Τέλος, ένα πολύ σημαντικό κομμάτι της απόδοσης ενός κατανεμημένου αλγορίθμου που δεν πρέπει να παραλειφθεί, είναι ο χρόνος εκτέλεσή του. Επειδή οι αλγόριθμοι ανωνυμοποίησης που παρουσιάσαμε έχουν έντονο υπολογιστικό φορτίο, θέλουμε ο καλύτερος (ποιοτικά) αλγόριθμος να μην είναι πολύ πιο αργός από τον χειρότερο (ποιοτικά). Αυτό σημαίνει ότι ο συνολικά προτιμότερος αλγόριθμος δεν είναι απαραίτητα εκείνος που κατά την εφαρμογή του χάνεται η λιγότερη πληροφορία, αλλά εκείνος που επιτυγχάνει τον καλύτερο συμβιβασμό ανάμεσα στον χρόνο εκτέλεσης και την απώλεια πληροφορίας.

Στις εκτελέσεις που θα παρουσιάσουμε θα δίνουμε μεγάλη έμφαση στο δείκτη GCP των δυο αλγορίθμων και στον χρόνο που χρειάστηκε για να ολοκληρωθεί η ανωνυμοποίηση.

4.3 Κεντρική εκτέλεση και απόδοση αλγορίθμων

Στο σημείο αυτό θα χρησιμοποιήσουμε τα δυο μικρά datasets που αναφέραμε προηγουμένως για την κεντρική εκτέλεση των αλγορίθμων και στη συνέχεια θα αυξήσουμε τον αριθμό των partitions για να εξετάσουμε πως επηρεάζεται η απόδοση της ανωνυμοποίησης όσο αυξάνεται ο αριθμός των partitions. Αρχικά παρουσιάζουμε τις εκτελέσεις για το τεχνητό dataset (smallData) και στη συνέχεια παρουσιάζουμε το πραγματικό dataset (Adults).

Για κάθε dataset και για κάθε αριθμό partition θα εκτελεστούν τρεις αλγόριθμοι ανωνυμοποίησης: ο αλγόριθμος TopDown, ο αλγορίθμος Mondrian που εκτελεί strict partitioning και ο αλγόριθμος Mondrian που εκτελεί relaxed partitioning. Παράλληλα, μπορούμε να εφαρμόσουμε και μια τέταρτη, προφανή μέθοδο ανωνυμοποίησης, η οποία δεν χρησιμοποιεί κάποιον αλγόριθμο, αλλά μετά την διαμέριση του συνόλου με βάση τα tuples-τομές που περιγράψαμε σε προηγούμενο κεφάλαιο, χωρίζει τα ταξινομημένα tuples σε ομάδες των k στοιχείων με τη σειρά που τα συναντά. Είναι προφανές ότι αυτή η μέθοδος αναμένουμε να είναι η χειρότερη όλων διότι δεν χρησιμοποιείται καμία αλγοριθμική τεχνική και το μόνο partitioning που πραγματοποιείται στα δεδομένα οφείλεται στον partitioner. Η μέθοδος αυτή που δεν στηρίζεται σε κάποιον αλγόριθμο, θα αποκαλείται στο εξής ως NoAlgorithm.

Ακόμη, για να εκτιμήσουμε τη λειτουργία του partitioner και τη συμβολή του στην ανωνυμοποίηση, δημιουργούμε έναν δεύτερο partitioner ο οποίος για να κάνει την διαμέριση του αρχικού πίνακα δεδομένων δε στηρίζεται σε tuples-τομές, αλλά την πραγματοποιεί με τυχαίο τρόπο. Συγκεκριμένα, οι mappers του partitioner διαβάζουν τα tuples των αρχείων εισόδου και κάθε mapper στέλνει κάθε tuple που διαβάζει σε διαφορετικό reducer κυκλικά. Όταν συμπληρωθεί ο αριθμός των reducer τα tuples ξαναρχίζουν από τον πρώτο reducer. Με αυτόν τον τρόπο τα tuples στέλνονται χωρίς κανένα κριτήριο στα partitions.

Ο partitioner αυτός μπορεί να αξιοποιηθεί συγκριτικά με τον SampleBased Partitioner που περιγράψαμε προηγουμένως για να μας δείξει πόσο καλύτερη είναι η μέθοδος partitioning που αναπτύξαμε σε σχέση με μια τυχαία, απλή μέθοδο.

4.3.1 Dataset smallData

Αρχικά εξετάζουμε το συνθετικό dataset smallData. Δοκιμάζουμε να εκτελέσουμε τους αλγορίθμους ανωνυμοποίησης με κεντρικό τρόπο (ή ισοδύναμα για 1 partition) και με κατανεμημένο τρόπο (για διάφορα partitions). Στους Πίνακες 4.3 και 4.4 παραθέτουμε τις τιμές GCP όλων των μεθόδων ανωνυμοποίησης, όταν χρησιμοποιείται ο Sample Based και ο Random partitioner αντίστοιχα.

Number of partitions	**Mondrian (relaxed)**	**Mondrian (strict)**	**NoAlgorithm**	**TopDown**
1	0.197625	0.187451	0.495595	0.242837
20	0.221432	0.221913	0.495770	0.284108
40	0.257514	0.257676	0.495639	0.330884
60	0.309482	0.286083	0.494605	0.362298
80	0.297538	0.298112	0.495869	0.385638
100	0.288795	0.297639	0.496052	0.408389

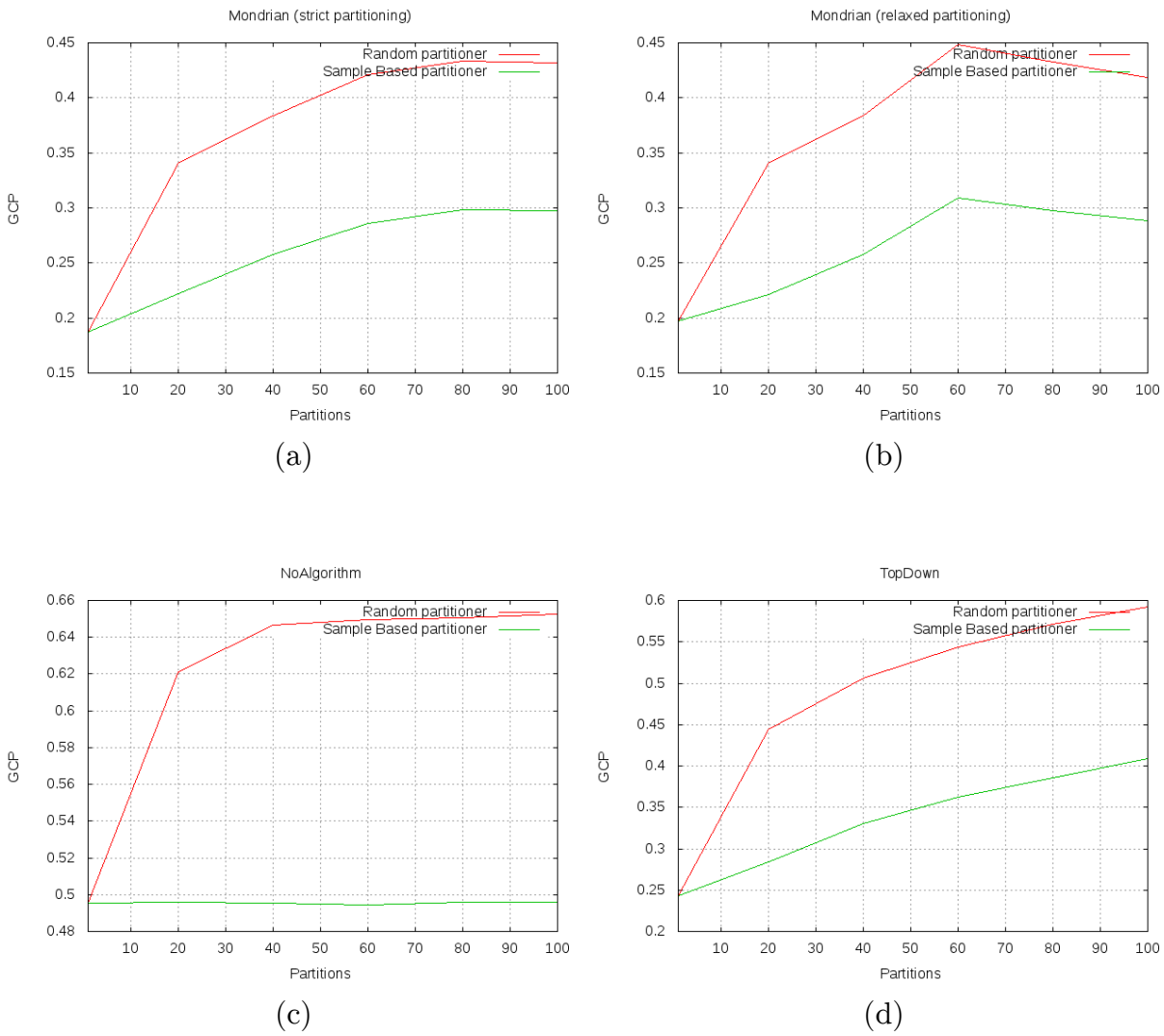
Πίνακας 4.3: smallData dataset με Sample Based Partitioner

Number of partitions	**Mondrian (relaxed)**	**Mondrian (strict)**	**NoAlgorithm**	**TopDown**
1	0.197625	0.187451	0.495595	0.242837
20	0.340689	0.340204	0.621441	0.444019
40	0.383597	0.383658	0.646505	0.506601
60	0.447960	0.420841	0.649481	0.543582
80	0.432790	0.433393	0.650560	0.570850
100	0.418802	0.431841	0.652703	0.591858

Πίνακας 4.4: smallData dataset με Random partitioner

Με τη βοήθεια των Πινάκων 4.3 και 4.4 μπορούμε να κατασκευάσουμε τις γραφικές παραστάσεις των Σχημάτων 4.2 και 4.3. Στο Σχήμα 4.2 παραθέτουμε μια σύγκριση για κάθε μέθοδο ανωνυμοποίησης όταν χρησιμοποιείται ο Sample Based partitioner και όταν χρησιμοποιείται ο Random partitioner. Παρατηρούμε σε όλες τις μεθόδους ότι η αύξηση του αριθμού των partitions αυξάνει το GCP συνεπώς η ποιότητα της ανωνυμοποίησης μειώνεται με την αύξηση του αριθμού των partitions.

Παράλληλα, η αύξηση του αριθμού των partitions όταν χρησιμοποιείται ο Sample Based partitioner βλέπουμε ότι αυξάνει το δείκτη GCP σε όλους τους αλγορίθμους. Αντίστοιχα, η αύξηση του αριθμού των partitions όταν πραγματοποιείται με το Random partitioner, επίσης αυξάνει το δείκτη GCP και μάλιστα αρκετά πιο έντονα σε σχέση με τον Sample



Σχήμα 4.2: Αλγόριθμοι με SampleBased και Random partitioner

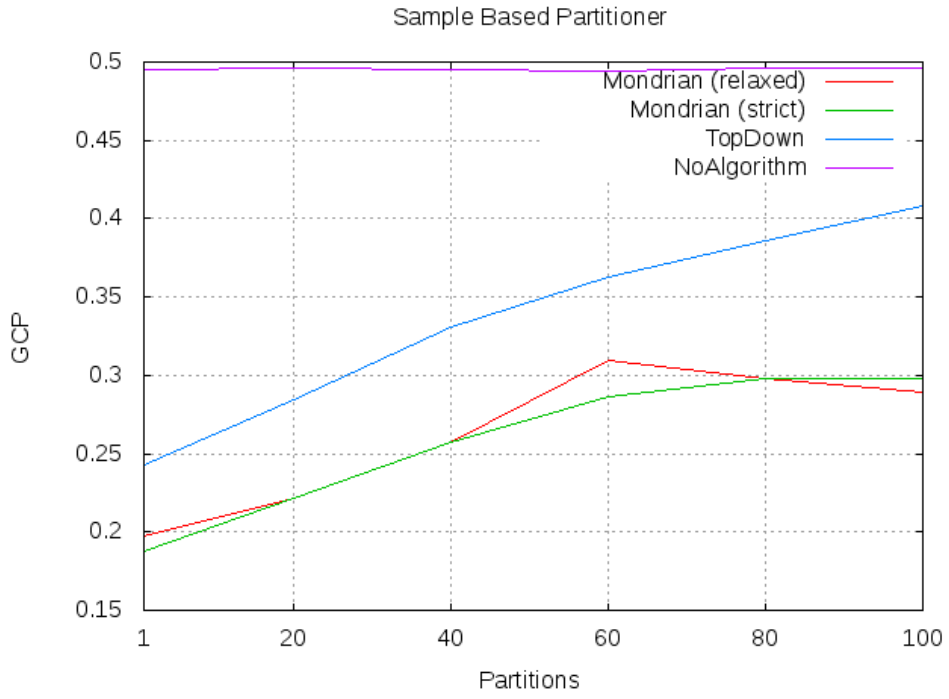
Based partitioner. Η μόνη μέθοδος που μένει πρακτικά ανεπηρέαστη από την αύξηση του αριθμού των partition είναι η μέθοδος NoAlgorithm, η οποία δεν επηρεάζεται τόσο έντονα από τον αριθμό των partitions όσο από τη διάταξη των tuples μέσα στα partitions, που παραμένει ίδια στην περίπτωση του Sample Based partitioner.

Η αύξηση του δείκτη GCP με την αύξηση των partitions είναι ένα φαινόμενο που οφείλεται στην παραλληλοποίηση του προβλήματος. Όταν ένας αλγόριθμος ανωνυμοποίησης γνωρίζει όλα τα δεδομένα του χώρου τότε μπορεί να λάβει καλύτερες αποφάσεις για τη διαμέρισή τους σε υποομάδες. Σε περίπτωση όμως που εμφανίζεται partitioning (όπως εδώ) πριν την εκτέλεση του αλγορίθμου, ο κάθε αλγόριθμος που εκτελείται έχει πληροφορίες μόνο για τα δεδομένα που ανήκουν στο partition του και, συνεπώς, η συνολική απόδοση του συστήματος μειώνεται. Παρ'όλα αυτά, σε κάποιες περιπτώσεις η μείωση της απόδοσης που προκαλείται από το partitioning μπορεί να θεωρηθεί αρκετά μικρή σε σχέση με τα πλεονεκτήματα που προσφέρει (γρήγορη και παράλληλη εκτέλεση, κοκ). Για παράδειγμα, αν συγκρίνουμε τις κεντρικές εκτελέσεις των αλγορίθμων Mondrian και TopDown (για ένα

partition) και τις κατανεμημένες εκτελέσεις με 20 partitions θα δούμε ότι η μείωση στην απόδοση της ανωνυμοποίησης είναι της τάξης του 18.4% για τον αλγόριθμο Mondrian (strict partitioning), 12% για τον αλγόριθμο Mondrian (relaxed partitioning) και 17% για τον αλγόριθμο TopDown. Σε κάποιες εφαρμογές της ανωνυμοποίησης, τα παραπάνω ποσοστά είναι ικανοποιητικά, εάν η κατανεμημένη εκτέλεση σημαίνει μικρή αύξηση της απώλειας πληροφορίας, αλλά παράλληλα μεγάλη μείωση στο χρόνο εκτέλεσης.

Να σημειώσουμε ότι στη γενική περίπτωση το ποσοστό μείωσης της απόδοσης των αλγορίθμων από την κατανεμημένη εκτέλεση εξαρτάται άμεσα από τα δεδομένα και τα χαρακτηριστικά τους. Σε κάποιες περιπτώσεις η μείωση στην απόδοση μπορεί να είναι αρκετά μεγάλη για να είναι ανεκτή, σε κάποιες άλλες περιπτώσεις μπορεί η μείωση στην απόδοση να είναι ικανοποιητική (όπως εδώ) και μπορεί ακόμα σε κάποιες περιπτώσεις η κατανεμημένη εκτέλεση να προκαλεί αύξηση στην απόδοση των μεθόδων ανωνυμοποίησης (όπως θα δούμε παρακάτω).

Τέλος, στο Σχήμα 4.3 βλέπουμε μια σύγκριση όλων των μεθόδων ανωνυμοποίησης που χρησιμοποιήσαμε. Οι μέθοδος NoAlgorithm που δεν εκτελεί κανέναν αλγόριθμο έχει τη χειρότερη απόδοση (χάνει περίπου τη διπλάσια πληροφορία σε σχέση με τις άλλες μεθόδους), η μέθοδος TopDown είναι η αμέσως καλύτερη, με το μειονέκτημα όμως ότι η αύξηση των partition αυξάνει αρκετά γρήγορα το δείκτη GCP και οι καλύτερες μέθοδος είναι οι μέθοδοι Mondrian που εκτελούν relaxed και strict partitioning.



Σχήμα 4.3: Σύγκριση αλγορίθμων για Sample Based Partitioner

Μια σημαντική παρατήρηση που πρέπει να κάνουμε για το παραπάνω σχήμα αφορά τη μείωση του δείκτη GCP για τη μέθοδο Mondrian (relaxed partitioning). Βλέπουμε ότι

η μέθοδος από 60 partitions και περισσότερα παρουσιάζει μείωση στο GCP. Το γεγονός αυτό είναι τυχαίο και μπορεί να συμβεί σε διάφορες περιπτώσεις. Υπάρχει δηλαδή περίπτωση η αύξηση των partitions να προκαλέσει μείωση του δείκτη GCP (αύξηση δηλαδή της ποιότητας ανωνυμοποίησης) και οφείλεται στον Partitioner και στις τομές των partitions και τα διάφορα χαρακτηριστικά των δεδομένων.

4.3.2 Dataset adults

Στη συνέχεια, εξετάζουμε το πραγματικό dataset Adults[13]. Τα εξεταζόμενα attributes του συγκεκριμένου dataset είναι 15 και τα ranges των attributes είναι αρκετά διαφορετικά (ποικίλουν από 2 μέχρι 1.5 εκατομμύριο τιμές). Εκτελούμε τις τέσσερις μεθόδους ανωνυμοποίησης που συναντήσαμε προηγουμένως κεντρικά (1 partition) και κατανεμημένα για partitions που έχουν δημιουργηθεί από τον Sample Based και τον Random partitioner, όπως και πριν. Στους πίνακες 4.5 και 4.6 παραθέτουμε τις τιμές GCP των μεθόδων ανωνυμοποίησης όταν χρησιμοποιείται ο Sample Based και ο Random partitioner αντίστοιχα.

Number of partitions	**Mondrian (relaxed)**	**Mondrian (strict)**	**NoAlgorithm**	**TopDown**
1	0.301303	0.145583	0.127943	0.136782
20	0.190099	0.127490	0.127614	0.129947
40	0.183148	0.127162	0.128026	0.129579
60	0.199586	0.126511	0.128025	0.128461
80	0.174675	0.128016	0.127526	0.129694
100	0.152949	0.128361	0.127346	0.129823
120	0.194936	0.128884	0.128141	0.129531
140	0.179721	0.128075	0.127458	0.129969
160	0.168079	0.128706	0.127556	0.130220
180	0.156041	0.128619	0.127324	0.129727
200	0.148546	0.130578	0.127921	0.129748

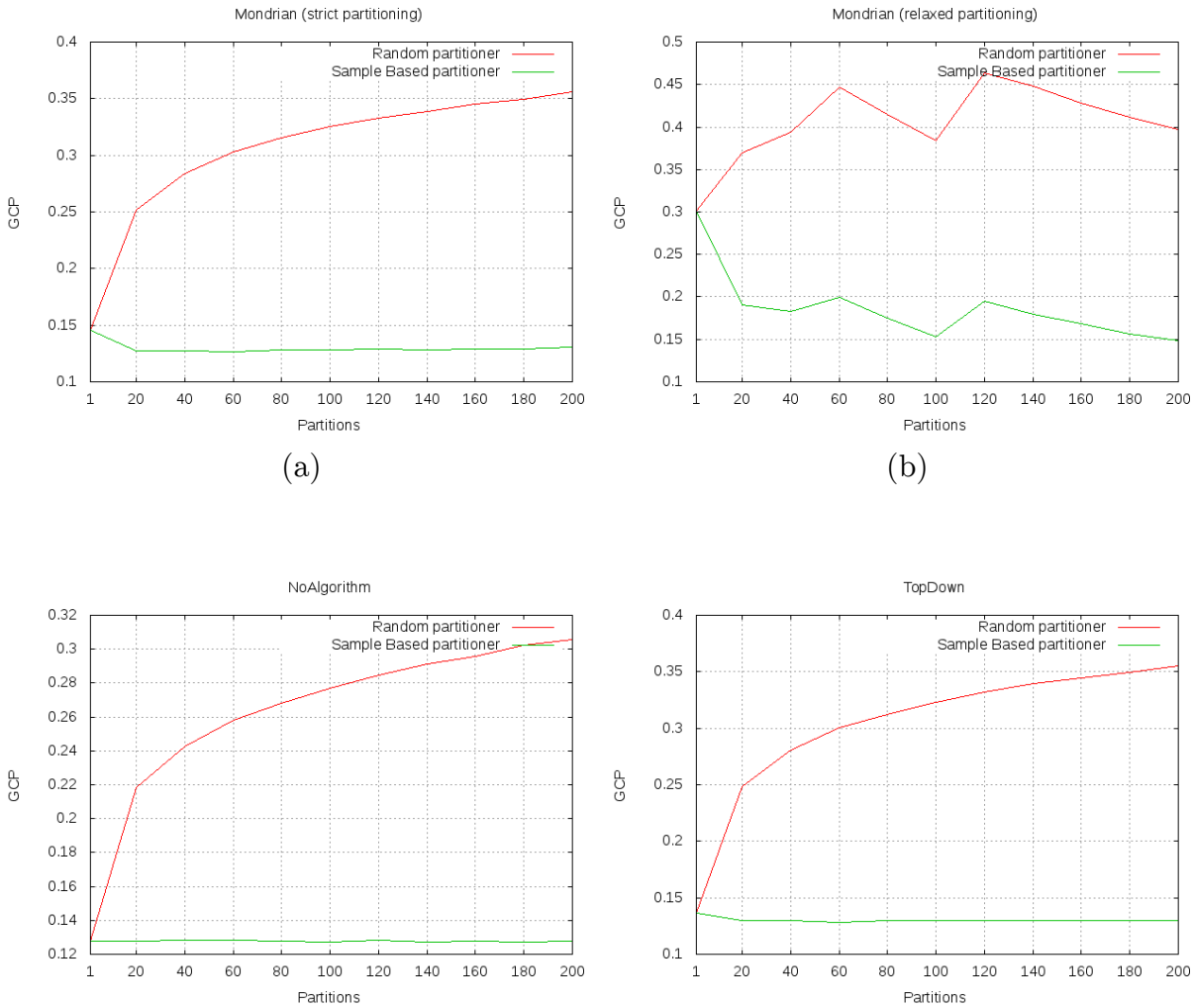
Πίνακας 4.5: Adult dataset για Sample Based Partitioner

Με βάση τις τιμές των πινάκων κατασκευάζουμε τις αντίστοιχες γραφικές παραστάσεις για κάθε αλγόριθμο στο Σχήμα 4.4. Στις εικόνες (a)-(d) παραθέτουμε τους δείκτες GCP όταν χρησιμοποιείται ο Sample Based και ο Random partitioner για τους αλγορίθμους Mondrian (strict), Mondrian (relaxed), NoAlgorithm και TopDown αντίστοιχα.

Από τις γραφικές παραστάσεις του Σχήματος 4.4 λαμβάνουμε μια διαφορετική εικόνα σε σχέση με τις αντίστοιχες γραφικές παραστάσεις του προηγούμενου, συνθετικού dataset. Και εδώ, όπως και προηγουμένως, ο Random partitioner μειώνει την απόδοση της ανωνυμοποίησης, όσο αυξάνονται τα partitions που δημιουργεί ανεξαρτήτως αλγορίθμου ανωνυμοποίησης. Με τη χρησιμοποίηση του sample based partitioner όμως, όσο αυξάνεται ο αριθμός των partitions όλες οι μέθοδοι ανωνυμοποίησης δείχνουν να βελτιώνουν την απόδοσή τους. Το συμβάν αυτό οφείλεται στην ποιότητα των δεδομένων. Επειδή το dataset

Number of partitions	Mondrian (relaxed)	Mondrian (strict)	NoAlgorithm	TopDown
1	0.301303	0.145583	0.127943	0.136782
20	0.369941	0.251815	0.218606	0.248598
40	0.393424	0.284017	0.242567	0.280972
60	0.446463	0.302827	0.258157	0.300834
80	0.414794	0.315640	0.268187	0.312330
100	0.384130	0.325671	0.277140	0.322871
120	0.464023	0.332648	0.284455	0.332374
140	0.447657	0.338771	0.291250	0.339703
160	0.428129	0.345665	0.295618	0.344793
180	0.411901	0.349837	0.302429	0.349315
200	0.397200	0.356270	0.305896	0.355656

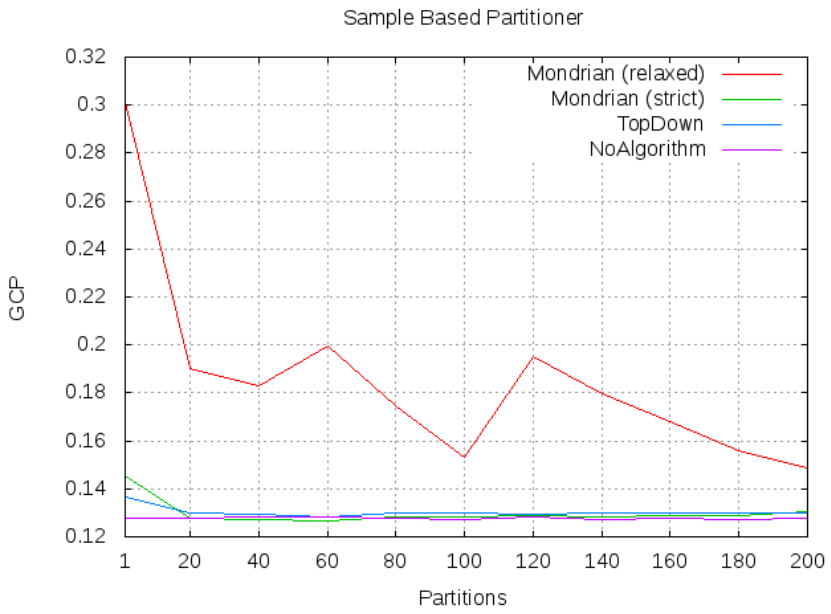
Πίναχας 4.6: Adult dataset για Random Partitioner



Σχήμα 4.4: Αλγόριθμοι για SampleBased και Random partitioner

που εξετάζουμε τώρα περιέχει tuples που τοποθετούνται σε ένα χώρο αρκετά περισσότερων διαστάσεων (και τα ranges των attributes είναι αρκετά μεγαλύτερα από πριν) ο partitioner με το διαμοιρασμό των tuples λειτουργεί ευεργετικά στην ανωνυμοποίηση γιατί μειώνει διαστάσεις από το QID, κάνει μια πρώτη κατηγοριοποίηση των tuples ως προς τα attributes και δίνει τη δυνατότητα στον αλγόριθμο να διαχωρίσει tuples που βρίσκονται σε πιο κοντινές αποστάσεις σε σχέση με το σύνολο των tuples. Για το λόγο αυτό βλέπουμε ότι όλες οι μέθοδοι όσο αυξάνονται τα partitions λειτουργούν αποδοτικότερα, με μόνη εξαίρεση την NoAlgorithm που, όπως είδαμε και προηγουμένως, δεν επηρεάζεται τόσο έντονα από την αύξηση των partitions και μένει πρακτικά σταθερή η απόδοσή της. Αν φυσικά συνεχίσουμε να αυξάνουμε τον αριθμό των partitions (σε σημείο που το μέγεθος του partition είναι συγκρίσιμο με το k) τότε η απόδοση των μεθόδων θα μειωθεί αρκετά γιατί σε αυτήν την περίπτωση η διαμέριση του partitioner θα σταματήσει να λειτουργεί ωφέλιμα για τις μεθόδους ανωνυμοποίησης και θα αρχίσει να συμβάλει αρνητικά στη λειτουργία τους.

Τέλος, με τη βοήθεια του Πίνακα 4.5 συγκρίνουμε τις αποδόσεις των μεθόδων ανωνυμοποίησης όταν χρησιμοποιείται ο Sample Based Partitioner. Στο Σχήμα 4.5 παραθέτουμε τις αντίστοιχες γραφικές παραστάσεις.



Σχήμα 4.5: Σύγκριση αλγορίθμων για Sample Based Partitioner

Ένα πολύ ενδιαφέρον στοιχείο της παραπάνω γραφικής παράστασης είναι η απόδοση της μεθόδου NoAlgorithm που υπενθυμίζουμε ότι δεν χρησιμοποιεί κάποιον αλγόριθμο ανωνυμοποίησης, αλλά δημιουργεί μια κλάση ισοδυναμίας όταν συναντά k tuples στα ταξινομημένα partitions. Στην ουσία δηλαδή, η μέθοδος αυτή είναι ενδεικτική για τη συμβολή του Partitioner στην ανωνυμοποίηση. Βλέπουμε λοιπόν ότι στο συγκεκριμένο dataset ο Partitioner παίζει αρκετά σημαντικό ρόλο στην τελική ανωνυμοποίηση, αφού η πιο αποδοτική μέθοδος ανωνυμοποίησης είναι η NoAlgorithm. Δεν είναι τυχαίο μάλιστα ότι όλες

οι υπόλοιπες μέθοδοι ανωνυμοποίησης συγκλίνουν στην τιμή GCP που έχει η μέθοδος NoAlgorithm, αφού όσο περισσότερο διαμερίζει το χώρο του προβλήματος ο partitioner, τόσο πιο αποδοτικά μπορούν να λειτουργήσουν οι αλγόριθμοι.

Να τονίσουμε ότι και σε αυτήν την περίπτωση, αν συνεχίσουμε να αυξάνουμε τον αριθμό των partitions η ποιότητα ανωνυμοποίησης σταδιακά θα αρχίσει να μειώνεται, επιβεβαιώνοντας έτσι την γενική περίπτωση που διατυπώσαμε στην προηγούμενη ενότητα. Παρ'όλα αυτά, ένα πολύ ενδιαφέρον συμπέρασμα που μπορεί να προκύψει από τις εκτελέσεις των δυο datasets που παρουσιάσαμε είναι το εξής: η λειτουργία του partitioner συμβάλει στην ανωνυμοποίηση των δεδομένων. Ο βαθμός της συμβολής του εξαρτάται σε πολύ μεγάλο βαθμό από τον τύπο και την ποιότητα των δεδομένων. Με κατάλληλα δεδομένα, η κατανεμημένη εκτέλεση ενός αλγορίθμου ανωνυμοποίησης μπορεί να είναι αποδοτικότερη από την αντίστοιχη κεντρική εκτέλεση (πχ, Adults dataset), ενώ σε άλλες περιπτώσεις η συμβολή του Partitioner στην ανωνυμοποίηση είναι αρνητική (η κεντρική εκτέλεση δηλαδή, να είναι αποδοτικότερη της κατανεμημένης όπως είδαμε στο προηγούμενο συνθετικό dataset). Κοινό χαρακτηριστικό και των δύο περιπτώσεων είναι ότι η αύξηση του αριθμού των partitions σε μεγάλο βαθμό (όταν δηλαδή το πλήθος των tuples του partition γίνεται συγκρίσιμο με το k) λειτουργεί αρνητικά στην ανωνυμοποίηση, αυξάνοντας το δείκτη GCP οποιασδήποτε μεθόδου ανωνυμοποίησης. Τελικά δηλαδή, οι παραπάνω γραφικές παραστάσεις θα αυξηθούν ανεξάρτητα από τα δεδομένα που χρησιμοποιούν.

4.4 Κατανεμημένη Εκτέλεση

Κατά την κατανεμημένη εκτέλεση των αλγορίθμων, υπάρχει μια πληθώρα μεταβλητών που μπορούν να επηρεάσουν την απόδοση και τη γενική λειτουργία του κατανεμημένου συστήματος. Αυτές οι μεταβλητές είναι:

- ο αριθμός των partitions στα οποία χωρίζονται τα δεδομένα,
- οι διαστάσεις στο QID,
- ο συντελεστής k,
- ο αριθμός των κόμβων του cluster,
- το μέγεθος και η κατανομή των δεδομένων.

Στη συνέχεια παρουσιάζουμε τις εκτελέσεις των κατανεμημένων αλγορίθμων για διάφορες τιμές των παραπάνω μεταβλητών.

4.4.1 Μεταβολή πλήθους partition

Στο σημείο αυτό, θα χρησιμοποιήσουμε το συνθετικό dataset που περιγράψαμε στην προηγούμενη ενότητα, όταν τα δεδομένα ακολουθούν Uniform κατανομή και ο πίνακας

δεδομένων αποτελείται από πέντε εκατομμύρια tuples. Με τη βοήθεια των δεδομένων, θα εκτελέσουμε τις τέσσερις μεθόδους ανωνυμοποίησης που χρησιμοποιήσαμε στις προηγούμενες, κεντρικές εκτελέσεις και θα μεταβάλλουμε τον αριθμό των partitions για να δούμε πως επηρεάζεται η ποιότητα και ο χρόνος εκτέλεσης της ανωνυμοποίησης για το συγκεκριμένο dataset.

Αρχικά, εκτελούμε την κατανεμημένη εφαρμογή για ένα μικρό πλήθος από partitions και σταδιακά αυξάνουμε το πλήθος τους. Το cluster στο οποίο πραγματοποιούμε τις εκτελέσεις αποτελείται από 6 κόμβους, κάθε ένας από τους οποίους διαθέτει 16 task slots (8 map slots και 8 reduce slots). Η τιμή k θεωρούμε ότι είναι ίση με 10, ο ρυθμός δειγματοληψίας είναι ίσος με 20% και το QID αποτελείται από όλα τα attributes (1-10). Τα δεδομένα είναι οργανωμένα σε 10 αρχεία εισόδου που αποτελούνται από 500 χιλιάδες tuples. Όλες οι παράμετροι παραμένουν σταθερές καθόλη τη διάρκεια των εκτελέσεων. Με τις παραπάνω σημειώσεις υπόψη, παραθέτουμε τα αποτελέσματα των δεικτών GCP για κάθε μέθοδο ανωνυμοποίησης.

Number of partitions	**Mondrian (relaxed)**	**Mondrian (strict)**	**NoAlgorithm**	**TopDown**
100	0.180326	0.178698	0.369348	0.225607
250	0.188960	0.185366	0.369291	0.225262
500	0.192296	0.188486	0.369306	0.228920
750	0.194757	0.192065	0.369319	0.235503
1000	0.201147	0.197674	0.369380	0.242262
1250	0.216136	0.208767	0.369323	0.248458
1500	0.206315	0.204067	0.369314	0.253892
1750	0.197236	0.202743	0.369320	0.258749
2000	0.215185	0.211567	0.369250	0.263120

Πίνακας 4.7: Δείκτες GCP για διάφορα πλήθη partition

Με τη βοήθεια του Πίνακα 4.7 δημιουργούμε την γραφική παράσταση που απεικονίζεται στο Σχήμα 4.6, στο οποίο απεικονίζουμε την ποιότητα της ανωνυμοποίησης (όπως μετράται με τη βοήθεια του GCP) συναρτήσει του πλήθους των partitions.

Όπως βλέπουμε από το Σχήμα 4.6, όσο μικρότερος είναι ο αριθμός των partition τόσο αποδοτικότερα λειτουργούν οι αλγόριθμοι ανωνυμοποίησης. Η μόνη μέθοδος που κρατά σταθερή την απόδοσή της είναι η NoAlgorithm, που όπως είδαμε και προηγουμένως δεν επηρεάζεται έντονα από αλλαγές στον αριθμό των partitions. Η απόδοση της μεθόδου NoAlgorithm είναι η χειρότερη από όλες τις υπόλοιπες μεθόδους, αφού εμφανίζει έναν δείκτη GCP που είναι σχεδόν διπλάσιος από τον δείκτη GCP της καλύτερη μεθόδου. Η μέθοδος TopDown είναι η αμέσως καλύτερη μέθοδος μετά από τη NoAlgorithm, με αρκετά χαμηλότερο δείκτη GCP και παρουσιάζει σχεδόν γραμμική αύξηση με την αύξηση του αριθμού των partitions. Τέλος, οι μέθοδοι που εκτελούν τον αλγόριθμο Mondrian και στις δυο περιπτώσεις partitioning παρουσιάζουν την καλύτερη απόδοση από όλες τις

Σχήμα 4.6: Γραφική παράσταση GCP για διάφορα πλήθη partition

μεθόδους. Μεταξύ των δυο πολιτικών διαμέρισης, το strict partitioning παρουσιάζει ελαφρώς καλύτερες τιμές GCP σε σχέση με το relaxed, με μόνη εξαίρεση κάποια σημεία στα οποία υπάρχει μια αυξομείωση της απόδοσης της μεθόδου που οφείλεται στα δεδομένα και στο partitioning, όπως είδαμε και σε προηγούμενη ενότητα. Σε όλες τις περιπτώσεις, καταλήγουμε στο συμπέρασμα που είχαμε διατυπώσει στις κεντρικές εκτελέσεις που παρουσιάσαμε προηγουμένως: η συνεχής αύξηση του αριθμού των partitions καταλήγει σε αύξηση του δείκτη GCP και μείωση της ποιότητας ανωνυμοποίησης.

Όπως βλέπουμε από το παραπάνω σχήμα, όλοι οι αλγόριθμοι λειτουργούν καλύτερα για τον μικρότερο δυνατό αριθμό partitions. Για να αποφασίσουμε, όμως, για τον αριθμό partition που θα επιλέξουμε για τις επόμενες εκτελέσεις πρέπει, πέρα από την ελαχιστοποίηση του GCP, να λάβουμε υπόψη δυο ακόμη πολύ σημαντικές παραμέτρους: πρώτον, πρέπει ο αριθμός των partition που θα επιλέξουμε να συνδυάζει εκτός από υψηλή απόδοση ανωνυμοποίησης και μικρό χρόνο εκτέλεσης και δεύτερον, πρέπει να λάβουμε υπόψη και τους φυσικούς περιορισμούς που έχουμε από το cluster, σχετικά με τον μέγιστο αριθμό mappers που μπορούμε να εκτελούμε στην ίδια Map Reduce εργασία. Πειραματικά, βρήκαμε ότι ο μέγιστος αριθμός για το cluster που χρησιμοποιούμε ανέρχεται σε 10000 tasks. Στην προκειμένη περίπτωση, ο περιορισμός αυτός δεν μας επηρεάζει, αλλά όπως θα δούμε και αργότερα, όταν αυξάνεται το πλήθος των δεδομένων ο περιορισμός αυτός είναι αρκετά σημαντικός.

Για την εύρεση λοιπόν του βέλτιστου αριθμού partition εξετάζουμε τους χρόνους εκτέλεσης των MapReduce εργασιών που εκτελέστηκαν. Για κάθε μέγεθος partition, εκτελούνται 6 MapReduce jobs: εκτελείται μια εργασία που κάνει δειγματοληψία (Sampler), μια εργασία που πραγματοποιεί το partitioning και την ταξινόμηση (Sample Based Partitioner)

και οι τέσσερις εργασίες που εκτελούν τις μεθόδους ανωνυμοποίησης. Στους Πίνακες που ακολουθούν παραθέτουμε τους χρόνους εκτέλεσης για το Sampler και τον Partitioner (Πίνακας 4.8), τους χρόνους εκτέλεσης των τεσσάρων μεθόδων ανωνυμοποίησης καθώς και το συνολικό χρόνο εκτέλεσης όλων των εργασιών για κάθε αριθμό partition (Πίνακας 4.9). Με τη βοήθεια των Πινάκων αυτών κατασκευάζουμε τις γραφικές παραστάσεις που απεικονίζονται στο Σχήμα 4.7. Στην γραφική παράσταση (a) απεικονίζεται ο χρόνος εκτέλεσης του Sampler και του Partitioner, στην παράσταση (b) απεικονίζεται ο χρόνος εκτέλεσης των μεθόδων ανωνυμοποίησης και στην γραφική παράσταση (c) απεικονίζεται ο συνολικός χρόνος εκτέλεσης όλων των εργασιών. Σημειώνουμε ότι οι χρόνοι εκτέλεσης απεικονίζονται στη μορφή "ώρες:λεπτά:δευτερόλεπτα".

Number of partitions	**Sampler**	**Sample Based Partitioner**
100	00:12:04	00:05:54
250	00:11:54	00:05:54
500	00:12:00	00:07:45
750	00:12:18	00:09:40
1000	00:11:29	00:11:47
1250	00:12:30	00:13:47
1500	00:11:46	00:15:47
1750	00:11:39	00:17:49
2000	00:11:21	00:20:00

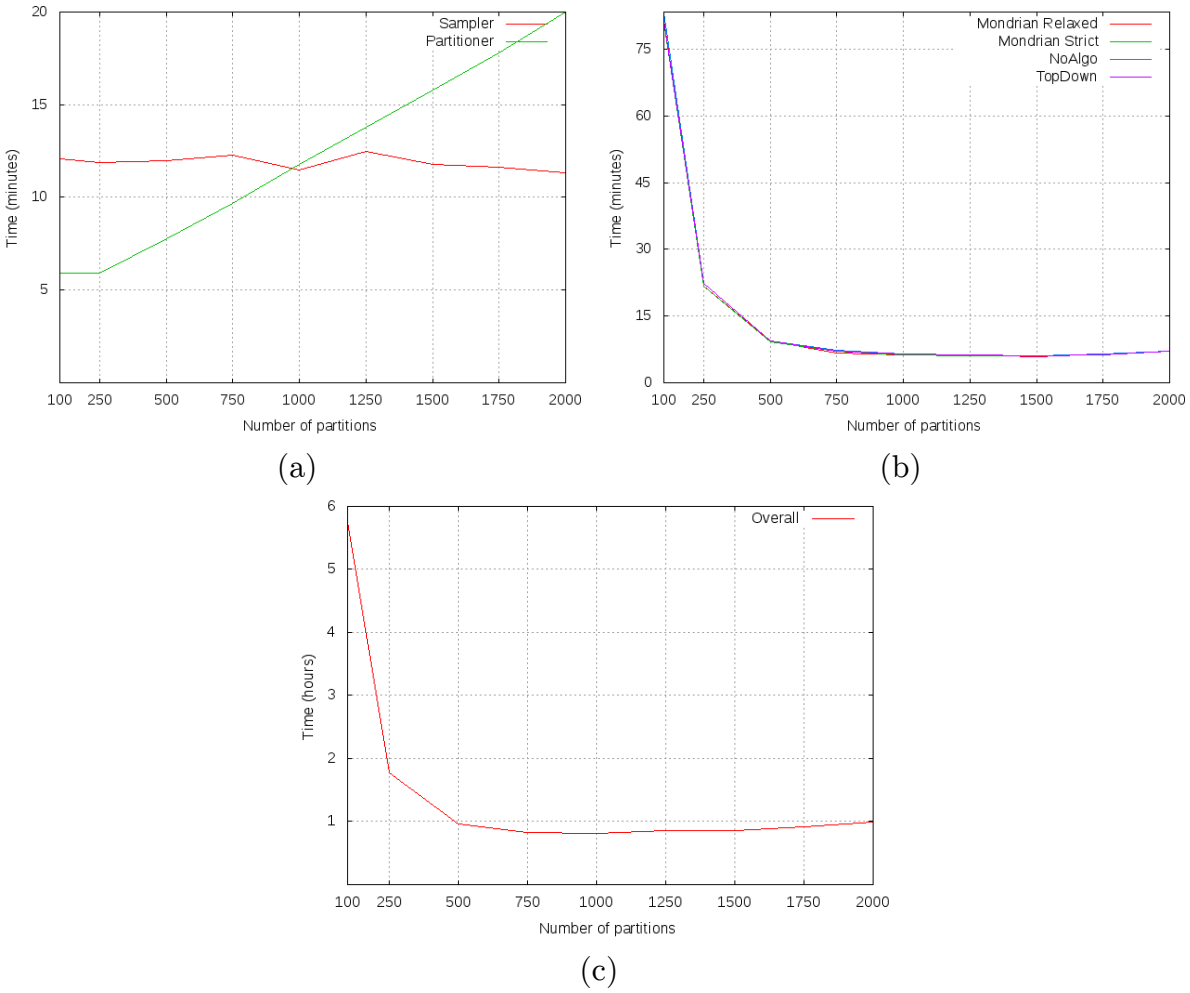
Πίνακας 4.8: Χρόνοι εκτέλεσης Sampler, Partitioner για διάφορα πλήθη partition

Number of partitions	**Mondrian (relaxed)**	**Mondrian (strict)**	**NoAlgorithm**	**TopDown**	**Overall**
100	01:21:31	01:20:56	01:23:05	01:21:11	05:44:41
250	00:21:47	00:21:48	00:22:23	00:22:18	01:46:04
500	00:09:30	00:09:14	00:09:25	00:09:30	00:57:24
750	00:06:47	00:07:00	00:07:13	00:07:05	00:50:03
1000	00:06:18	00:06:20	00:06:29	00:06:25	00:48:48
1250	00:06:16	00:06:04	00:06:21	00:06:20	00:51:18
1500	00:05:54	00:06:03	00:06:04	00:06:04	00:51:38
1750	00:06:34	00:06:31	00:06:34	00:06:22	00:55:29
2000	00:07:04	00:07:02	00:07:04	00:07:03	00:59:34

Πίνακας 4.9: Χρόνοι εκτέλεσης μεθόδων ανωνυμοποίησης για διάφορα πλήθη partition

Στη γραφική παράσταση 4.7 (a) βλέπουμε ότι ο χρόνος εκτέλεσης του Sampler παραμένει ίδιος, ανεξαρτήτως του αριθμού των partition ενώ ο χρόνος εκτέλεσης του Partitioner αυξάνει γραμμικά με το πλήθος των partition. Ο Sampler δεν επηρεάζεται από τον αριθμό των partitions, αφού σε κάθε περίπτωση θα διαβάσει όλα τα tuples από τα αρχεία εισόδου, θα εκτελέσει τη δειγματοληψία και στη συνέχεια θα επιλέξει τις κατάλληλες τομές για τα

partitions. Οι παράμετροι δηλαδή που επηρεάζουν τον χρόνο εκτέλεσής του είναι το πλήθος των tuples των αρχείων εισόδου καθώς και ο ρυθμός δειγματοληψίας. Ο Partitioner αντίθετα, επηρεάζεται άμεσα από το πλήθος των partitions, αφού θα δημιουργήσει τόσους reducer όσα και τα partitions και κάθε tuple της εισόδου θα συγκριθεί με τις τομές, που είναι ίσες με τον αριθμό των partitions πλην ένα. Για αυτό το λόγο βλέπουμε ότι η αύξηση των partition αυξάνει με γραμμικό τρόπο το χρόνο εκτέλεσής του.



Σχήμα 4.7: Γραφικές παραστάσεις χρόνων εκτέλεσης εργασιών για διάφορα πλήθη partition

Στη γραφική παράσταση 4.7 (b) βλέπουμε ότι ο χρόνος εκτέλεσης όλων των μεθόδων ανωνυμοποίησης είναι πρακτικά ίδιος. Όλες οι μέθοδοι μειώνουν το χρόνο εκτέλεσής τους με την αύξηση των partitions, διότι η αύξηση του αριθμού των partitions οδηγεί σε μείωση του μεγέθους του κάθε partition, αφού το dataset παραμένει το ίδιο. Αυτό σημαίνει ότι κάθε εκτελούμενη μέθοδος, έχει ως είσοδο λιγότερα δεδομένα και, ως αποτέλεσμα, μειώνεται ο χρόνος εκτέλεσης της. Παρατηρούμε ότι η αύξηση των partitions από ένα σημείο και μετά (για 750 ή 1000 partitions) δεν συνεχίζει να μειώνει το χρόνο εκτέλεσης. Αυτό συμβαίνει γιατί η μείωση του μεγέθους του partition ενώ μειώνει το χρόνο εκτέλεσης κάθε αλγορίθμου, αυξάνει παράλληλα τον αριθμό των tasks που απαιτούνται για την εκτέλεση των μεθόδων. Σε εκείνο το σημείο δηλαδή, υπάρχει ένα overhead που έχει σχέση με τη

διαχείριση των tasks και περαιτέρω αύξηση του αριθμού των partitions θα οδηγήσει τελικά σε αύξηση του χρόνου εκτέλεσης γιατί το overhead αυτό γίνεται σημαντικά μεγαλύτερο από την πραγματική εκτέλεση των αλγορίθμων. Να υπενθυμίσουμε ότι κάθε task στο hadoop είναι μια διεργασία που δημιουργείται σε κάποιο κόμβο του cluster και διαχειρίζεται από το master κόμβο. Αν αυξηθεί πολύ ο αριθμός των tasks (όπως συμβαίνει με την αύξηση των partitions) τότε το κόστος διαχείρισης της κάθε διεργασίας γίνεται αρκετά μεγάλο και παίζει σημαντικό ρόλο στο χρόνο ολοκλήρωσης ενός MapReduce job. Η αρχική αύξηση, δηλαδή στον αριθμό των partitions μειώνει το χρόνο εκτέλεσης γιατί σε κάθε αλγόριθμο (που όπως είδαμε δεν έχει γραμμική πολυπλοκότητα) μειώνεται το μέγεθος της εισόδου αλλά η συνεχιζόμενη αύξηση (πάνω από 2000 partitions που απεικονίζονται στη γραφική παράσταση) των partitions αυξάνει τελικά το χρόνο εκτέλεσης γιατί το κόστος διαχείρισης των tasks γίνεται σημαντικό.

Τέλος, στη γραφική παράσταση 4.7 (c) απεικονίζεται ο συνολικός χρόνος εκτέλεσης όλων των διεργασιών. Στην ουσία, η γραφική παράσταση προκύπτει από το κατακόρυφο άρθοισμα όλων των υπόλοιπων γραφικών παραστάσεων των σχημάτων (a) και (b). Στο πρώτο τμήμα της γραφικής (από 100 μέχρι 750 partitions περίπου) ο χρόνος εκτέλεσης οφείλεται στο χρόνο εκτέλεσης των μεθόδων. Στο δεύτερο τμήμα (από 750 μέχρι 1250 περίπου) ο χρόνος εκτέλεσης παραμένει σταθερός και ελάχιστος και στο τρίτο τμήμα (από 1500 μέχρι 2000) αρχίζει η σταδιακή αύξηση του συνολικού χρόνου εκτέλεσης, που οφείλεται στον αυξανόμενο χρόνο εκτέλεσης του Partitioner και ταυτόχρονα το στάσιμο χρόνο εκτέλεσης των μεθόδων ανωνυμοποίησης.

Συμπερασματικά, μελετώντας τους χρόνους εκτέλεσης και την απόδοση των εργασιών, θέλουμε να επιλέξουμε το μικρότερο δυνατό αριθμό partition (για να έχουμε τη μεγαλύτερη δυνατή ποιότητα ανωνυμοποίησης) και παράλληλα θέλουμε στον επιλεγμένο αριθμό partition, η εκτέλεση να πραγματοποιείται σε ένα σχετικά σύντομο χρονικό διάστημα. Αν λοιπόν επιλέξουμε ο αριθμός των partition να είναι ίσος με 250, τότε η μέγιστη απώλεια σε GCP που έχουμε στις μεθόδους ανωνυμοποίησης κυμαίνεται περίπου στο 4.5% αλλά ταυτόχρονα ο χρόνος εκτέλεσης υποτετραπλασιάζεται από την περίπτωση που έχουμε 100 partitions. Αν επιλέξουμε αριθμό partition 500 τότε η μέγιστη απώλεια σε GCP από τις μεθόδους γίνεται 6% αλλά ο χρόνος εκτέλεσης μειώνεται κατά 8 φορές σε σχέση με τον αριθμό partition ίσο με 100. Συνεπώς, επιλέγουμε τον αριθμό partition να είναι ίσος με 500 (περίπου 10000 tuples/partition).

4.4.2 Μεταβολή QID

Στη συνέχεια, θα εξετάσουμε την επίδραση του QID στην απόδοση και το χρόνο εκτέλεσης της ανωνυμοποίησης. Θα διατηρήσουμε όλες τις παραμέτρους σταθερές, όπως και πριν και θα μεταβάλλουμε μόνο τα attributes που ανήκουν στο QID. Συγκεκριμένα, ο αριθμός των partition θα είναι ίσος με 500 (ή ισοδύναμα, το μέγεθος του κάθε partition

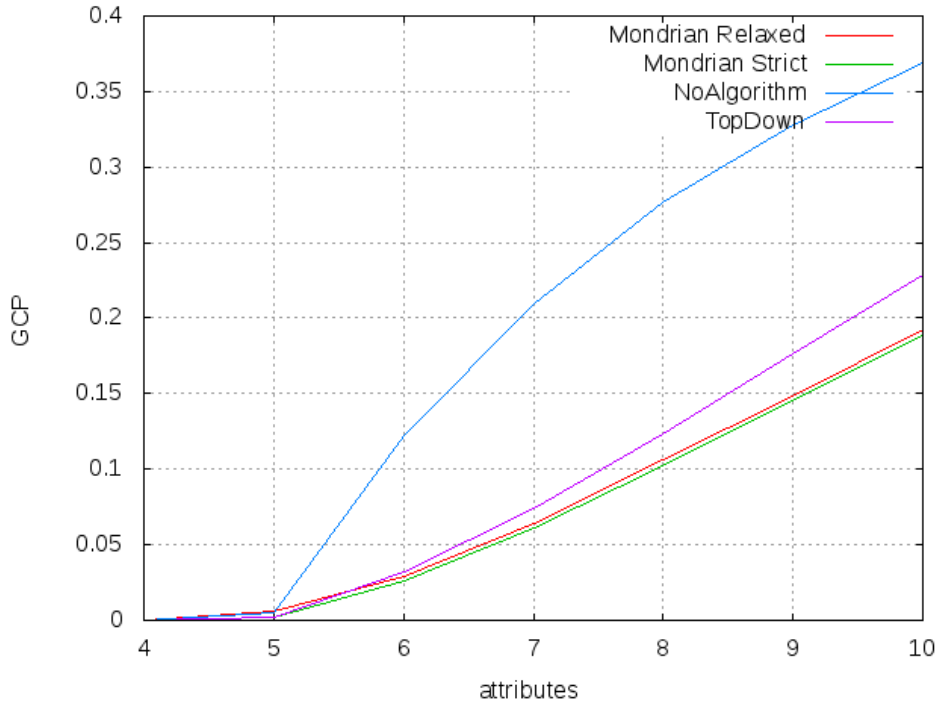
είναι περίπου 10000 tuples) και το k θα είναι ίσο με 10. Για την εκτέλεση των μεθόδων, χρησιμοποιούμε τα ίδια δεδομένα με προηγουμένως (5 εκατομμύρια tuples που ακολουθούν Uniform κατανομή), τα οποία έχουν χωριστεί από τον Sample Based partitioner σε 500 partitions, αφού πρώτα έγινε δειγματοληψία στα δεδομένα με ρυθμό 20%. Τα ταξινομημένα αυτά δεδομένα θα χρησιμοποιηθούν για όλες τις εκτελέσεις χωρίς να συμμετέχει ο Partitioner εκ νέου για κάθε εκτέλεση.

Όπως παρουσιάσαμε στην αρχή του Κεφαλαίου, τα δεδομένα που χρησιμοποιούμε για τις κατανεμημένες εκτελέσεις έχουν 10 attributes κάθε ένα από τα οποία έχει συγκεκριμένο εύρος τιμών. Για να εξετάσουμε την επίδραση του αριθμού των attributes του QID στην απόδοση της κατανεμημένης εφαρμογής, σε κάθε εκτέλεση προσθέτουμε ένα attribute στο QID με σειρά σπουδαιότητας. Όπως δείξαμε και στον Πίνακα 4.1, τα attributes ταξινομημένα κατά σειρά σπουδαιότητας είναι τα εξής: $\{1, 3, 10, 2, 7, 6, 4, 9, 5, 8\}$. Η πρώτη εκτέλεση πραγματοποιείται για τα πρώτα 4 attributes ($\{1, 3, 10, 2\}$) και στη συνέχεια προστίθεται 1 attribute μέχρι να προστεθούν όλα. Στον πίνακα που ακολουθεί παραθέτουμε τις τιμές GCP που λάβαμε από τις τέσσερις μεθόδους ανωνυμοποίησης για κάθε QID. Επίσης, με τη βοήθεια του Πίνακα 4.10 κατασκευάζουμε την αντίστοιχη γραφική παράσταση για τις τέσσερις μεθόδους ανωνυμοποίησης που φαίνεται στο Σχήμα 4.8.

Number of attributes	**Mondrian (relaxed)**	**Mondrian (strict)**	**NoAlgorithm**	**TopDown**
4	0.000137	0.000004	0.000105	0.000001
5	0.005748	0.002406	0.004815	0.002288
6	0.028958	0.025826	0.122147	0.031685
7	0.064526	0.061423	0.209987	0.074036
8	0.105907	0.102570	0.276340	0.123752
9	0.148774	0.145355	0.327956	0.176739
10	0.192114	0.188707	0.369296	0.228708

Πίνακας 4.10: Δείκτες GCP για διάφορα QID

Όπως βλέπουμε από το Σχήμα 4.8, η αύξηση του αριθμού των attributes που συμμετέχουν στο QID αυξάνει το δείκτη GCP για όλες τις μεθόδους ανωνυμοποίησης. Το φαινόμενο αυτό είναι λογικό, γιατί η ύπαρξη λιγότερων attributes στο QID σημαίνει μείωση των διαστάσεων των δεδομένων και κατά συνέπεια οδηγεί στην απλοποίηση του προβλήματος. Για το λόγο αυτό βλέπουμε ότι όλοι οι αλγόριθμοι λειτουργούν πολύ καλύτερα για ένα μικρό αριθμό στοιχείων. Το φαινόμενο αυτό εξηγεί και το λόγο για τον οποίο ο Sample Based partitioner σε κάποιες περιπτώσεις έχει ευεργετική επίδραση στην ανωνυμοποίηση των δεδομένων, όπως είδαμε και στην Ενότητα 4.3.2. Συγκεκριμένα, η ταξινόμηση των tuples με βάση τη σπουδαιότητα των attributes οδηγεί σε partitions που μπορεί όλα τα tuples στα πρώτα τους attributes να έχουν ακριβώς τις ίδιες τιμές. Σε αυτήν την περίπτωση, οι διαστάσεις που έχουν τις ίδιες τιμές δεν επηρεάζουν την ανωνυμοποίηση, με αποτέλεσμα οι μέθοδοι ανωνυμοποίησης να εξετάζουν μόνο τα λιγότερα σημαντικά



Σχήμα 4.8: Γραφική παράσταση GCP για διάφορα QID

attributes των δεδομένων. Στην ουσία δηλαδή, αγνοούν τα πιο σημαντικά attributes, αφαιρώντας έτσι διαστάσεις από τα δεδομένα, με αποτέλεσμα τη βελτίωσης της ποιότητας της ανωνυμοποίησης[1].

Ακόμη, μια σημαντική πληροφορία που μπορούμε να λάβουμε από τη γραφική παράσταση αφορά τη συμπεριφορά των μεθόδων με την αύξηση των attributes του QID. Όπως βλέπουμε, η μέθοδος NoAlgorithm αυξάνεται με το μεγαλύτερο ρυθμό από τις υπόλοιπες μεθόδους. Η αμέσως καλύτερη μέθοδος είναι η TopDown (ενώ βλέπουμε ότι για 4 και 5 attributes η TopDown είναι η καλύτερη μέθοδος από όλες) και τέλος την καλύτερη απόδοση την έχουν οι μέθοδοι που υλοποιούν τον αλγόριθμο Mondrian. Η καλύτερη μέθοδος είναι εκείνη που υλοποιεί strict partitioning με πολύ μικρή διαφορά από τη μέθοδο που υλοποιεί relaxed partitioning.

Στη συνέχεια θα εξετάσουμε πως επηρεάζει ο αριθμός των attributes του QID του χρόνους εκτέλεσης των μεθόδων ανωνυμοποίησης. Στον Πίνακα 4.11 παραθέτουμε για κάθε QID το χρόνο εκτέλεσης για κάθε μέθοδο ανωνυμοποίησης.

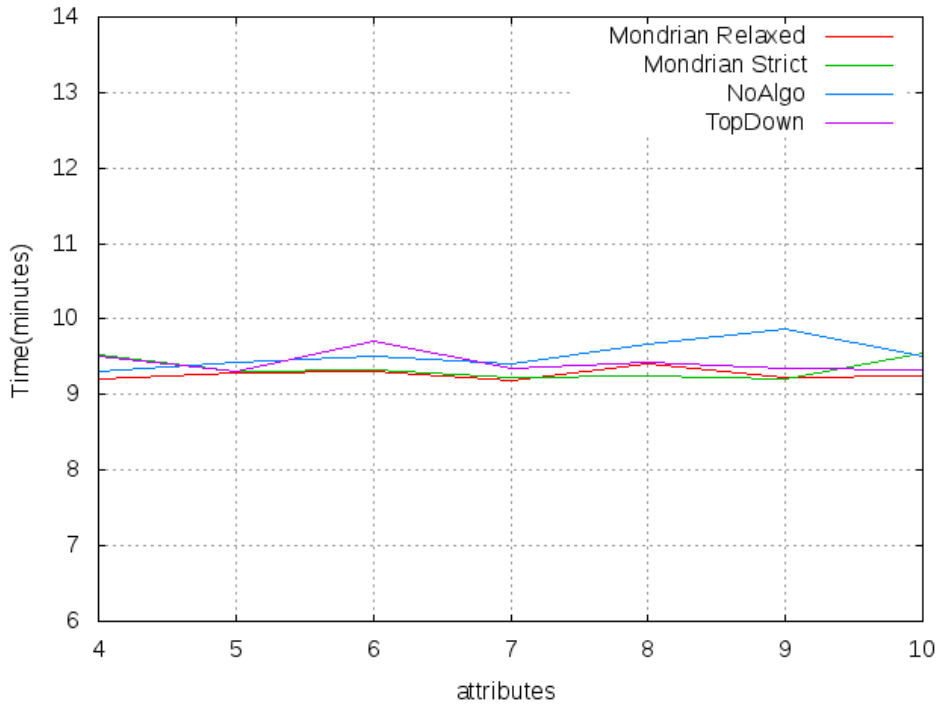
Με τη βοήθεια του Πίνακα 4.11 παρουσιάζουμε τη γραφική παράσταση που απεικονίζεται στο Σχήμα 4.9. Όπως μπορούμε να διαπιστώσουμε οι χρόνοι εκτέλεσης των μεθόδων ανωνυμοποίησης παραμένουν ίδιοι ανεξαρτήτως του αριθμού των attributes του QID. Αυτό οφείλεται στην πολύ μικρή επίδραση που έχει το q στη χρονική πολυπλοκότητα των μεθόδων ανωνυμοποίησης. Σε περίπτωση που το q λάμβανε μεγαλύτερες τιμές θα βλέπαμε

[1]Παρ'όλα αυτά, για το αν η επίδραση του Partitioner είναι θετική ή αρνητική έχει πολύ μεγάλη σημασία η ποιότητα και το είδος των δεδομένων.

Number of attributes	Mondrian (relaxed)	Mondrian (strict)	NoAlgorithm	TopDown
4	00:09:12	00:09:32	00:09:18	00:09:30
5	00:09:17	00:09:18	00:09:26	00:09:18
6	00:09:18	00:09:20	00:09:31	00:09:42
7	00:09:11	00:09:14	00:09:25	00:09:21
8	00:09:24	00:09:15	00:09:40	00:09:26
9	00:09:14	00:09:12	00:09:52	00:09:21
10	00:09:15	00:09:33	00:09:31	00:09:20

Πίνακας 4.11: Χρόνοι εκτέλεσης για διάφορα QID

μεγαλύτερη επίδρασή του στον τελικό χρόνο εκτέλεσης.



Σχήμα 4.9: Γραφική παράσταση χρόνων εκτέλεσης για διάφορα QID

4.4.3 Μεταβολή k

Στη συνέχεια, εξετάζουμε την επίδραση της μεταβλητής k στην ποιότητα και το χρόνο εκτέλεσης της ανωνυμοποίησης. Υπενθυμίζουμε ότι ο συντελεστής k δείχνει τον ελαχιστο αριθμό των tuples που θα πρέπει να υπάρχουν στις τελικές κλάσεις ισοδυναμίας των ανωνυμοποιημένων δεδομένων. Η αύξηση του αριθμού k επομένως σημαίνει ότι πιο πολλά tuples θα βρίσκονται σε λιγότερες κλάσεις ισοδυναμίας, πράγμα που σημαίνει ότι τα tuples γενικεύονται σε μεγαλύτερες ομάδες και χάνεται περισσότερη πληροφορία. Αυτό που θέλουμε να διαπιστώσουμε με τις συγκεκριμένες εκτελέσεις είναι πως αντιδρούν οι μέθοδοι

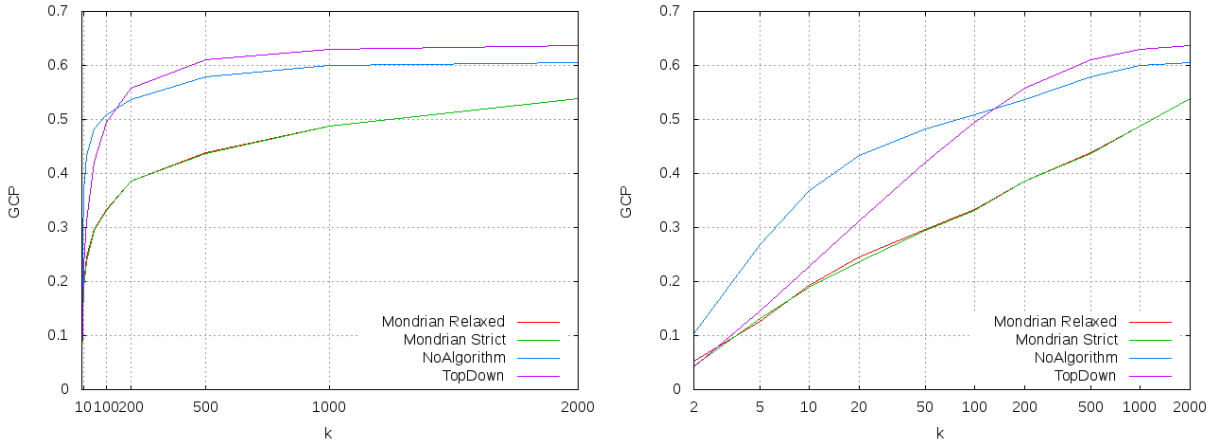
ανωνυμοποίησης με την αύξηση του k τόσο ως προς την ποιότητα ανωνυμοποίησης όσο και ως προς το χρόνο εκτέλεσης.

Για την εκτέλεση των μεθόδων ανωνυμοποίησης θα χρησιμοποιήσουμε τα ίδια ταξινομημένα δεδομένα με προηγουμένως (500 partitions τα οποία έχουν προκύψει από τα δεδομένα των 5 εκατομμυρίων tuples που ακολουθούν Uniform κατανομή και έχουν υποστεί δειγματοληψία με ρυθμό 20%), το QID θα περιέχει όλα τα attributes και θα μεταβάλλουμε μόνο την τιμή του k. Στον Πίνακα 4.12 παραθέτουμε τους δείκτες GCP όλων των μεθόδων ανωνυμοποίησης για διάφορες τιμές του k.

k	**Mondrian (relaxed)**	**Mondrian (strict)**	**NoAlgorithm**	**TopDown**
2	0.051963	0.044704	0.102955	0.042069
5	0.126663	0.131346	0.268382	0.144842
10	0.192114	0.188707	0.369296	0.228708
20	0.246366	0.237247	0.433472	0.312603
50	0.295883	0.294455	0.482010	0.420577
100	0.332710	0.331639	0.508305	0.495478
200	0.386292	0.385664	0.536190	0.557470
500	0.437797	0.437468	0.578763	0.610545
1000	0.488416	0.488312	0.600281	0.629593
2000	0.538601	0.538525	0.605836	0.636398

Πίνακας 4.12: Δείκτες GCP για διάφορες τιμές k

Με τη βοήθεια του παραπάνω πίνακα αναπαριστούμε τους δείκτες GCP στις γραφικές παραστάσεις του Σχήματος 4.10. Στον κατακόρυφο άξονα αναπαρίσταται η τιμή του δείκτη GCP και στον οριζόντιο άξονα αναπαρίσταται το k. Στο Σχήμα (a) ο οριζόντιος άξονας είναι γραμμικός και στο Σχήμα (b) ο άξονας είναι λογαριθμικός (για την καλύτερη απεικόνιση κάποιων λεπτομερειών για μικρές τιμές του k).



(a) (b)

Σχήμα 4.10: Γραφικές παραστάσεις GCP για διάφορες τιμές k

Αρχικά βλέπουμε ότι οι δείκτες GCP για όλες τις μεθόδους ανωνυμοποίησης αυξάνονται

με σχεδόν παρόμοιο τρόπο. Αρχικά, για μικρές τιμές του k, μικρές αυξήσεις του k οδηγούν σε πολύ μεγάλες αυξήσεις του GCP για όλες τις μεθόδους ανωνυμοποίησης. Ενδεικτικά, η αύξηση του k από 2 σε 10 αυξάνει περίπου 5.5 φορές το GCP για το TopDown, 4 φορές περίπου για τις μεθόδους Mondrian (relaxed και strict partitioning) και 3.5 φορές περίπου για τη μέθοδο NoAlgorithm. Αν συνεχίσουμε να αυξάνουμε το k παρατηρούμε ότι από ένα σημείο και μετά η αύξηση στο GCP σταματάει να είναι τόσο έντονη όσο ήταν για μικρά k και η κλίση του τμήματος της καμπύλης του GCP γίνεται σταδιακά μικρότερη. Το σημείο αυτό είναι διαφορετικό για κάθε μέθοδο ανωνυμοποίησης. Συγκεκριμένα, για τη NoAlgorithm αυτό συμβαίνει για k=100, ενώ για τις υπόλοιπες μεθόδους αυτό συμβαίνει για k=200[2]. Τέλος, αν συνεχίσουμε να αυξάνουμε ακόμα περισσότερο το k τότε ο ρυθμός αύξησης των δεικτών GCP για όλες τις γραφικές γίνεται ακόμα μικρότερος μέχρι που από κάποιο σημείο και έπειτα μηδενίζεται (πρακτικά). Για παράδειγμα, οι δείκτες GCP των μεθόδων NoAlgorithm και TopDown αυξάνονται ελάχιστα μετά το σημείο k=1000. Οι μέθοδοι Mondrian όταν εκτελούν strict και relaxed partitioning, αντίθετα, συνεχίζουν να αυξάνουν το GCP τους ακόμα και για μεγάλες τιμές k, αλλά παραμένουν πάντα σε χαμηλότερα επίπεδα από τις γραφικές των δυο άλλων μεθόδων ανωνυμοποίησης.

Ένα πολύ ενδιαφέρον συμπέρασμα που προκύπτει από τις εκτελέσεις που παρουσιάσαμε αφορά την καταλληλότητα του αλγορίθμου TopDown για περιπτώσεις που πρέπει να εκτελέσουμε ανωνυμοποίηση για μικρές και μεγάλες τιμές του k. Σε περιπτώσεις λοιπόν που το k είναι αρκετά χαμηλό και κατά την ανωνυμοποίηση που εκτελούμε θέλουμε να χαθεί ελάχιστη πληροφορία ο αλγόριθμος TopDown λειτουργεί καλύτερα συγκριτικά με τους άλλους αλγορίθμους. Αντίθετα, σε περιπτώσεις που η εκτελούμενη ανωνυμοποίηση θα οδηγήσει σε μεγάλες γενικεύσεις των δεδομένων, ο αλγόριθμος TopDown είναι χειρότερος του αλγορίθμου Mondrian και της μεθόδου NoAlgorithm.

Στη συνέχεια, εξετάζουμε την επίδραση που έχει η παράμετρος k στο χρόνο εκτέλεσης των αλγορίθμων ανωνυμοποίησης. Στον Πίνακα 4.13 παρουσιάζουμε τους χρόνους εκτέλεσης όλων των μεθόδων για διάφορες τιμές k. Με βάση τον πίνακα αυτό, κατασκευάζουμε τις γραφικές παραστάσεις που απεικονίζονται στο Σχήμα 4.11.

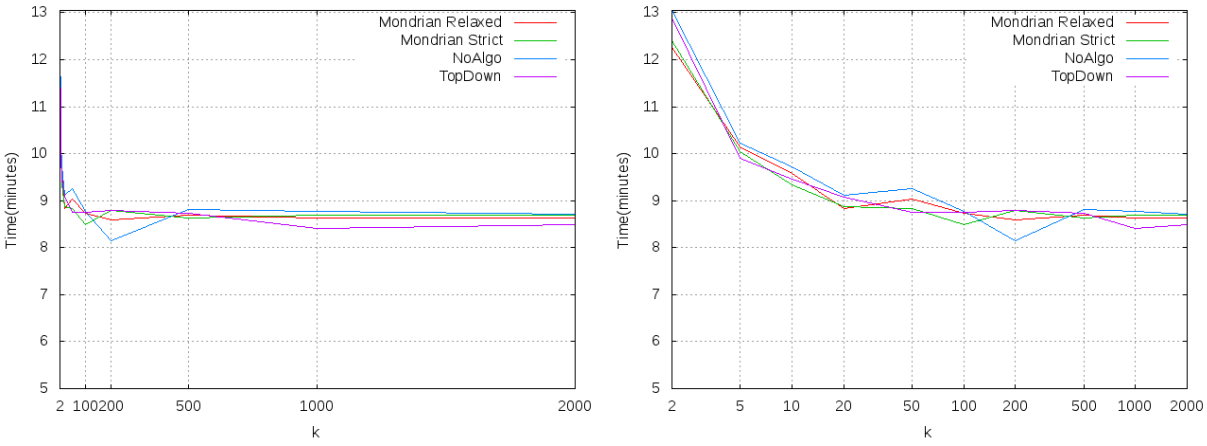
Στο Σχήμα 4.11 (a) απεικονίζεται στον κάθετο άξονα ο χρόνος (σε λεπτά) και στον οριζόντιο άξονα το k (σε γραμμική κλίμακα) και στο Σχήμα 4.11 (b) απεικονίζεται ο χρόνος στον οριζόντιο άξονα και το k σε λογαριθμική κλίμακα για να αποτυπωθεί με μεγαλύτερη λεπτομέρεια η συμπεριφορά των καμπυλών για μικρές τιμές του k.

Όπως βλέπουμε από τις γραφικές παραστάσεις στο διάστημα από k=2 μέχρι k=100 όλες οι μέθοδοι ανωνυμοποίησης συμπεριφέρονται με τον ίδιο τρόπο στην αύξηση του k αφού ο χρόνος εκτέλεσής τους φθίνει πολύ γρήγορα με την αύξηση του k. Αυτό είναι λογικό, διότι όπως είδαμε και στην παρουσίαση των αλγορίθμων Mondrian και TopDown ο συντελεστής

[2]Παρατηρούμε ότι στο ενδιάμεσο τμήμα για k από 100 μέχρι 200, η καμπύλη GCP της TopDown ξεπερνά την αντίστοιχη καμπύλη της NoAlgorithm.

k	Mondrian (relaxed)	Mondrian (strict)	NoAlgorithm	TopDown
2	00:12:16	00:12:24	00:13:03	00:12:53
5	00:10:09	00:10:03	00:10:13	00:09:54
10	00:09:35	00:09:20	00:09:43	00:09:27
20	00:08:50	00:08:53	00:09:07	00:09:05
50	00:09:02	00:08:50	00:08:43	00:08:45
100	00:08:44	00:08:29	00:08:46	00:08:45
200	00:08:36	00:08:48	00:08:09	00:08:47
500	00:08:41	00:08:38	00:08:49	00:08:44
1000	00:08:38	00:08:41	00:08:46	00:08:24
2000	00:08:38	00:08:41	00:08:43	00:08:30

Πίνακας 4.13: Χρόνοι εκτέλεσης για διάφορες τιμές k



(a) (b)

Σχήμα 4.11: Γραφικές παραστάσεις χρόνων εκτέλεσης για διάφορες τιμές k

k καθορίζει το ύψος του δέντρου των αναδρομών που θα εκτελέσει ο αλγόριθμος[3]. Όσο μεγαλύτερο είναι το k τόσο νωρίτερα θα ολοκληρωθεί ο αλγόριθμος. Για το λόγο αυτό βλέπουμε ότι η αύξηση του k οδηγεί σε ραγδαία μείωση του χρόνου εκτέλεσης όλων των μεθόδων.

Αν παρ'όλα αυτά το k συνεχίσει να αυξάνεται, οι μέθοδοι ανωνυμοποίησης δεν συνεχίζουν να μειώνουν το χρόνο εκτέλεσής του αλλά, από ένα σημείο και μετά ο χρόνος εκτέλεσης παραμένει σταθερός. Συγκεκριμένα, για τιμές k μεγαλύτερες από 100 οι μέθοδοι Mondrian και TopDown διατηρούν σταθερό το χρόνο εκτέλεσής τους, διότι το k έχει φτάσει σε πολύ μεγάλες τιμές και περαιτέρω αύξησή του δεν μειώνει αισθητά το χρόνο εκτέλεσης του αλγορίθμου, αφού το δέντρο αναδρομών ήδη έχει αρκετά μικρό ύψος[4] και

[3]Στη μέση περίπτωση και όταν τα δεδομένα δεν έχουν πολύ έντονη πόλωση στο χώρο, όπως εδώ, ένα από τα δύο σύνολα που παράγονται σε κάποιο βήμα του αλγορίθμου δεν θα έχει συνεχώς πολύ λίγα στοιχεία. Σε αυτή την περίπτωση, όπως έχουμε πει και κατά την παρουσίαση των αλγορίθμων η αύξηση του k οδηγεί σε γραμμική αύξηση του χρόνου εκτέλεσης των αλγορίθμων ανωνυμοποίησης.

[4]Για k=100 και partition size περίπου 10000 tuples το ύψος του δέντρου αναδρομών κάθε αλγορίθμου είναι περίπου ίσο με $\log \frac{10000}{100} \approx 6.64$.

η μείωση του ύψους του δέντρου δεν θα έχει αισθητή διαφορά στο χρόνο εκτέλεσης. Η μέθοδος NoAlgorithm όμως, που εκτελείται σε γραμμικό χρόνο (αφού διαβάζει τα ταξινομημένα δεδομένα και απλά δημιουργεί κλάσεις ισοδυναμίας k στοιχείων) και δεν λειτουργεί αναδρομικά βλέπουμε ότι εκτελείται σε παρόμοιο χρόνο με τις υπόλοιπες μεθόδους. Ο λόγος για τον οποίο συμβαίνει αυτό έχει σχέση με τον αριθμό των παραγόμενων κλάσεων ισοδυναμίας. Οι τελικές κλάσεις ισοδυναμίας που θα παράγουν οι μέθοδοι αναμένουμε να είναι περίπου $\frac{n}{k}$. Για κάθε κλάση ισοδυναμίας, ο mapper γράφει στην έξοδό του το NCP της κλάσης και το πλήθος των στοιχείων της. Στη συνέχεια ο reducer συγκεντρώνει τα NCP όλων των κλάσεων και βρίσκει το τελικό GCP. Αυτό σημαίνει ότι ο reducer για την εύρεση του GCP αθροίζει $\frac{n}{k}$ στοιχεία. Συνεπώς, από τη στιγμή που το πλήθος των tuples παραμένει σταθερό, η μείωση του k οδηγεί σε μείωση του αριθμού των στοιχείων που αθροίζονται. Η μείωση συνεχίζεται μέχρι κάποιο οριακό σημείο το οποίο καθορίζεται από τη συνολική λειτουργία της μεθόδου (πχ, εκτέλεση των mappers, κοκ). Από εκείνο το σημείο και μετά δεν συμβαίνει κάποια άλλη μείωση στο χρόνο εκτέλεσης γιατί η μεταβολή του k δεν επηρεάζει έντονα το συνολικό χρόνο εκτέλεσης, ο οποίος τώρα αποτελείται κυρίως από τις εργασίες που πραγματοποιούνται στους mappers και στην ομαδοποίηση των tuples σε ομάδες των k στοιχείων παρά στον reducer.

4.4.4 Μεταβολή μεγέθους cluster

Στη συνέχεια, εξετάζουμε τη συμπεριφορά των μεθόδων ανωνυμοποίησης σε περιπτώσεις που αλλάζει ο αριθμός των κόμβων του cluster. Για δεδομένα εισόδου χρησιμοποιούμε τα ίδια με προηγουμένως (500 partitions ταξινομημένων δεδομένων, που προήλθαν από το dataset των 5 εκατομμυρίων tuples μετά από δειγματοληψία με ρυθμό 20%), ο συντελεστής k είναι ίσος με 10 και το QID περιέχει όλα τα attributes. Υπενθυμίζουμε επίσης ότι κάθε κόμβος του cluster έχει 16 διαθέσιμα task slots (8 map slots και 8 reduce slots).

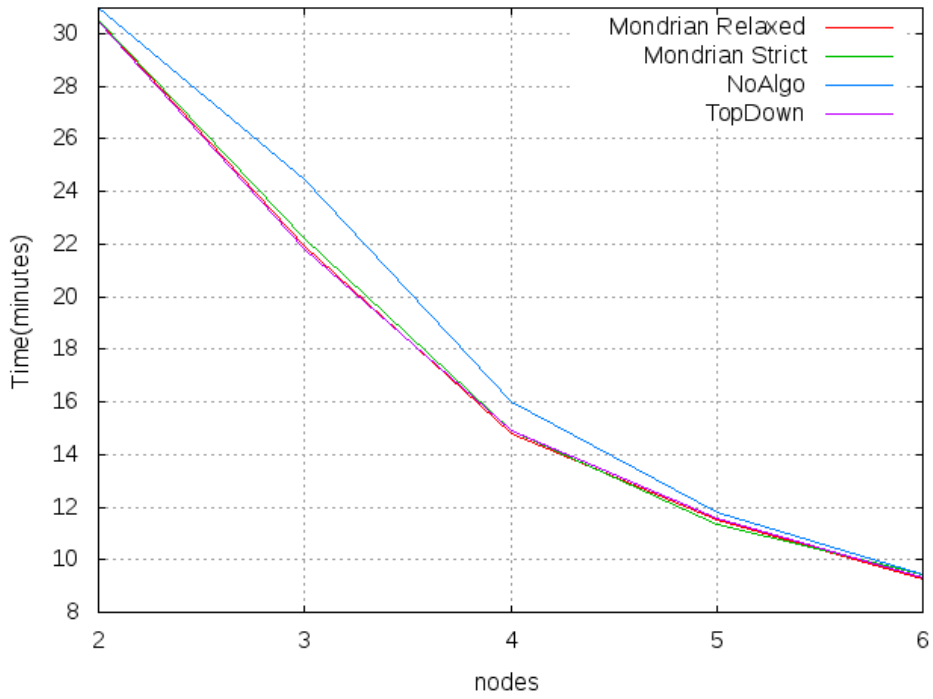
Κατά την εκτέλεση των μεθόδων ανωνυμοποίησης, μεταβάλλαμε το μέγεθος του cluster από 2 κόμβους ως 6 κόμβους προσθέτοντας ένα κόμβο κάθε φορά. Τα αποτελέσματα των δεικτών GCP που λάβαμε για κάθε εκτέλεση ήταν ακριβώς ίδια (για κάθε μέθοδο ανωνυμοποίησης) ανεξαρτήτως αριθμού κόμβων[5]. Αυτό οφείλεται στο γεγονός ότι ο αριθμός των κόμβων του cluster δεν επηρεάζουν την ποιοτική απόδοση των μεθόδων ανωνυμοποίησης, από τη στιγμή που κάθε κόμβος εκτελεί ένα συγκεκριμένο αριθμό από mappers (8 όπως είδαμε) και ο αριθμός των κόμβων δείχνει πόσοι mappers μπορούν να εκτελεστούν παράλληλα. Από τη στιγμή που τα χαρακτηριστικά που μπορούν να επηρεάσουν την απόδοση της ανωνυμοποίησης μένουν αμετάβλητα (πχ, αριθμός partition, dataset, ρυθμός δειγματοληψίας), το μόνο που μεταβάλλεται με τον αριθμό των κόμβων του cluster είναι ο αριθμός των tasks που μπορούν να εκτελεστούν ταυτόχρονα στο cluster. Συνεπώς, η μεταβολή του μεγέθους του cluster επηρεάζει μόνο το χρόνο εκτέλεσης των εργασιών.

[5]Τα αποτελέσματα που λάβαμε για κάθε μέθοδο ήταν ίδια με τα αποτελέσματα που απεικονίζονται στον Πίνακα 4.12 για k=10.

Στον Πίνακα 4.14 παρουσιάζουμε τους χρόνους εκτέλεσης όλων των μεθόδων ανωνυμοποίησης για διάφορα μεγέθη cluster. Με βάση τον πίνακα, παρουσιάζουμε τη γραφική παράσταση των χρόνων εκτέλεσης των μεθόδων ανωνυμοποίησης σαν συνάρτηση του μεγέθους του cluster στο Σχήμα 4.12. Στον οριζόντιο άξονα αναπαριστούμε τον αριθμό των κόμβων ενώ στον κατακόρυφο άξονα αναπαρίσταται ο χρόνος (σε λεπτά).

number of nodes	**Mondrian (relaxed)**	**Mondrian (strict)**	**NoAlgorithm**	**TopDown**
2	00:30:31	00:30:29	00:30:58	00:30:28
3	00:21:56	00:22:14	00:24:28	00:21:50
4	00:14:46	00:14:55	00:16:01	00:14:54
5	00:11:32	00:11:19	00:11:48	00:11:35
6	00:09:17	00:09:27	00:09:27	00:09:20

Πίνακας 4.14: Χρόνοι εκτέλεσης για διάφορα μεγέθη cluster



Σχήμα 4.12: Γραφική παράσταση χρόνων εκτέλεσης για διάφορα μεγέθη cluster

Όπως παρατηρούμε από τη γραφική παράσταση, η αύξηση του αριθμού των κόμβων του cluster μειώνει γραμμικά το χρόνο εκτέλεσης των μεθόδων ανωνυμοποίησης. Η εικόνα της γραφικής παράστασης είναι αναμενόμενη, αφού όπως εξηγήσαμε και προηγουμένως ο αριθμός των κόμβων του cluster καθορίζει τον αριθμό των tasks που μπορούν να εκτελούνται παράλληλα. Από τη στιγμή που προσθέτουμε νέους κόμβους στο υπάρχον cluster, επιτρέπουμε την παράλληλη εκτέλεση περισσότερων tasks και το MapReduce job συνολικά ολοκληρώνεται γρηγορότερα.

Παρ'όλα αυτά, αν συνεχίσουμε να αυξάνουμε τους κόμβους του cluster πάνω από έναν αριθμό, τότε η μείωση του χρόνου εκτέλεσης γίνεται ολοένα μικρότερη και σε κάποιο σημείο μηδενίζεται. Βλέπουμε και στο σχήμα ότι η κλίση της γραφικής παράσταση μειώνεται όσο αυξάνεται ο αριθμός των κόμβων. Σε περίπτωση που προσθέσουμε τόσους κόμβους έτσι ώστε το γινόμενο $nodes \times tasksslots$ να είναι ίσο με τον αριθμό των partitions τότε οποιαδήποτε προσθήκη κόμβων δεν θα μεταβάλλει το χρόνο εκτέλεσης των μεθόδων ανωνυμοποίησης.

Μια σημαντική παρατήρηση που πρέπει να γίνει αφορά το ρόλο των task slots κάθε κόμβου στο χρόνο εκτέλεσης των εργασιών. Είπαμε πριν ότι σε ένα cluster για να βρούμε τον αριθμό των tasks που μπορούν να εκτελεστούν παράλληλα αρκεί να πολλαπλασιάσουμε τον αριθμό των κόμβων του cluster με τον αριθμό των task slots κάθε κόμβου[6]. Είδαμε επίσης ότι η μεταβολή του αριθμού των κόμβων του cluster επηρεάζει με γραμμικό τρόπο το χρόνο ολοκλήρωσης των MapReduce jobs. Η μεταβολή των task slots των κόμβων, όμως, δεν επηρεάζει γραμμικά τον χρόνο ολοκλήρωσης των εργασιών. Περισσότερα task slots σημαίνει ότι πιο πολλά tasks θα εκτελεστούν παράλληλα, αλλά τα tasks αυτά θα εκτελεστούν στο ίδιο cluster, πράγμα που σημαίνει ότι πιο πολλά tasks θα διεκδικούν τους ίδιους πόρους και τελικά θα καθυστερήσουν περισσότερο στην ολοκλήρωσή τους, αφήνοντας τον συνολικό χρόνο εκτέλεσης αμετάβλητο. Σε περιπτώσεις μάλιστα που ο αριθμός των task slots αυξάνεται πολύ, οι κόμβοι του cluster δαπανούν πιο πολύ ώρα στην χρονοδρομολόγηση των tasks και στο συμβιβασμό των πολλών ενεργών διεργασιών παρά στην ωφέλιμη εκτέλεση των ενεργειών του task, με αποτέλεσμα ο τελικός χρόνος εκτέλεσης του MapReduce job να είναι χειρότερος από πριν. Γενικά, η παραπάνω γραφική παράσταση ισχύει μόνο σε περίπτωση που αυξάνονται οι κόμβοι του cluster (προστίθενται δηλαδή και άλλοι πόροι στο cluster) και όχι τα task slots.

4.4.5 Μεταβολή μεγέθους και κατανομής δεδομένων

Τέλος, θα εξετάσουμε την επίδραση του μεγέθους των δεδομένων και της κατανομής που ακολουθούν στην ποιότητα της ανωνυμοποίησης και στο χρόνο εκτέλεσης της. Μέχρι στιγμής, σε όλες τις εκτελέσεις που παρουσιάσαμε είχαμε χρησιμοποιήσει το dataset των 5 εκατομμυρίων tuples, όταν τα δεδομένα ακολουθούσαν τη Uniform κατανομή (υπενθυμίζουμε ότι με την κατανομή αυτή, όλες οι τιμές ενός attribute έχουν την ίδια πιθανότητα εμφάνισης). Στο σημείο αυτό, θα εκτελέσουμε την ανωνυμοποίηση για όλα τα dataset που παρουσιάσαμε στην Ενότητα 4.1.2.

Μια πολύ σημαντική παράμετρος της κατανεμημένης ανωνυμοποίησης είναι, όπως εί-

[6]Ο αριθμός των task slots αν και στη γενική περίπτωση δεν είναι κατ' ανάγκη ίδιος για όλους τους κόμβους, είναι γενικά προτιμότερο οι κόμβοι του cluster να είναι των ίδιων δυνατοτήτων και να παρέχουν τους ίδιους πόρους στο σύστημα. Με βάση την επεξεργαστική ισχύ που μπορεί να παρέχει κάθε υπολογιστής στο cluster (cores και μνήμη) καθορίζεται συνήθως και ο αριθμός των task slots του κόμβου. Στην παρούσα εργασία το cluster αποτελείται από παρόμοιους υπολογιστές κάθε ένας από τους οποίους συμμετέχει εξίσου στην εκτέλεση των tasks (ένα τέτοιο cluster καλείται και συμμετρικό).

δαμε, το μέγεθος του partition. Το βέλτιστο μέγεθος partition είναι αυτό στο οποίο επιτυγχάνεται με τον καλύτερο δυνατό τρόπο ο συμβιβασμός ανάμεσα στην μείωση της ποιότητας της ανωνυμοποίησης και της μείωσης του χρόνου εκτέλεσής της. Επειδή η εύρεση του ιδανικού partition size έχει άμεση σχέση με τα δεδομένα και τα χαρακτηριστικά τους, όπως τα ranges των attributes τους, η κατανομή που ακολουθούν οι τιμές τους, κοκ δεν εξετάζουμε για κάθε dataset ξεχωριστά ποιό είναι το βέλτιστο partition size, αλλά χρησιμοποιούμε το ίδιο για όλα τα dataset. Το κριτήριο μας για την επιλογή του είχε σχέση με την ελαχιστοποίηση του χρόνου εκτέλεσης των αλγορίθμων: θέλαμε να επιλέξουμε το μεγαλύτερο δυνατό partition size (έτσι ώστε οι αλγόριθμοι να εκτελούν ποιοτικά υψηλή ανωνυμοποίηση) το οποίο όμως να μας εξασφαλίζει ικανοποιητικούς χρόνους εκτέλεσης. Είδαμε ότι στην εκτέλεση της Ενότητας 4.4.1, το partition size που συνδυάζει με το βέλτιστο τρόπο αυτά τα δυο στοιχεία είναι 10000 tuples/partition και για το λόγο αυτό θα το χρησιμοποιήσουμε.

Ένα δεύτερο πολύ σημαντικό στοιχείο της κατανεμημένης ανωνυμοποίησης το οποίο δεν αναφέρθηκε ιδιαίτερα προηγουμένως είναι ο ρυθμός δειγματοληψίας των δεδομένων (που εκτελείται από τον Sampler). Στις προηγούμενες εκτελέσεις χρησιμοποιούσαμε ένα ποσοστό της τάξης του 20%. Όσο μεγαλύτερο είναι το ποσοστό δειγματοληψίας, τα tuples-τομές που βρίσκει ο sampler μεταξύ των partitions είναι πιο ακριβή και βρίσκονται πλησιέστερα στις πραγματικές τομές που θα υπήρχαν αν εξετάζαμε όλα τα tuples. Παράλληλα όμως, όσο μεγαλύτερο είναι το ποσοστό δειγματοληψίας, τόσα περισσότερα tuples θα εξεταστούν από τον reducer του Sampler, πράγμα αυξάνει το χρόνο εκτέλεσης του Sampler, επιβαρύνει τον κόμβο που εκτελείται ο reducer και αυξάνει κατά πολύ το κόστος σφάλματος σε περίπτωση που για κάποιο λόγο η διεργασία που εκτελεί τον reducer δεν τερματίσει και ξεκινήσει από την αρχή. Αντίθετα, μικρό ποσοστό δειγματοληψίας οδηγεί σε λιγότερη ακρίβεια στην εύρεση των τομών με αποτέλεσμα να αυξάνεται η πιθανότητα να βρεθούν tuples-τομές που βρίσκονται αρκετά μακριά μεταξύ τους, με αποτέλεσμα να δημιουργηθούν partitions τα οποία είναι πολύ μεγάλα, επιβαρύνοντας έτσι περισσότερο τους κόμβους στους οποίους εκτελούνται και αυξάνοντας κατά πολύ το χρόνο εκτέλεσης. Τελικά, για την επιλογή του ρυθμού δειγματοληψίας χρειάζεται να χρησιμοποιήσουμε το μικρότερο δυνατό ποσοστό για το οποίο τα partitions που δημιουργούνται είναι αρκετά παρόμοια. Το ποσοστό αυτό για τις εκτελέσεις που θα ακολουθήσουν είναι ίσο με 15%.

Στις υποενότητες που ακολουθούν θα παρουσιάσουμε τους δείκτες GCP και τους χρόνους εκτέλεσης για τα datasets των τριών κατανομών. Στην τελευταία υποενότητα θα παρουσιάσουμε μια σύγκριση στην απόδοση των αλγορίθμων για κάθε κατανομή.

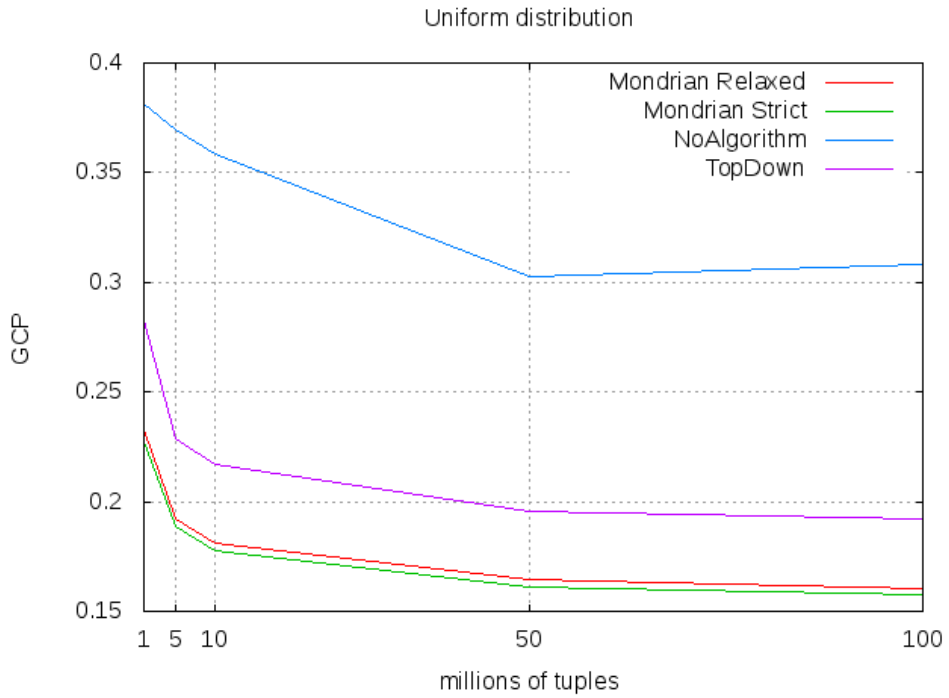
Uniform κατανομή

Αρχικά, θα εξετάσουμε datasets των οποίων τα tuples ακολουθούν Uniform κατανομή μέσα στα διαστήματα τιμών που μπορούν να λάβουν σε κάθε attribute[7]. Με την κατανομή αυτή τα δεδομένα τοποθετούνται ομοιόμορφα στο χώρο, χωρίς να υπάρχουν σημεία στα οποία παρατηρείται έντονη συσσώρευση δεδομένων σε σχέση με άλλες περιοχές του χώρου. Για το λόγο αυτό λέμε ότι τα δεδομένα δεν είναι πολωμένα. Παρακάτω, στον Πίνακα 4.15 παραθέτουμε τους δείκτες GCP των τεσσάρων μεθόδων ανωνυμοποίησης για κάθε dataset.

dataset size (millions of tuples)	Mondrian (relaxed)	Mondrian (strict)	NoAlgorithm	TopDown
1	0.232675	0.227643	0.381071	0.282812
5	0.192458	0.188623	0.369370	0.228808
10	0.181009	0.177613	0.358540	0.217191
50	0.164735	0.160968	0.302497	0.195312
100	0.160147	0.157666	0.307911	0.191862

Πίνακας 4.15: Δείκτες GCP για datasets που ακολουθούν Uniform κατανομή

Με βάση τον Πίνακα 4.15 κατασκευάζουμε τη γραφική παράσταση του Σχήματος 4.13 στην οποία απεικονίζεται η μεταβολή του GCP για κάθε μέθοδο σαν συνάρτηση του μεγέθους του dataset.



Σχήμα 4.13: Γραφική παράσταση GCP για datasets που ακολουθούν Uniform κατανομή

[7]Υπενθυμίζουμε ότι με τη Uniform κατανομή κάθε τιμή που βρίσκεται μέσα στο διάστημα τιμών ενός attribute (range) έχει την ίδια πιθανότητα εμφάνισης.

Όπως παρατηρούμε από το παραπάνω Σχήμα, η αύξηση του μεγέθους του dataset οδηγεί στη μείωση του δείκτη GCP των μεθόδων ανωνυμοποίησης. Το φαινόμενο αυτό οφείλεται στην αύξηση της ποσότητας των tuples σε ένα χώρο ο οποίος παραμένει σταθερός (από τη στιγμή που ο αριθμός των attributes και τα ranges τους παραμένουν σταθερά, τα μεγαλύτερα datasets περιέχουν περισσότερα tuples στον ίδιο ακριβώς χώρο). Η ύπαρξη περισσότερων tuples στον ίδιο χώρο, οδηγεί τις μεθόδους ανωνυμοποίησης να έχουν περισσότερες επιλογές στη δημιουργία των τελικών κλάσεων ισοδυναμίας, με αποτέλεσμα τα tuples που περιέχονται στην ίδια κλάση να βρίσκονται πιο κοντά σε σχέση με την περίπτωση που υπήρχαν λιγότερα διαθέσιμα δεδομένα. Με άλλα λόγια, η ύπαρξη περισσότερων δεδομένων στον ίδιο χώρο οδηγεί σε περισσότερες κλάσεις ισοδυναμίας που έχουν πιο μικρά εύρη τιμών με περίπου τον ίδιο αριθμό στοιχείων (περίπου k). Άρα, πιο μεγάλα datasets οδηγούν σε μείωση του NCP των κλάσεων ισοδυναμίας με αποτέλεσμα τη μείωση του GCP των μεθόδων ανωνυμοποίησης.

Ένα πολύ ενδιαφέρον στοιχείο που προκύπτει από τη γραφική παράσταση του Σχήματος 4.13 αφορά το ρυθμό μείωσης του δείκτη GCP για μικρά και μεγαλύτερα datasets. Παρατηρούμε ότι όταν αυξάνουμε το μέγεθος του dataset από τα 1 στα 5 εκατομμύρια tuples ο ρυθμός μείωσης του GCP είναι πολύ μεγάλος ενώ σταδιακά όσο αυξάνεται το πλήθος των tuples στο dataset, ο ρυθμός μείωσης γίνεται ολοένα και μικρότερος. Αυτό οφείλεται, σύμφωνα με την προηγούμενη περιγραφή μας, στο ότι τα μεγαλύτερα datasets ενώ προσφέρουν τη δυνατότητα να δημιουργηθούν μικρότερες κλάσεις ισοδυναμίας, τα πολύ μεγάλα datasets δεν μπορούν να μειώνουν επ'αόριστον το NCP των κλάσεων. Δηλαδή η αύξηση του πλήθους των tuples αν και στην αρχή συμβάλλει στη γενική απόδοση της ανωνυμοποίησης με τη δημιουργία περισσότερων (σε αριθμό) και μικρότερων (σε ranges) κλάσεων ισοδυναμίας η συνεχής αύξηση δεν μπορεί να μειώνει συνεχώς τα ranges των κλάσεων ισοδυναμίας. Για αυτό το λόγο ο ρυθμός μείωσης του GCP αρχικά είναι μεγάλος ενώ στη συνέχεια μειώνεται και σε αρκετά μεγάλα datasets παρουσιάζει πολύ μικρές μεταβολές. Ακόμη, σε κάποιες περιπτώσεις όπως βλέπουμε από τη NoAlgorithm, ο δείκτης GCP μετά από κάποιο μέγεθος dataset μπορεί να σταματήσει να μειώνεται και να αρχίσει να αυξάνεται. Το φαινόμενο αυτό οφείλεται σε αυτό που περιγράψαμε προηγουμένως με την ολοένα και μικρότερη συμβολή των περισσότερων tuples στη μείωση του NCP των κλάσεων ισοδυναμίας που, όπως θα δούμε και παρακάτω, σε κάποιες περιπτώσεις σταματά να βοηθά την ανωνυμοποίηση και αρχίζει να μειώνει την απόδοση της.

Στη συνέχεια, μετά την εξέταση της απόδοσης της ανωνυμοποίησης, θα εξετάσουμε το χρόνο εκτέλεσης των Map Reduce εργασιών που χρειάστηκαν για την εκτέλεση των μεθόδων. Στον Πίνακα 4.16 παραθέτουμε τους χρόνους εκτέλεσης των αρχικών εργασιών που απαιτούνται για το αρχικό partitioning των δεδομένων. Στον Πίνακα 4.17 παραθέτουμε τους χρόνους εκτέλεσης των μεθόδων ανωνυμοποίησης καθώς και το συνολικό χρόνο εκτέλεσης που απαιτήθηκε για την εκτέλεση όλων των εργασιών. Τέλος, στο Σχήμα 4.14 παραθέτουμε τις αντίστοιχες γραφικές παραστάσεις (με τη βοήθεια των δυο πινάκων) που αναπαριστούν:

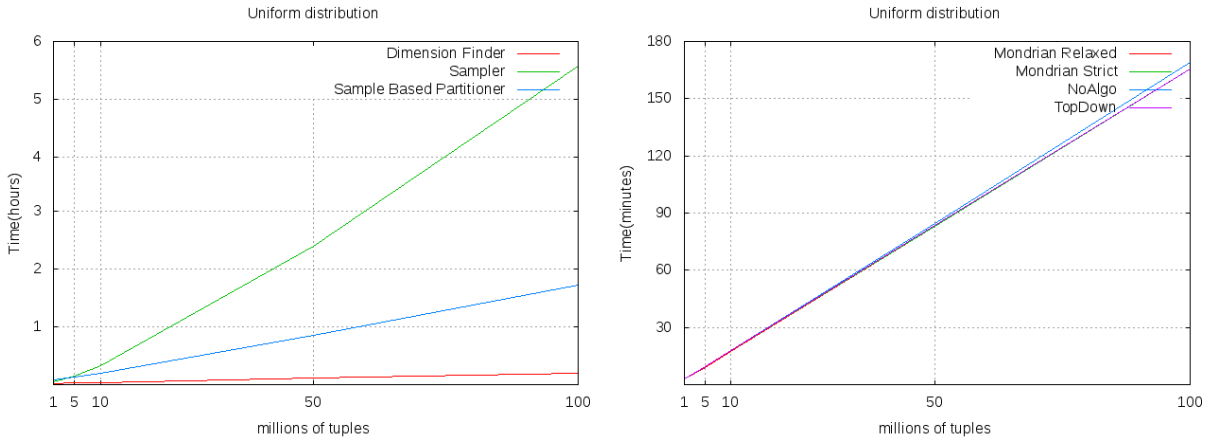
(a) το χρόνο εκτέλεσης των 3 αρχικών εργασιών και (b) το χρόνο εκτέλεσης των 4 μεθόδων ανωνυμοποίησης.

dataset size (millions of tuples)	Dimension finder	Sampler	Sample Based Partitioner
1	00:01:17	00:02:39	00:04:51
5	00:01:33	00:09:01	00:07:49
10	00:01:56	00:20:00	00:12:11
50	00:06:51	02:24:12	00:50:49
100	00:11:57	05:30:53	01:43:05

Πίνακας 4.16: Χρόνοι εκτέλεσης αρχικών εργασιών για datasets που ακολουθούν Uniform κατανομή

dataset size	Mondrian (relaxed)	Mondrian (strict)	NoAlgorithm	TopDown	Overall
1	00:02:57	00:02:54	00:02:51	00:03:03	00:20:32
5	00:09:11	00:09:17	00:09:31	00:09:24	00:55:46
10	00:17:26	00:17:44	00:18:02	00:17:55	01:45:14
50	01:23:16	01:23:08	01:24:39	01:23:34	08:56:29
100	02:45:41	02:45:30	02:49:08	02:45:44	18:31:58

Πίνακας 4.17: Χρόνοι εκτέλεσης εργασιών ανωνυμοποίησης για datasets που ακολουθούν Uniform κατανομή



(a) (b)

Σχήμα 4.14: Γραφικές παραστάσεις χρόνων εκτέλεσης για datasets που ακολουθούν Uniform κατανομή

Όπως βλέπουμε από τις γραφικές παραστάσεις του Σχήματος 4.14, όλα τα MapReduce jobs μεταβάλλουν με γραμμικό τρόπο το χρόνο εκτέλεσής τους ως προς τον αριθμό των tuples της εισόδου, όπως ήταν και αναμενόμενο. Από τις αρχικές Map Reduce εργασίες, βλέπουμε ότι ο Sampler είναι εκείνος που απαιτεί τον περισσότερο χρόνο εκτέλεσης, ο

Sample Based Partitioner είναι ο αμέσως επόμενος ενώ ο Dimension Finder (το Map Reduce job που εξετάζει τη σπουδαιότητα των attributes του QID καθώς και τα ranges των attributes) εκτελείται στο συντομότερο διάστημα. Τέλος, οι μέθοδοι ανωνυμοποίησης βλέπουμε ότι εκτελούνται περίπου στον ίδιο χρόνο και αυξάνουν το χρόνο εκτέλεσής τους με γραμμικό τρόπο ως προς το μέγεθος του dataset.

Βλέπουμε δηλαδή ότι η αύξηση του μεγέθους του dataset αυξάνει με γραμμικό τρόπο το χρόνο εκτέλεσης των κατανεμημένων εργασιών. Παράλληλα, είδαμε στην Ενότητα 4.4.4 ότι η αύξηση του μεγέθους του cluster μειώνει με γραμμικό τρόπο το χρόνο εκτέλεσης των MapReduce εργασιών. Αν συνδυάσουμε αυτές τις δυο πληροφορίες, τότε καταλαβαίνουμε ότι αν υπάρχει η δυνατότητα να αυξήσουμε το μέγεθος του cluster, μπορούμε με προσθήκη κατάλληλου αριθμού κόμβων σε κάθε περίπτωση να διατηρήσουμε σταθερό το χρόνο εκτέλεσης των εργασιών, παρά την αύξηση των δεδομένων. Η δυνατότητα αυτή ονομάζεται scaling και είναι μια πολύ σημαντική ιδιότητα των κατανεμημένων συστημάτων.

Gauss κατανομή

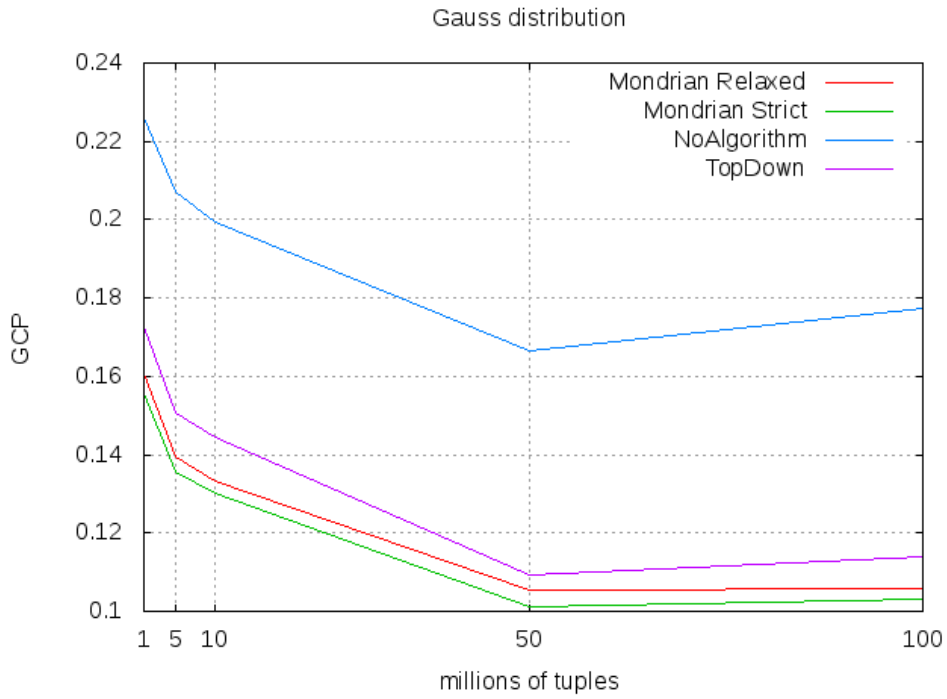
Στη συνέχεια, εξετάζουμε datasets τα οποία έχουν δεδομένα των οποίων οι τιμές ακολουθούν την κατανομή Gauss. Η κατανομή αυτή συγκεντρώνει το μεγαλύτερο μέρος των δεδομένων περίπου στη μέση του διαστήματος τιμών των attributes, με αποτέλεσμα τα περισσότερα δεδομένα να είναι συγκεντρωμένα στη μέση περίπου του q-διάστατου χώρου τον οποίο ορίζουν τα attributes. Επειδή, σε αντίθεση με την προηγούμενη κατανομή, υπάρχουν περιοχές του χώρου στις οποίες συναντώνται περισσότερα δεδομένα σε σχέση με τις υπόλοιπες, για το λόγο αυτό η κατανομή αυτή λέμε ότι δημιουργεί πόλωση στα δεδομένα.

Εκτελούμε τις μεθόδους ανωνυμοποίησης, όπως και πριν και παραθέτουμε τους δείκτες GCP των μεθόδων για κάθε dataset που χρησιμοποιήσαμε. Στον Πίνακα 4.18 παραθέτουμε τους δείκτες GCP και στο Σχήμα 4.15 παραθέτουμε την αντίστοιχη γραφική παράσταση που απεικονίζει την ποιότητα της ανωνυμοποίησης που εκτέλεσαν οι μέθοδοι σαν συνάρτηση του μεγέθους του dataset.

dataset size (millions of tuples)	**Mondrian (relaxed)**	**Mondrian (strict)**	**NoAlgorithm**	**TopDown**
1	0.160877	0.155546	0.226103	0.172844
5	0.139385	0.135619	0.206988	0.150542
10	0.133072	0.130159	0.199450	0.144337
50	0.105370	0.101334	0.166712	0.109140
100	0.105656	0.103286	0.177242	0.113823

Πίνακας 4.18: Δείκτες GCP για datasets που ακολουθούν Gauss κατανομή

Όπως βλέπουμε από την παραπάνω γραφική παράσταση, η αύξηση του μεγέθους του dataset (δεδομένου ότι το εύρος των τιμών των attributes παραμένει σταθερό) αρχικά βελτιώνει την ποιότητα της ανωνυμοποίησης, όπως είχαμε παρατηρήσει και στην περίπτωση



Σχήμα 4.15: Γραφική παράσταση GCP για datasets που ακολουθούν Gauss κατανομή

της Uniform κατανομής. Βλέπουμε ότι όταν το dataset έχει λίγα tuples (πχ, 1 εκατομμύριο tuples) και στη συνέχεια το μέγεθος του dataset αυξηθεί σημειώνεται ραγδαία πτώση στον δείκτη GCP που ισοδυναμεί με άνοδο στην ποιότητα της ανωνυμοποίησης. Όπως είπαμε και προηγουμένως, η αύξηση του αριθμού των tuples σημαίνει ότι οι αλγόριθμοι ανωνυμοποίησης έχουν την ευκαιρία να ομαδοποιήσουν tuples τα οποία βρίσκονται πιο κοντά σε σχέση με πριν (διότι αυξάνεται η πυκνότητα των tuples στο χώρο) με αποτέλεσμα οι παραγόμενες κλάσεις ισοδυναμίας να έχουν μικρότερο δείκτη NCP και τελικά, η μέθοδος στο σύνολό της λαμβάνει μικρότερες τιμές GCP.

Παρ'όλα αυτά, αν συνεχίσουμε να αυξάνουμε τον αριθμό των tuples πάνω από 50 εκατομμύρια, βλέπουμε κάτι το οποίο δεν εντοπίσαμε στην προηγούμενη περίπτωση. Για αύξηση του μεγέθους του dataset από 50 σε 100 εκατομμύρια tuples, βλέπουμε ότι όλοι οι μέθοδοι ανωνυμοποίησης αυξάνουν το δείκτη GCP τους, κάτι που υποδηλώνει ότι χειροτερεύει η ποιότητα της ανωνυμοποίησης. Το φαινόμενο αυτό οφείλεται στην αύξηση του αριθμού των tuples σε συνδυασμό με τη λειτουργία του partitioner. Είδαμε προηγουμένως ότι όταν το partition size γίνει πολύ μικρό (ή αλλιώς, όταν ο αριθμός των partition μεγαλώνει πολύ) η λειτουργία του partitioner αρχίζει και επηρεάζει εντονότερα τη λειτουργία των αλγορίθμων με αποτέλεσμα να μειώνει την απόδοσή τους. Στην περίπτωση των 100 εκατομμυρίων tuples, ο αριθμός των partitions είναι 10000 (αφού έχουμε περίπου 10000 tuples/partition), που είναι ένας αρκετά μεγάλος αριθμός. Αυτό έχει ως αποτέλεσμα ο partitioner να δημιουργεί πολλές ομάδες και να διαχωρίζει tuples τα οποία θα έπρεπε (αν δεν εφαρμοζόταν partitioning) να καταλήξουν στην ίδια ομάδα.

Ένα σημαντικό ζήτημα σχετικά με το φαινόμενο αυτό είναι το γιατί η επίδραση του partitioner φάνηκε εντονότερα στην περίπτωση της κατανομής Gauss και όχι στην προηγούμενη περίπτωση της Uniform κατανομής (θυμίζουμε ότι στη Uniform περίπτωση η μόνη μέθοδος της οποίας αυξήθηκε το GCP ήταν η μέθοδος NoAlgorithm). Ο λόγος που συνέβη αυτό έχει σχέση με την πόλωση των δεδομένων. Όταν τα δεδομένα είναι πολωμένα, όπως εδώ, τα partitions που θα δημιουργηθούν αναμένουμε να περιέχουν (τα περισσότερα από αυτά) tuples που ανήκουν στην περιοχή της έντονης πόλωσης. Η αύξηση των tuples σημαίνει ότι θα υπάρξουν ακόμη περισσότερα partitions που περιέχουν tuples της περιοχής της έντονης πόλωσης. Έτσι λοιπόν, από ένα σημείο και μετά τα partitions αρχίζουν να παρουσιάζουν επικαλύψεις μεταξύ τους και η διαμέριση των tuples γίνεται λιγότερο αποδοτική στο χώρο έντονης πόλωσης γιατί κοντινά tuples τοποθετούνται σε διαφορετικές ομάδες. Στην περίπτωση που δεν έχουμε πολωμένη κατανομή (όπως η Uniform) η αύξηση των tuples γίνεται ομοιόμορφα και για να αντιμετωπίσουμε το παραπάνω πρόβλημα πρέπει να προστεθεί ένας μεγάλος αριθμός από tuples σε όλο το χώρο. Στην ουσία δηλαδή, το πρόβλημα επιδεινώνεται πολύ από την πόλωση των δεδομένων και για αυτό το λόγο στην περίπτωση της κατανομής Gauss εμφανίζεται στα 100 εκατομμύρια tuples ενώ στην περίπτωση της κατανομής Uniform δεν εμφανίζεται παρά σε μια μόνο μέθοδο[8].

Συνεχίζοντας, παραθέτουμε τους χρόνους εκτέλεσης όλων των κατανεμημένων εργασιών που εκτελέστηκαν για την ανωνυμοποίηση των δεδομένων με τις τέσσερις μεθόδους. Στον Πίνακα 4.19 παραθέτουμε τους χρόνους εκτέλεσης των προπαρασκευαστικών εργασιών όπως είναι η εύρεση της σπουδαιότητας των attributes (Dimension Finder), η δειγματοληψία (Sampler) και η διαμέριση των tuples (Sample Based Partitioner) και στον Πίνακα 4.20 παραθέτουμε τους χρόνους εκτέλεσης των μεθόδων ανωνυμοποίησης για κάθε dataset. Με βάση τους δυο πίνακες κατασκευάζουμε και τις γραφικές παραστάσεις που απεικονίζονται στο Σχήμα 4.16.

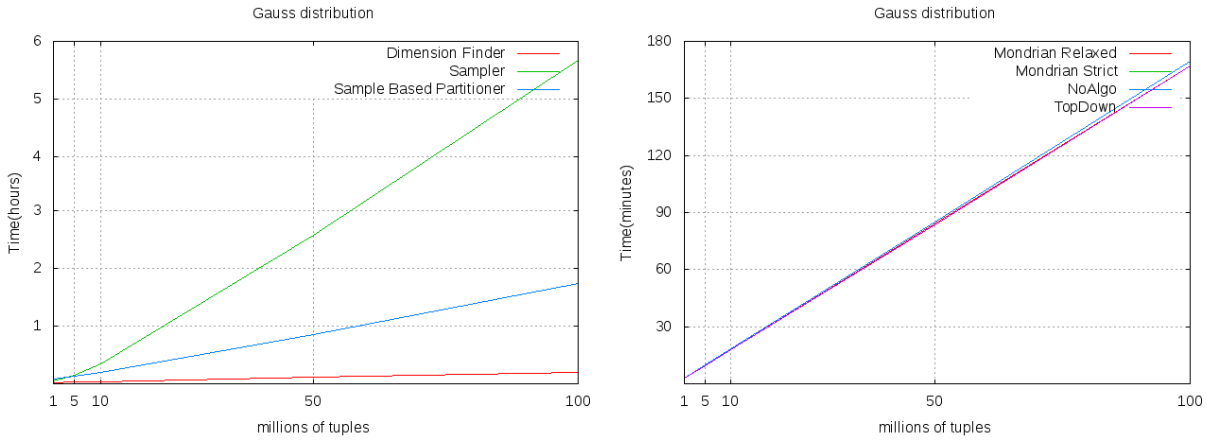
dataset size (millions of tuples)	**Dimension finder**	**Sampler**	**Sample Based Partitioner**
1	00:01:19	00:02:46	00:05:24
5	00:01:37	00:09:21	00:07:49
10	00:02:01	00:21:06	00:11:52
50	00:06:40	02:34:22	00:51:28
100	00:11:41	05:36:57	01:44:10

Πίνακας 4.19: Χρόνοι εκτέλεσης αρχικών εργασιών για datasets που ακολουθούν Gauss κατανομή

[8]Η πόλωση των δεδομένων, θα μπορούσαμε να πούμε (αρκετά ελεύθερα) ότι "μειώνει" το μέγεθος του χώρου, αφού συγκεντρώνει την πλειοψηφία των tuples σε ένα χώρο αρκετά μικρότερων (όχι λιγότερων) διαστάσεων. Για παράδειγμα, στην κατανομή Gauss το 75% των tuples αναμένεται να είναι συγκεντρωμένα στο μεσαίο 1/3 του διαστήματος τιμών για κάθε attribute. Αυτό σημαίνει ότι η αύξηση του αριθμού των tuples "συσσωρεύει" μεγάλο αριθμό tuples πολύ γρηγορότερα σε περιοχές πόλωσης σε σχέση με μη πολωμένες κατανομές.

dataset size	Mondrian (relaxed)	Mondrian (strict)	NoAlgorithm	TopDown	Overall
1	00:02:57	00:02:57	00:02:57	00:03:00	00:21:20
5	00:09:27	00:09:23	00:09:44	00:09:26	00:56:47
10	00:17:52	00:17:55	00:18:12	00:17:53	01:46:51
50	01:23:31	01:23:30	01:25:01	01:23:45	09:08:17
100	02:47:08	02:47:01	02:49:31	02:46:57	18:43:25

Πίνακας 4.20: Χρόνοι εκτέλεσης εργασιών ανωνυμοποίησης για datasets που ακολουθούν Gauss κατανομή



(a) (b)

Σχήμα 4.16: Γραφικές παραστάσεις χρόνων εκτέλεσης για datasets που ακολουθούν Gauss κατανομή

Όπως βλέπουμε και στην περίπτωση της Gauss κατανομής ο χρόνος εκτέλεσης των κατανεμημένων εργασιών παραμένει σταθερός και αυξάνεται γραμμικά με την αύξηση του μεγέθους της εισόδου. Στο Σχήμα 4.16 (a) βλέπουμε τους χρόνους εκτέλεσης των αρχικών εργασιών και στο Σχήμα 4.16 (b) βλέπουμε τους χρόνους εκτέλεσης των μεθόδων ανωνυμοποίησης. Παρατηρούμε ότι τα δυο σχήματα εμφανίζουν φανερή ομοιότητα με τις αντίστοιχες γραφικές παραστάσεις του Σχήματος 4.14. Το συμπέρασμα που προκύπτει από τη σύγκριση των χρόνων εκτέλεσης των datasets που ακολουθούν Uniform και Gauss κατανομή είναι ότι ο χρόνος εκτέλεσης των κατανεμημένων εργασιών είναι ανεξάρτητος από την κατανομή που ακολουθούν τα δεδομένα, αλλά εξαρτάται μόνο από το πλήθος [9] των δεδομένων.

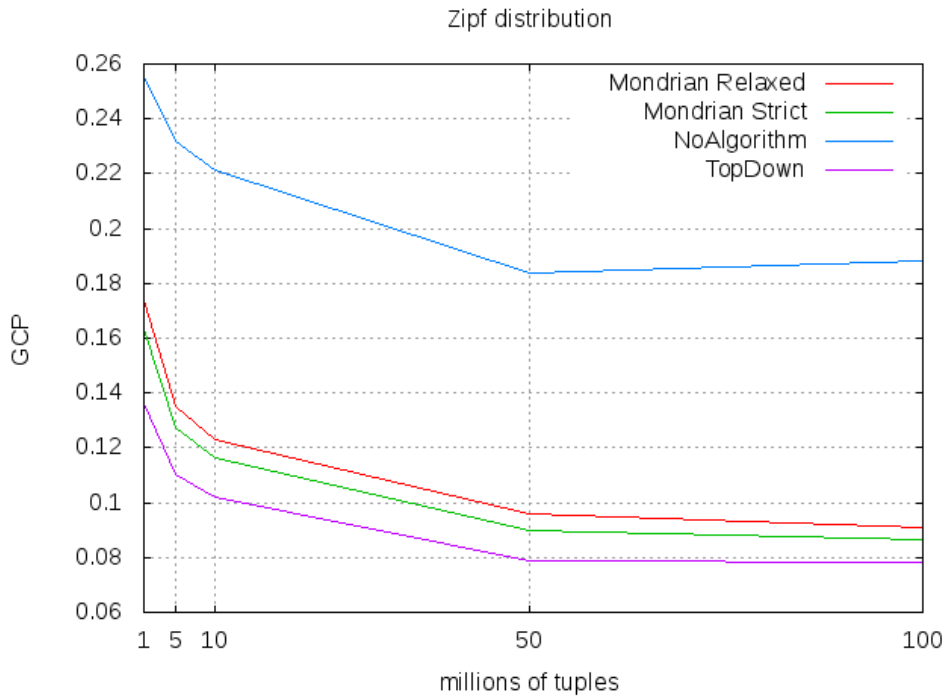
[9]Όπως θα δούμε και παρακάτω, εκτός από το πλήθος των δεδομένων το χρόνο εκτέλεσης επηρεάζει ελαφρώς και το μέγεθος του dataset σε bytes. Όπως παρουσιάσαμε και στην αρχή, τα μεγέθη των datasets των διαφορετικών κατανομών δεν είναι απόλυτα ίσα σε bytes και αυτό επηρεάζει το χρόνο εκτέλεσης κάποιων εργασιών.

Zipf κατανομή

Η τελευταία κατανομή που θα εξετάσουμε είναι η κατανομή Zipf. Και στην περίπτωση της κατανομής Zipf τα δεδομένα είναι πολωμένα, αλλά όχι στο μέσο του διαστήματος των τιμών κάθε attribute, όπως στην περίπτωση της Gauss κατανομής, αλλά στο άκρο των διαστημάτων. Συγκεκριμένα, τα δεδομένα σε αυτήν την περίπτωση δεν έχουν τα ranges των τιμών που εμφανίζονται στον Πίνακα 4.1, αλλά το κάτω άκρο των διαστημάτων ταυτίζεται με το 0, ενώ το εύρος των διαστημάτων παραμένει σταθερό. Με βάση αυτή τη μετατροπή, αναμένουμε τα περισσότερα tuples να βρίσκονται κοντά στην τιμή 0[10]. Στον Πίνακα που ακολουθεί, παραθέτουμε τους δείκτες GCP των τεσσάρων μεθόδων ανωνυμοποίησης για μέγεθος dataset.

dataset size (millions of tuples)	Mondrian (relaxed)	Mondrian (strict)	NoAlgorithm	TopDown
1	0.173654	0.163385	0.255645	0.136291
5	0.135127	0.127632	0.231809	0.110536
10	0.122813	0.116532	0.221507	0.101723
50	0.095866	0.089933	0.183493	0.079006
100	0.090835	0.086619	0.188403	0.078356

Πίνακας 4.21: Δείκτες GCP για datasets που ακολουθούν Zipf κατανομή



Σχήμα 4.17: Γραφική παράσταση GCP για datasets που ακολουθούν Zipf κατανομή

[10]Υπενθυμίζουμε ότι ο συντελεστής s της κατανομής Zipf τέθηκε στην τιμή 2, συνεπώς η κατανομή είναι αρκετά πολωμένη.

Με βάση τον Πίνακα 4.21 κατασκευάζουμε τις γραφικές παραστάσεις που του Σχήματος 4.17, στις οποίες απεικονίζονται οι δείκτες GCP των μεθόδων ανωνυμοποίησης σαν συνάρτηση του μεγέθους του dataset. Όπως βλέπουμε από τις γραφικές παραστάσεις, οι μέθοδοι ανωνυμοποίησης και σε αυτήν την περίπτωση βελτιώνουν τη λειτουργία τους με την αύξηση του πλήθους των στοιχείων του dataset. Παρατηρούμε ότι όλες οι μέθοδοι εμφανίζουν μειωμένο GCP όσο αυξάνεται το πλήθος των partition, με μόνη εξαίρεση την μέθοδο NoAlgorithm που όπως έχουμε δει και από τη Uniform κατανομή, η αύξηση του πλήθους των tuples σε συνδυασμό με τη δημιουργία ενός μεγάλου αριθμού partitions (έτσι ώστε το partition size να παραμείνει σταθερό) αυξάνει το δείκτη GCP.

Ένα πολύ ενδιαφέρον στοιχείο που προκύπτει από την παραπάνω γραφική παράσταση είναι η απόδοση του αλγορίθμου TopDown σε σύγκριση με τον αλγόριθμο Mondrian. Σε όλες τις εκτελέσεις μέχρι τώρα, είχαμε παρατηρήσει ότι ο αλγόριθμος Mondrian ανεξαρτήτως της πολιτικής διαμέρισης που εκτελούσε είχε καλύτερη απόδοση από τον αλγόριθμο TopDown με ελάχιστες εξαιρέσεις[11]. Παρ'όλα αυτά, στην περίπτωση που τα δεδομένα ακολουθούν Zipf κατανομή βλέπουμε ότι η απόδοση του αλγορίθμου TopDown είναι αρκετά καλύτερη από την απόδοση του αλγορίθμου Mondrian. Ενδεικτικά αναφέρουμε ότι στην ο αλγόριθμος TopDown παρουσιάζει μικρότερο GCP κατά 27.4% για 1 εκατομμύριο tuples από τον Mondrian relaxed partitioning, με το ποσοστό αυτό να γίνεται 15.9% για 100 εκατομμύρια tuples ενώ για τον Mondrian strict ο TopDown παρουσιάζει μικρότερο GCP κατά 19.9% και 10.5% για 1 και 100 εκατομμύρια tuples αντίστοιχα.

Το παραπάνω φαινόμενο οφείλεται στον τρόπο με τον οποίο έχουν κατανεμηθεί τα δεδομένα στο χώρο. Στις προηγούμενες περιπτώσεις τα δεδομένα ήταν τοποθετημένα στο χώρο με συμμετρικό τρόπο. Με την κατανομή Zipf όμως, η περιοχή έντονης πόλωσης των δεδομένων είναι η περιοχή που βρίσκεται κοντά στην αρχή των αξόνων του q-διάστατου χώρου. Με τον τρόπο αυτό, το μεγαλύτερο μέρος των δεδομένων συσσωρεύεται σε ένα μόνο πολύ μικρό σημείο του χώρου και στον υπόλοιπο χώρο συναντάται ένας πολύ μικρός αριθμός στοιχείων. Αν συνδυάσουμε αυτό το γεγονός με τον τρόπο λειτουργίας του αλγορίθμου Mondrian, τότε μπορούμε να εξηγήσουμε το λόγο για τον οποίο ο αλγόριθμος αυτός δεν χειρίζεται της μη-συμμετρικές κατανομές με αποδοτικό τρόπο. Όπως είχαμε δει και στην παρουσίαση του αλγορίθμου στην Ενότητα 3.1.1, ο αλγόριθμος αφού επιλέξει μια διάσταση δημιουργεί δυο σύνολα με βάση ένα ενδιάμεσο (ως προς τη διάσταση) στοιχείο. Αυτό σημαίνει ότι στη μέση περίπτωση θα δημιουργήσει δυο σύνολα που θα έχουν έναν παρόμοιο αριθμό στοιχείων (αν μάλιστα εκτελείται relaxed partitioning θα έχουν ακριβώς τον ίδιο αριθμό στοιχείων). Όταν η κατανομή είναι συμμετρική και τα στοιχεία του dataset είναι πολλά, τότε αφού εκτελεστούν αρκετά βήματα ο αλγόριθμος θα καταλήξει να διαμερίζει ένα σύνολο που πλέον δεν είναι συμμετρικό[12], που τότε ο χώρος του προβλήματος

[11] Όπως είδαμε σε προηγούμενη ενότητα, για πολύ μικρά k ο αλγόριθμος TopDown λειτουργούσε αποδοτικότερα από τον Mondrian.

[12] Ο ισχυρισμός αυτός στηρίζεται στην ακόλουθη συλλογιστική: αν τα σημεία του χώρου είναι πολλά σε αριθμό (έτσι ώστε κάθε τομή ως προς μια διάσταση να αφήνει τις υπόλοιπες διαστάσεις με στοιχεία που

θα έχει μειωθεί κατά πολύ. Αν όμως η κατανομή δεν είναι συμμετρική, τότε ο αλγόριθμος Mondrian δημιουργεί κατά τα γνωστά δυο σύνολα περίπου ίσων στοιχείων, όπου όμως το ένα από τα δυο σύνολα έχει πολύ υψηλό δείκτη NCP και το άλλο έχει πολύ χαμηλό δείκτη. Επειδή δηλαδή, για την διαμέριση είναι απαραίτητη εύρεση της διαμέσου και η διάμεσος σε αρκετά πολωμένες κατανομές είναι πολύ κοντά στην περιοχή πόλωση, δημιουργούνται σύνολα που έχουν πολύ αυξημένο δείκτη NCP.

Αντίθετα, ο αλγόριθμος TopDown που λειτουργεί με μοναδικό κριτήριο την "απόσταση" NCP κάθε σημείου με τα δυο σημεία αναφοράς κάθε συνόλου, διαχειρίζεται πολύ καλύτερα τις μη συμμετρικές κατανομές, αφού θα δημιουργεί ένα σύνολο που περιέχει το μεγαλύτερο μέρος των στοιχείων (προφανώς, τα στοιχεία αυτά θα είναι στην περιοχή έντονης πόλωσης) και ένα δεύτερο σύνολο που θα περιέχει πολύ λιγότερα στοιχεία. Με αυτόν τον τρόπο ελαχιστοποιείται το NCP του συνόλου που περιέχει στοιχεία από την "αραιή" περιοχή του χώρου και τελικά, όπως βλέπουμε πειραματικά ενισχύεται η απόδοση του αλγορίθμου σε σχέση με τον αλγόριθμο Mondrian.

Τέλος, παραθέτουμε τους χρόνους εκτέλεσης των αρχικών εργασιών για την εκτέλεση της ανωνυμοποίησης στον Πίνακα 4.22 και των κατανεμημένων μεθόδων στον Πίνακα 4.23.

dataset size (millions of tuples)	**Dimension finder**	**Sampler**	**Sample Based Partitioner**
1	00:01:17	00:02:41	00:05:15
5	00:01:34	00:09:32	00:07:54
10	00:02:04	00:20:47	00:12:04
50	00:06:29	02:37:03	00:51:45
100	00:11:39	05:34:41	01:44:22

Πίνακας 4.22: Χρόνοι εκτέλεσης αρχικών εργασιών για datasets που ακολουθούν Zipf κατανομή

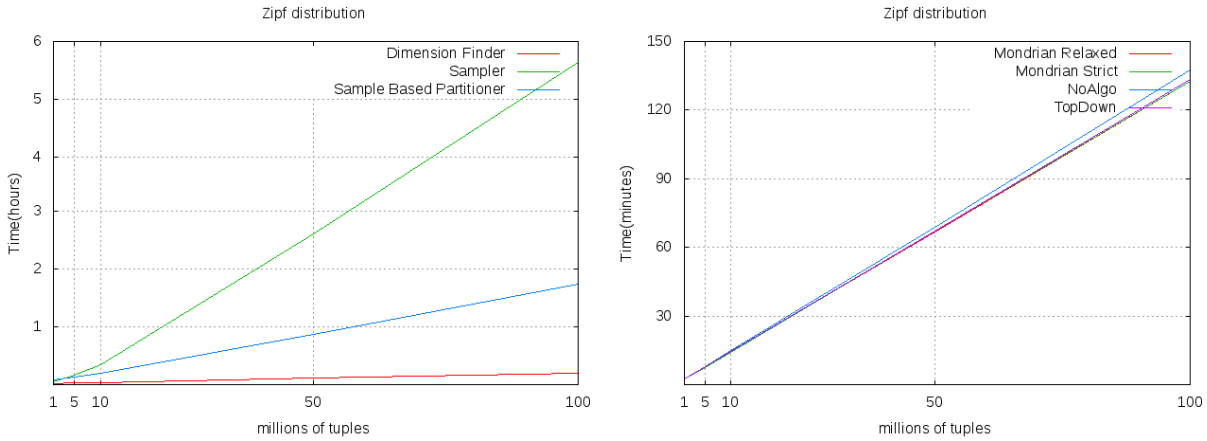
dataset size	**Mondrian (relaxed)**	**Mondrian (strict)**	**NoAlgorithm**	**TopDown**	**Overall**
1	00:02:30	00:02:23	00:02:29	00:02:35	00:19:10
5	00:07:38	00:07:38	00:07:51	00:07:43	00:49:50
10	00:14:34	00:14:16	00:14:47	00:14:27	01:32:59
50	01:06:42	01:07:13	01:08:48	01:07:18	08:05:18
100	02:12:36	02:12:32	02:17:40	02:13:33	16:27:03

Πίνακας 4.23: Χρόνοι εκτέλεσης εργασιών ανωνυμοποίησης για datasets που ακολουθούν Zipf κατανομή

Με τη βοήθεια των δυο Πινάκων κατασκευάζουμε τις γραφικές παραστάσεις που απεικονίζονται στο Σχήμα 4.18. Στο Σχήμα (a) απεικονίζεται ο χρόνος εκτέλεσης των MapReduce

έχουν τιμές σε όλο το εύρος τιμών τους) τότε ο αλγόριθμος πριν αναγκαστεί να διαχωρίσει ως προς μια μη συμμετρική διάσταση θα έχει εκτελέσει q βήματα (τόσα όσες και οι διαστάσεις) διότι στην αρχή, θα επιλέγονται συνεχώς οι διαστάσεις που δεν έχουν επιλεγεί μέχρι τότε. Μετά όμως από q βήματα, ο χώρος έχει διασπαστεί σε 2^q μέρη, με αποτέλεσμα για q=10 ο χώρος να έχει διαμεριστεί σε 1024 τμήματα.

job Dimension Findes, Sampler και SampleBased Partitioner και στο Σχήμα (b) παραθέτουμε το χρόνο εκτέλεσης των μεθόδων ανωνυμοποίησης.



(a) (b)

Σχήμα 4.18: Γραφικές παραστάσεις χρόνων εκτέλεσης για datasets που ακολουθούν Zipf κατανομή

Αν συγκρίνουμε τους χρόνους εκτέλεσης όλων των εργασιών για όλες τις κατανομές θα καταλήξουμε στο συμπέρασμα ότι η κατανομή των δεδομένων στα διαστήματα τιμών δεν επηρεάζει τους χρόνους εκτέλεσης των κατανεμημένων εργασιών, αφού οι αντίστοιχες εργασίες εκτελούνται σε περίπου ίσα χρονικά διαστήματα ανεξαρτήτως της κατανομής των δεδομένων. Αντίθετα, η κατανομή των δεδομένων επηρεάζει σημαντικά της απόδοση των μεθόδων ανωνυμοποίησης, αφού για τις κατανομές που δεν πολώνουν τα δεδομένα (όπως η Uniform), οι μέθοδοι ανωνυμοποίησης παρουσιάζουν τους υψηλότερους δείκτες GCP, ενώ για τις κατανομές που πολώνουν τα δεδομένα, η συμμετρία στην πόλωση (όπως η κατανομή Gauss) επιτρέπει την καλύτερη λειτουργία του αλγορίθμου Mondrian ενώ σε μη συμμετρικές κατανομές (όπως η κατανομή Zipf) ο αλγόριθμος TopDown εκτελεί την καλύτερη ποιοτικά ανωνυμοποίηση.

Κεφάλαιο 5

Επίλογος

Στο κεφάλαιο αυτό θα κάνουμε μια σύντομη ανασκόπηση στο πρόβλημα το οποίο μελετήσαμε, θα περιγράψουμε εν συντομία την μέθοδο που χρησιμοποιήσαμε για την αντιμετώπισή του και τέλος, θα παρουσιάσουμε τα συμπεράσματα που προέκυψαν κατά την επίλυσή του.

5.1 Πρόβλημα κατανεμημένης ανωνυμοποίησης

Το πρόβλημα το οποίο κληθήκαμε να αντιμετωπίσουμε στην παρούσα διπλωματική εργασία ήταν το πρόβλημα των ανωνυμοποίησης σχεσιακών δεδομένων με χρήση κατανεμημένων τεχνικών. Σκοπός μας ήταν να αναπτύξουμε μια κατανεμημένη μέθοδο με την οποία θα μπορούσαμε να γενικεύσουμε ένα μεγάλο όγκο δεδομένων με βέλτιστο τρόπο, έτσι ώστε τα παραχθέντα δεδομένα να ικανοποιούν την ιδιότητα k-anonymity αλλά παράλληλα κατά τη γενίκευση να χάνεται η λιγότερη δυνατή πληροφορία σε σχέση με τα αρχικά δεδομένα.

Κατά την ανάπτυξη της μεθόδου, συναντήσαμε πολλές προκλήσεις που είχαν σχέση με την παραλληλοποίηση των δεδομένων, την εύρεση και την ανάπτυξη centralized αλγορίθμου που να εκτελεί την ανωνυμοποίηση με αποδοτικό τρόπο, τη διαχείριση των δεδομένων με ένα γενικό τρόπο έτσι ώστε η κατανεμημένη μέθοδος να εκτελείται αποδοτικά για διαφορετικά δεδομένα. Στην ενότητα που ακολουθεί περιγράφουμε συνοπτικά τους τρόπους με τους οποίους αντιμετώπισαμε όλες τις παραπάνω προκλήσεις και συνοψίζουμε τη λειτουργία της κατανεμημένης μεθόδου[1].

5.2 Γενική λύση προβλήματος

Η αρχιτεκτονική της μεθόδου που αναπτύξαμε για την επίλυση του προβλήματος της κατανεμημένης ανωνυμοποίησης αποτελείται από δυο μέρη. Το πρώτο μέρος αφορά τον διαμερισμό των δεδομένων έτσι ώστε να γίνει δυνατή η κατανεμημένη εκτέλεση της. Το

[1] Η κατανεμημένη εφαρμογή που αναπτύχθηκε περιγράφηκε λεπτομερώς στο Κεφάλαιο 3.

δεύτερο μέρος αφορά την κατάλληλη γενίκευση των δεδομένων που διαμερίστηκαν έτσι ώστε να πετύχουμε την ικανοποίηση του k-anonymity και, τελικά, την ανωνυμοποίηση.

Για την **διαμέριση** των δεδομένων δημιουργήσαμε μια σχέση διάταξης μεταξύ τους, έτσι ώστε να ταξινομήσουμε και στη συνέχεια να διαιρέσουμε το σύνολο των ταξινομημένων δεδομένων σε ένα σύνολο από partitions. Η σχέση διάταξης στηρίζεται στις τιμές που έχουν τα δεδομένα στα attributes εκείνα που έχουν τα μικρότερα εύρη τιμών. Θεωρούμε ότι τα σημαντικότερα attributes των δεδομένων είναι αυτά που έχουν τα μικρότερα εύρη τιμών. Αν λοιπόν ένα στοιχείο έχει μεγαλύτερη τιμή σε κάποιο σημαντικό attribute σε σχέση με κάποιο άλλο στοιχείο, θεωρούμε ότι το πρώτο στοιχείο είναι μεγαλύτερο του δεύτερου. Σε περίπτωση που οι τιμές του πρώτου σημαντικού attribute είναι ίσες και για τα δύο στοιχεία εξετάζουμε το δεύτερο σημαντικότερο attribute, κοκ. Με αυτόν τον τρόπο μπορούμε να ταξινομήσουμε όλα τα στοιχεία και στη συνέχεια να τα διαιρέσουμε σε μικρότερες υποομάδες που μπορούν να ανωνυμοποιηθούν παράλληλα από το κατανεμημένο σύστημα.

Για την **ανωνυμοποίηση** των partitions που αναφέραμε παραπάνω, χρησιμοποιούμε centralized αλγορίθμους ανωνυμοποίησης. Συγκεκριμένα, χρησιμοποιήσαμε τον αλγόριθμο Mondrian [2] και αναπτύξαμε και τον αλγόριθμο TopDown (με βάση τον αλγόριθμο που περιγράφεται στο [6]), οι οποίοι εκτελούν local recoding και λειτουργούν εκτελώντας συνεχή διαμέριση του συνόλου των δεδομένων που εξετάζουν. Ο αλγόριθμος Mondrian υλοποιήθηκε για δυο διαφορετικές πολιτικές διαμέρισης (relaxed και strict partitioning) ενώ ο αλγόριθμος TopDown εκτελεί αποκλειστικά strict partitioning. Επίσης, αναπτύξαμε και μια greedy μέθοδο ανωνυμοποίησης που δεν στηρίζεται σε κάποιο αλγόριθμο, αλλά ομαδοποιεί τα tuples με διαδοχικό τρόπο σε ομάδες των k στοιχείων με βάση τα ταξινομημένα δεδομένα και μας χρησιμεύει ιδιαίτερα στην κατανόηση της συμβολής του partitioner στη διαδικασία της ανωνυμοποίησης.

Τα δυο μέρη της κατανεμημένης εφαρμογής που περιγράψαμε επηρεάζονται έντονα από διάφορες παραμέτρους της ανωνυμοποίησης: το πλήθος των δεδομένων, την κατανομή τους στο χώρο, την παράμετρο k του k-anonymity, το μέγεθος των partitions και άλλα. Στην ενότητα που ακολουθεί θα συνοψίσουμε τα συμπεράσματα που προέκυψαν από τις εκτελέσεις που περιγράψαμε στο προηγούμενο κεφάλαιο, στις οποίες μεταβάλαμε τις τιμές των παραπάνω παραμέτρων και μελετήσαμε τη συμπεριφορά της κατανεμημένης μεθόδου που αναπτύξαμε.

5.3 Συμπεράσματα

Στο Κεφάλαιο 4 παρουσιάσαμε τα αποτελέσματα των εκτελέσεων της κατανεμημένης εφαρμογής για διάφορες τιμές των παραμέτρων που την επηρεάζουν. Σε αυτή την ενότητα θα συνοψίσουμε τα συμπεράσματα που προέκυψαν από τις εκτελέσεις και την ανάλυση της συμπεριφοράς των κατανεμημένων μεθόδων. Σε κάθε περίπτωση θεωρούμε ότι η παράμε-

τρος που μεταβάλλεται είναι η αναφερόμενη και όλες οι υπόλοιπες παράμετροι παραμένουν σταθερές. Τα συμπεράσματα που προέκυψαν είναι τα εξής:

1. Σύγκριση κεντρικής και κατανεμημένης ανωνυμοποίησης

    - Η απόδοση της κατανεμημένης ανωνυμοποίησης επηρεάζεται σημαντικά από τον Sample Based Partitioner. Η λειτουργία του Partitioner έχει άμεση σχέση με τα δεδομένα, συνεπώς η απόδοση της κατανεμημένης ανωνυμοποίησης επηρεάζεται σημαντικά από τα δεδομένα.
    - Στην περίπτωση του συνθετικού dataset smallData, η απόδοση της κατανεμημένης εκτέλεσης ήταν χειρότερη κατά περίπου 18.4% από την κεντρική εκτέλεση για τον αλγόριθμο Mondrian (strict partitioning), περίπου 12% χειρότερη για τον αλγόριθμο Mondrian (relaxed partitioning) και περίπου 17% χειρότερη για τον αλγόριθμο TopDown, όταν τα δεδομένα διαμερίζονταν σε 20 partitions.
    - Στην περίπτωση του πραγματικού dataset Adults [13], η απόδοση της κατανεμημένης εκτέλεσης ήταν **καλύτερη** σε σχέση με την κεντρική εκτέλεση των αλγορίθμων Mondrian (strict), Mondrian (relaxed) και TopDown κατά 12.42%, 36.9% και 5% αντίστοιχα, όταν ο αριθμός των partitions στην κατανεμημένη εκτέλεση ήταν 20.

2. Εξάρτηση από αριθμό partition

    - Η αύξηση του αριθμού των partitions (ή ισοδύναμα, η μείωση του μεγέθους του partition) μειώνει την απόδοση της ανωνυμοποίησης, επειδή σε αυτήν την περίπτωση ο Partitioner λαμβάνει περισσότερες αποφάσεις για την τελική κλάση ισοδυναμίας που θα καταλήξει κάθε tuple, με αποτέλεσμα να αφήνει στους αλγορίθμους ανωνυμοποίησης μικρότερο περιθώριο να τις καθορίσουν.
    - Η αύξηση του αριθμού των partition μειώνει το χρόνο εκτέλεσης του κατανεμημένου συστήματος, διότι με την αύξηση του αριθμού των partitions αυξάνεται με γραμμικό τρόπο ο αριθμός των εκτελούμενων tasks (στην ουρά εκτέλεσης), αλλά μειώνεται ο χρόνος εκτέλεσης κάθε task γιατί μειώνεται το μέγεθος της εισόδου του (με ρυθμό που είναι ταχύτερος από το γραμμικό). Συνολικά, ο χρόνος εκτέλεσης μειώνεται με την αύξηση του αριθμού των partitions, μέχρι κάποιο οριακό σημείο. Από το σημείο εκείνο και μετά, η μείωση του μεγέθους του partition δεν μειώνει το χρόνο εκτέλεσης κάθε task (διότι το μέγεθος του partition είναι ήδη αρκετά μικρό), με αποτέλεσμα να προστίθενται περισσότερα tasks στην ουρά προς εκτέλεση. Από το οριακό σημείο εκείνο και μετά δηλαδή, η αύξηση του αριθμού των partitions οδηγεί σε αύξηση του χρόνου εκτέλεσης της κατανεμημένης εφαρμογής.

3. Εξάρτηση από μέγεθος cluster

- Η μεταβολή του μεγέθους του cluster αφήνει ανεπηρέαστη την ποιότητα της ανωνυμοποίησης.
- Η μεταβολή του μεγέθους του cluster επηρεάζει με αντιστρόφως ανάλογο τρόπο το χρόνο εκτέλεσης της εφαρμογής. Διπλασιασμός των κόμβων οδηγεί σε υποδιπλασιασμό του χρόνου εκτέλεσης και το αντίστροφο.

4. Εξάρτηση από QID
   - Η αύξηση του αριθμού των attributes του QID μειώνει την απόδοση της ανωνυμοποίησης.
   - Η μεταβολή του αριθμού των attributes του QID δεν μεταβάλλει το χρόνο εκτέλεσης των μεθόδων ανωνυμοποίησης.
5. Εξάρτηση από k
   - Η αύξηση του k αυξάνει, γενικά, τους δείκτες GCP των μεθόδων ανωνυμοποίησης.
   - Η αύξηση του k μειώνει το χρόνο εκτέλεσης των μεθόδων ανωνυμοποίησης μέχρι ένα οριακό σημείο, μετά από το οποίο ο χρόνος εκτέλεσης των μεθόδων ανωνυμοποίησης παραμένει σταθερός.
   - Για πολύ μικρές τιμές του k ο αλγόριθμος TopDown είναι ο αποδοτικότερος όλων.
   - Για πολύ μεγάλες τιμές του k ο αλγόριθμος Mondrian είναι ο αποδοτικότερος όλων και ο αλγόριθμος TopDown είναι ο λιγότερο αποδοτικός.
6. Εξάρτηση από μέγεθος δεδομένων
   - Η μεταβολή του μεγέθους των δεδομένων επηρεάζει αντιστρόφως ανάλογα το χρόνο εκτέλεσης της εφαρμογής.
   - Αύξηση του μεγέθους των δεδομένων βελτιώνει αρχικά την ποιότητα της ανωνυμοποίησης μέχρι ένα σημείο στο οποίο ο αριθμός των partition γίνεται πολύ μεγάλος. Από το σημείο εκείνο και μετά αρχίζει και μειώνεται η ποιότητα της ανωνυμοποίησης (λόγω των συμπερασμάτων που αναφέρονται στο (2)).
7. Εξάρτηση από κατανομή δεδομένων
   - Ο αλγόριθμος TopDown χειρίζεται πιο αποδοτικά τις μη συμμετρικές κατανομές από τον αλγόριθμο Mondrian.
   - Η κατανομή των δεδομένων δεν επηρεάζει το χρόνο εκτέλεσης της εφαρμογής.

- Μια κατανομή που προκαλεί μεγάλη πόλωση στα δεδομένα, θα προκαλεί αύξηση του GCP για σχετικά μικρά μεγέθη dataset γιατί σε αυτήν την περίπτωση τα δεδομένα που είναι συσσωρευμένα σε ένα πολύ μικρό σημείο του χώρου θα χωρίζονται σε πολλά partitions, με αποτέλεσμα (όπως είδαμε και στο (2)) να αυξάνεται τελικά το GCP των μεθόδων ανωνυμοποίησης.

Τέλος, με το κατανεμημένο σύστημα που αναπτύξαμε καταφέραμε να ανωνυμοποιήσουμε με επιτυχία δεδομένα τα οποία αποτελούνταν από 1 μέχρι 100 εκατομμύρια στοιχεία, επιτυγχάνοντας ένα πολύ καλό utility για διάφορες κατανομές. Με τον τρόπο αυτό καταφέραμε να ανωνυμοποιήσουμε δεδομένα, τα οποία δεν θα μπορούσαν να ανωνυμοποιηθούν με κάποιο κεντρικό τρόπο, λόγω του πλήθους των στοιχείων και καταφέραμε επίσης να συνδυάσουμε την κατανεμημένη εκτέλεση της ανωνυμοποίησης με τη διατήρηση της ποιότητάς της.

Βιβλιογραφία

[1] L.Sweeney. k-anonymity: A model for protecting privacy. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 2002.

[2] K.LeFevre, D.J.DeWitt, and R.Ramakrishnan. Mondrian multidimensional k-anonymity. *ICDE '06 Proceedings of the 22nd International Conference on Data Engineering*, 2006.

[3] Adam Meyerson and Ryan Williams. On the complexity of optimal k-anonymity. *PODS '04 Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, 2004.

[4] Kristen LeFevre, David J. DeWitt, and Raghu Ramakrishnan. Incognito: Efficient full-domain k-anonymity. *SIGMOD '05 Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, 2005.

[5] Jiuyong Li, Raymond Chi-Wing Wong, Ada Wai-Chee Fu, and Jian Pei. Anonymisation by local recoding in data with attribute hierarchical taxonomies, 2008.

[6] Jian Xu, Wei Wanh, Jian Pei, Xiaoyuan Wang, Baile Shi, and Ada Wai-Chee Fu. Utility-based anonymization using local recoding. *KDD '06 Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2006.

[7] J.Dean and S.Ghemawat. Mapreduce: Simplified data processing on large clusters. *OSDI'04: Sixth Symposium on Operating System Design and Implementation*, 2004.

[8] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, 2009.

[9] Sheng Zhong. On distributed k-anonymization. *Fundamenta Informaticae*, 2009.

[10] Wei Jiang and Chris Clifton. A secure distributed framework for achieving k-anonymity. *The VLDB Journal — The International Journal on Very Large Data Bases*, 2006.

[11] Katerina Doka, Dimitrios Tsoumakos, and Nektarios Koziris. Kanis: Preserving k-anonymity over distributed data. *In proceedings of the 5th International Workshop on Personalized Access, Profile Management, and Context Awareness in Databases*, 2011.

[12] Gabriel Ghinita, Panagiotis Karras, Panos Kalnis, and Nikos Mamoulis. Fast data anonymization with low information loss. *VLDB '07 Proceedings of the 33rd international conference on Very large data bases*, 2007.

[13] C.Blake and C.Merz. Uci repository of machine learning databases, 1998.