ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ **Ε**ΠΙΚΟΙΝΩΝΙΩΝ, **Η**ΛΕΚΤΡΟΝΙΚΗΣ **Κ**ΑΙ **Σ**ΥΣΤΗΜΑΤΩΝ **Π**ΛΗΡΟΦΟΡΙΚΗΣ

Σχεδιασμός και μελέτη αναφοράς επεκτάσεων λογισμικού για τη μη σχεσιακή βάση δεδομένων MongoDB με σκοπό την ολοκληρωμένη παροχή συστήματος διαχείρισης ταυτόχρονων συνδιαλλαγών

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΦΙΛΙΠΠΟΥ Γ. ΚΑΤΕΡΙΝΟΠΟΥΛΟΥ

Επιβλέπων : Θεοδώρα Βαρβαρίγου
Καθηγήτρια Ε.Μ.Π.

Αθήνα, Φεβρουάριος 2015

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Σχεδιασμός και μελέτη αναφοράς επεκτάσεων λογισμικού για τη μη σχεσιακή βάση δεδομένων MongoDB με σκοπό την ολοκληρωμένη παροχή συστήματος διαχείρισης ταυτόχρονων συνδιαλλαγών

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΦΙΛΙΠΠΟΥ Γ. ΚΑΤΕΡΙΝΟΠΟΥΛΟΥ

Επιβλέπων : Θεοδώρα Βαρβαρίγου
Καθηγήτρια Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 12η Μαρτίου 2015.

.....	.....	.....
Βαρβαρίγου Θεοδώρα	Λούμος Βασίλειος	Ασκούνης Δημήτριος
Καθηγήτρια Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Μάρτιος 2015

.....

ΦΙΛΙΠΠΟΥ Γ. ΚΑΤΕΡΙΝΟΠΟΥΛΟΥ

Διπλωματούχου Ηλεκτρολόγου Μηχανικού και Μηχανικού Υπολογιστών Ε.Μ.Π.

Περίληψη

Στη σύγχρονη εποχή, η χρήση του υπολογιστικού νέφους και συνεπώς των συστημάτων νέφους εξυπηρέτησης αυξάνεται συνεχώς. Μαζί με τις τεχνολογίες αυτές, αυξάνονται όμως και οι απαιτήσεις σε ευέλικτο αποθηκευτικό χώρο. Γι' αυτόν το λόγο, αναπτύχθηκαν και διαδόθηκαν οι μη-σχεσιακές βάσεις δεδομένων. Αυτές οι βάσεις προσφέρουν τόσο την ευελιξία που χρειάζονται τα σύγχρονα συστήματα όσο και την ευχρηστία και την επάρκεια του χώρου αποθήκευσης.

Παρόλα αυτά, οι μη-σχεσιακές βάσεις δεδομένων παρουσιάζουν τα δικά τους μειονεκτήματα, με κυριότερο από αυτά να είναι η μη υποστήριξη συνδιαλλαγών και συνεπώς και των ACID ιδιοτήτων που αυτές προσφέρουν σε μία βάση δεδομένων. Γι' αυτό το λόγο, επεκτείνουμε μία μη-σχεσιακή βάση δεδομένων με σκοπό να υποστηρίζει τον έλεγχο ταυτόχρονων πολλαπλών εκδόσεων δεδομένων(MultiVersion Concurrency Control [1]), με σκοπό την υποστήριξη του προγράμματος του Διαχειριστή Συνδιαλλαγών, ο οποίος μπορεί να δώσει στο σύστημα τη δυνατότητα χρήσης συνδιαλλαγών για τη βάση δεδομένων και να εξασφαλίσει τις ιδιότητες αυτές.

Έτσι, σκοπός αυτής της διπλωματικής είναι η επέκταση της μη-σχεσιακής βάσης MongoDB σε ένα σύστημα με Έλεγχο Ταυτόχρονων Πολλαπλών Εκδόσεων, ώστε να παρέχεται ένα ολοκληρωμένο σύστημα πολλαπλών ταυτόχρονων συνδιαλλαγών. Παράλληλα, γίνεται αξιολόγηση της ολοκληρωμένης MongoDB σε σύγκριση με τη φυσική της έκδοση, μέσω εργαλείου λογισμικού για μετρήσεις σε βάσεις δεδομένων, με επέκταση για την υποστήριξη της συγκεκριμένης βάσης, ώστε να μελετηθεί πόσο αποτελεσματικό θα ήταν ένα τέτοιο σύστημα σε σχέση με το κόστος που επιφέρει σε χρόνο και χρήμα.

Λέξεις Κλειδιά: Μη-σχεσιακές βάσεις δεδομένων, Υπολογιστικό Νέφος, Συστήματα Εξυπηρέτησης Νέφους, MongoDB, Ταυτοχρονισμός Πολλαπλών Εκδόσεων, Διαχείριση Συνδιαλλαγών, YCSB, Java

Abstract

Nowadays, the use of Cloud Computing and with it the Cloud Serving Systems has begun growing rapidly. Consequently, the requirements in flexible data stores that can also host the Big Data of today have begun growing as much. Therefore, there has been developed and become popular a new paradigm of databases, the Non-Relational Databases (NoSQL). The Non-Relational Databases provide the flexibility, as well as the simplicity and storing space, required by the modern data systems.

In spite of all their advantages, Non-Relational Databases have their own disadvantages, with the biggest of those being not supporting Transactions and therefore the ACID properties that Transactions offer. Thus, non-relational Databases are extended to a MultiVersion Database, in order to support the Transactions Manager component, which enables the use of Transactions for the database and ensures the aforementioned properties.

Thus, the main purpose of this thesis project is the upgrade of the non-relational MongoDB to an integrated system, which includes Multi Version Concurrency Control, so that it can provide a system which offers transactional semantics. In addition, the integrated MongoDB system is evaluated, through a Cloud Serving System Benchmarking tool, which has been extended for supporting both the native and integrated versions of the MongoDB, in order investigate its performance and the tradeoffs between performance and support of transactional management.

Keywords: NoSQL Databases, Cloud Computing, Cloud Serving Systems, MongoDB, MultiVersion Concurrency, Transaction Management, YCSB, Java

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΠΙΝΑΚΑΣ ΕΙΚΟΝΩΝ

1

Εισαγωγή

1.1 Διαχείριση ταυτόχρονων συνδιαλλαγών σε μη σχεσιακές βάσεις δεδομένων

Τις τελευταίες δεκαετίες το διαδίκτυο έχει αναπτυχθεί και διαδοθεί στο μεγαλύτερο ποσοστό του ανεπτυγμένου κόσμου και πλέον αποτελεί κομμάτι της καθημερινότητας και της εργασίας για μεγάλο ποσοστό ανθρώπων. Οι τεχνολογίες με βάση το διαδίκτυο είναι ιδιαίτερα δημοφιλείς. Μία από αυτές είναι και το υπολογιστικό νέφος, το οποίο μπορεί να υποστηρίξει μεγάλο όγκο δεδομένων και απαιτήσεις σε υπολογιστική ισχύ, μέσω συμπλεγμάτων υπολογιστών. Έτσι, το υπολογιστικό νέφος αποτελεί συχνά μία οικονομική και αποδοτική λύση για τον επιχειρηματικό κόσμο και όχι μόνο.

Μαζί με την ανάπτυξή του διαδικτύου, όμως, αυξήθηκαν σημαντικά και οι απαιτήσεις σε αποθηκευτικό χώρο, ενώ λόγω της δυναμικότητας που το διέπει πλέον και της συνεχούς αυξομείωσης της ζήτησης στις εφαρμογές και σελίδες διαδικτύου, απαιτείται ευελιξία στις βάσεις δεδομένων. Έτσι, προχώρησε και η τεχνολογία των βάσεων δεδομένων και εξελίχθηκε από τις παλιότερες σχεσιακές βάσεις στις νέες μη-σχεσιακές βάσεις. Αυτές προσφέρουν την απαιτούμενη ευελιξία, την αποθήκευση μεγάλου όγκου δεδομένων και τις προγραμματιστικές δυνατότητες που χρειάζεται ένας προγραμματιστής εφαρμογών διαδικτύου.

Παρόλα αυτά οι μη-σχεσιακές βάσεις δεδομένων δεν προσφέρουν σημασιολογία Συνδιαλλαγών. Αυτό σημαίνει ότι δεν προσφέρονται οι ιδιότητες ACID [2]

(Atomicity, Consistency, Isolation και Durability), ενώ προκύπτουν προβλήματα σε κάποιες περιπτώσεις λόγω των κλειδωμάτων που χρησιμοποιούνται για συγχρονισμό στις φυσικές εκδόσεις των βάσεων, όπως είναι η παρεμπόδιση από κάποιο read ενός write και αντίστροφα.

Οι συνδιαλλαγές είναι ένα πολύ σημαντικό ζήτημα αφαιρετικότητας (abstraction) στο πρόγραμμα καθώς καταργούν δύο πολύ σημαντικά προβλήματα για τον προγραμματισμό εφαρμογών.

Το πρώτο είναι η αντιμετώπιση του ταυτοχρονισμού. Οι χρήστες δεν χρειάζεται να νοιάζονται για τον έλεγχό του, όταν προγραμματίζουν εφαρμογές με συνδιαλλαγές για να έχουν πρόσβαση σε διαμοιραζόμενα δεδομένα. Τα πρωτόκολλα που υλοποιούν την ιδιότητα της απομόνωσης θα χειριστούν τις ταυτόχρονες προσβάσεις.

Το δεύτερο είναι ότι οι εφαρμογές δε χρειάζεται να αντιμετωπίσουν τις αποτυχίες. Τα πρωτόκολλα ατομικότητας και ανθεκτικότητας παρέχουν αυτόματη επαναφορά σε περίπτωση αποτυχίας.

1.2 Αντικείμενο διπλωματικής

Σκοπός της παρούσας διπλωματικής είναι η δημιουργία, η ανάλυση και η αξιολόγηση ενός ολοκληρωμένου συστήματος παροχής υπηρεσιών, επεκτείνοντας τη μη-σχεσιακή βάση δεδομένων MongoDB σε μία έκδοση πολλαπλών εκδόσεων με υποστήριξη ταυτόχρονων συνδιαλλαγών. Στοχεύει ακόμη, στην αξιολόγηση του ολοκληρωμένου συστήματος, ως προς τις επιδόσεις του, και την επαλήθευση των μηχανισμών του.

1.2.1 Συνεισφορά

Το πρόβλημα της υποστήριξης της σημασιολογίας των Συνδιαλλαγών και των ιδιοτήτων ACID από τις μη-σχεσιακές βάσεις δεδομένων, μπορεί να λυθεί χρησιμοποιώντας ένα εξωτερικό πρόγραμμα Διαχείρισης Συνδιαλλαγών (Transactional Management). Αυτό με τη σειρά του μπορεί να συγχρονίσει τις συνδιαλλαγές της βάσης δεδομένων και ταυτόχρονα εγγυάται τις ACID δυνατότητες για το σύστημά μας.

Παρόλα αυτά, οι φυσικές (native) εκδόσεις των βάσεων δεδομένων δεν υποστηρίζουν τη χρήση του Διαχειριστή Συνδιαλλαγών. Απαιτείται μία έκδοση των βάσεων με Πολλαπλές Εκδόσεις (MultiVersion), προκειμένου να μπορεί να γίνει η ολοκλήρωση με τον Διαχειριστλή Συνδιαλλαγών.. Έτσι τελικά, δημιουργείται ένα συνολικό ολοκληρωμένο σύστημα βάσης δεδομένων με Έλεγχο Ταυτοχρονισμού Πολλαπλών Εκδόσεων (MultiVersion Concurrency Control). Στην παρούσα διπλωματική δημιουργείται ένα τέτοιο ολοκληρωμένο σύστημα για τη βάση δεδομένων MongoDB, με τον Ολιστικό Διαχειριστή Συνδιαλλαγών που παρέχεται από το πρόγραμμα CoherentPaaS [3] και την αντίστοιχη έκδοση του CoherentPaaS για τη MongoDB MVCC [1].

Ένα τέτοιο σύστημα, όμως, απαιτεί πόρους, κυρίως σε υποδομή, ενώ εισάγει επιπλέον φορτίο στην εφαρμογή, οπότε είναι απαραίτητη η επαλήθευση των μηχανισμών του και η αξιολόγηση της αποδοτικότητας και των επιδόσεών του. Αυτό γίνεται στην παρούσα διπλωματική, μέσω του εργαλείου μέτρησης (benchmark) Yahoo! Cloud Service Benchmark (YCSB) [4]. Έτσι, μελετώνται τα αποτελέσματα του τελικού συστήματος, συγκρίνονται με τη φυσική έκδοση της MongoDB και προκύπτουν συμπεράσματα για την αξιολόγηση του συστήματος σε σχέση με το κόστος του και για την περαιτέρω βελτίωση του.

1.3 Οργάνωση κειμένου

Η διπλωματική εργασία αυτή χωρίζεται σε 8 κεφάλαια. Στο 1ο κεφάλαιο παρουσιάζεται ο σκοπός της εργασίας και των προβλημάτων τα οποία σκοπεύει να λύσει. Στο 2ο γίνεται περιγραφή των βασικών θεωρητικών εννοιών που απαιτούνται στην πλήρη κατανόηση της εργασίας. Το 3ο κεφάλαιο αναφέρεται στις απαιτήσεις του συστήματος και την ανάλυσή τους. Στο 4ο κεφάλαιο γίνεται αναφορά στη συνολική σχεδίαση του συστήματος και την αρχιτεκτονική του. Στο 5ο κεφάλαιο περιγράφεται η υλοποίηση του συστήματος, περιγράφοντας τις τεχνολογίες που χρησιμοποιήθηκαν και τις κλάσεις των διαφόρων μερών του. Στο 6ο κεφάλαιο περιγράφεται ο Έλεγχος στο σύστημά μας, που είναι και ο τελικός σκοπός της εργασίας. Παρουσιάζεται το πρόγραμμα ελέγχου YCSB, τα αποτελέσματα και τα συμπεράσματα πάνω σε αυτά. Στο κεφάλαιο 7 γίνεται μία σύνοψη της διαδικασίας, παρουσιάζονται τα συμπεράσματα και οι δυνατές επεκτάσεις και επεμβάσεις στο σύστημα προς βελτίωση αυτού. Τέλος, στο 8ο κεφάλαιο παρουσιάζεται η απαραίτητη βιβλιογραφία και οι αναφορές, οι οποίες περιέχονται σε όλη την έκταση της παρούσας εργασίας.

2

Περιγραφή βασικών εννοιών

Στην παρούσα ενότητα περιγράφονται σημαντικές έννοιες και τεχνολογίες που απαιτούνται για την κατανόηση της εργασίας. Συγκεκριμένα αναφέρονται οι μη-σχεσιακές βάσεις δεδομένων, οι ταυτόχρονες συνδιαλλαγές και η διαχείρισή τους, η έννοια του Υπολογιστικού Νέφους και η βάση δεδομένων MongoDB, στην οποία βασίζεται το σύστημά μας.

2.1 Εισαγωγή στις μη σχεσιακές βάσεις δεδομένων

2.1.1 Εισαγωγή

Μία Βάση Δεδομένων (Database – DB) είναι μία οργανωμένη συλλογή δεδομένων. Τα δεδομένα συνήθως οργανώνονται ούτως ώστε να μοντελοποιήσουν πλευρές της πραγματικότητας, με τέτοιον τρόπο που να υποστηρίζει διεργασίες, οι οποίες απαιτούν πληροφορία. Για παράδειγμα, η μοντελοποίηση της διαθεσιμότητας δωματίων σε ένα ξενοδοχείο, γίνεται με τέτοιον τρόπο, ώστε να υποστηρίζεται η εύρεση ξενοδοχείου με κενά δωμάτια.

Τα Συστήματα Διαχείρισης Βάσεων Δεδομένων (Database Management Systems – DBMS) είναι εφαρμογές λογισμικού υπολογιστή που αλληλεπιδρούν με το χρήστη, άλλες εφαρμογές και με την ίδια τη βάση δεδομένων, ώστε να συλλάβει και να αναλύσει δεδομένα. Ένα DBMS γενικού-σκοπού είναι σχεδιασμένο να επιτρέπει τον ορισμό, τη δημιουργία, τα ερωτήματα, την ενημέρωση και τη διαχείριση των βάσεων δεδομένων. Τα πιο διάσημα συστήματα διαχείρισης βάσεων δεδομένων είναι η MySQL, η PostgreSQL, ο Microsoft SQL Server, το Oracle, το SAP και η IBM DB2. Συνήθως, μία βάση δεδομένων δεν είναι φορητή μεταξύ διαφορετικών DBMSs, αλλά διαφορετικά DBMS μπορούν να δια-λειτουργούν χρησιμοποιώντας πρότυπα όπως η SQL και το ODBC ή JDBC που επιτρέπουν σε μία εφαρμογή να λειτουργήσει με πάνω από ένα DBMS. Τα συστήματα διαχείρισης βάσεων δεδομένων κατηγοριοποιούνται ανάλογα με το μοντέλο βάσης δεδομένων που χρησιμοποιούν. Τα πιο δημοφιλή DBMS από το 1980 και ύστερα, υποστήριζαν το σχεσιακό μοντέλο όπως εκπροσωπείται από τη γλώσσα SQL.

Εικόνα 1: Βάση Δεδομένων

Τα υπάρχοντα συστήματα διαχείρισης βάσεων δεδομένων παρέχουν διάφορες λειτουργίες, οι οποίες επιτρέπουν τη διαχείριση μίας βάσης και των δεδομένων της και μπορούν να κατηγοριοποιηθούν στις εξής λειτουργικές ομάδες:

- **Καθορισμός Δεδομένων:** Δημιουργία, τροποποίηση και αφαίρεση ορισμών που καθορίζουν την οργάνωση των δεδομένων.
- **Ενημέρωση:** Εισαγωγή, τροποποίηση και διαγραφή των πραγματικών δεδομένων.
- **Ανάκτηση:** Παροχή πληροφοριών σε μία μορφή, η οποία να είναι άμεσα χρησιμοποιήσιμη ή για περαιτέρω επεξεργασία από άλλες εφαρμογές. Τα ανακτημένα δεδομένα μπορεί να είναι διαθέσιμα σε μορφή βασικά ίδια με αυτήν που αποθηκεύτηκαν στη βάση δεδομένων ή σε μία νέα, η οποία αποκτήθηκε μεταβάλλοντας ή συνδυάζοντας υπάρχοντα δεδομένα της βάσης.
- **Διαχείριση:** Η καταχώρηση και παρακολούθηση χρηστών, η επιβολή της ασφάλειας των δεδομένων, η παρακολούθηση της επίδοσης, η διατήρηση της ακεραιότητας των δεδομένων, η αντιμετώπιση προβλημάτων ταυτόχρονης χρήσης και η επανάκτηση φθαρμένων δεδομένων.

Μία βάση δεδομένων μαζί με το σύστημα διαχείρισής της συμμορφώνονται προς τα πρότυπα ενός συγκεκριμένου μοντέλου βάσης δεδομένων, δηλαδή ενός τύπου μοντέλου δεδομένων που καθορίζει τη λογική δομή της βάσης δεδομένων και τον τρόπο με το οποίο τα δεδομένα αυτά μπορεί να αποθηκευτούν, να οργανωθούν και να γίνουν αντικείμενο χειρισμού. [5] Το μοντέλο της βάσης, το σύστημα διαχείρισης της και η ίδια η βάση δεδομένων αποτελούν συνολικά το «Σύστημα Βάσης Δεδομένων» (Database System). [6]

Οι εξυπηρετητές βάσεων δεδομένων (database servers) είναι υπολογιστές αφιερωμένοι σε αυτήν τη λειτουργία, στους οποίους διατηρούνται οι πραγματικές βάσεις δεδομένων και οι οποίοι τρέχουν μόνο το DBMS και σχετικό λογισμικό. Οι εξυπηρετητές αυτοί είναι συνήθως υπολογιστές πολλαπλών επεξεργαστών, με μεγάλη μνήμη και συστοιχίες δίσκων RAID για την σταθερή αποθήκευση. Το RAID χρησιμοποιείται και για την επανάκτηση δεδομένων, σε περίπτωση που κάποιος δίσκος αποτύχει. Υλικοί επιταχυντές βάσεων δεδομένων, οι οποίοι συνδέονται σε έναν ή περισσότερους εξυπηρετητές μέσω ενός καναλιού υψηλής-ταχύτητας χρησιμοποιούνται σε περιβάλλοντα που παρουσιάζουν μεγάλο όγκο επεξεργασίας συνδιαλλαγών δεδομένων.

Οι βάσεις δεδομένων, τέλος, μπορεί να κατηγοριοποιηθούν σύμφωνα με τα μοντέλα βάσεων δεδομένων που υποστηρίζουν (για παράδειγμα σχεσιακές ή XML), τον τύπο του υπολογιστή στον οποίο τρέχουν (από ένα σύμπλεγμα εξυπηρετητών μέχρι ενός κινητού τηλεφώνου), τη γλώσσα ερωτημάτων (query language) που χρησιμοποιούν για την πρόσβαση στη βάση (όπως για παράδειγμα SQL και XQuery) και την εσωτερική μηχανική τους, που επηρεάζει τις επιδόσεις, την επεκτασιμότητα, την αντοχή και την ασφάλειά τους.

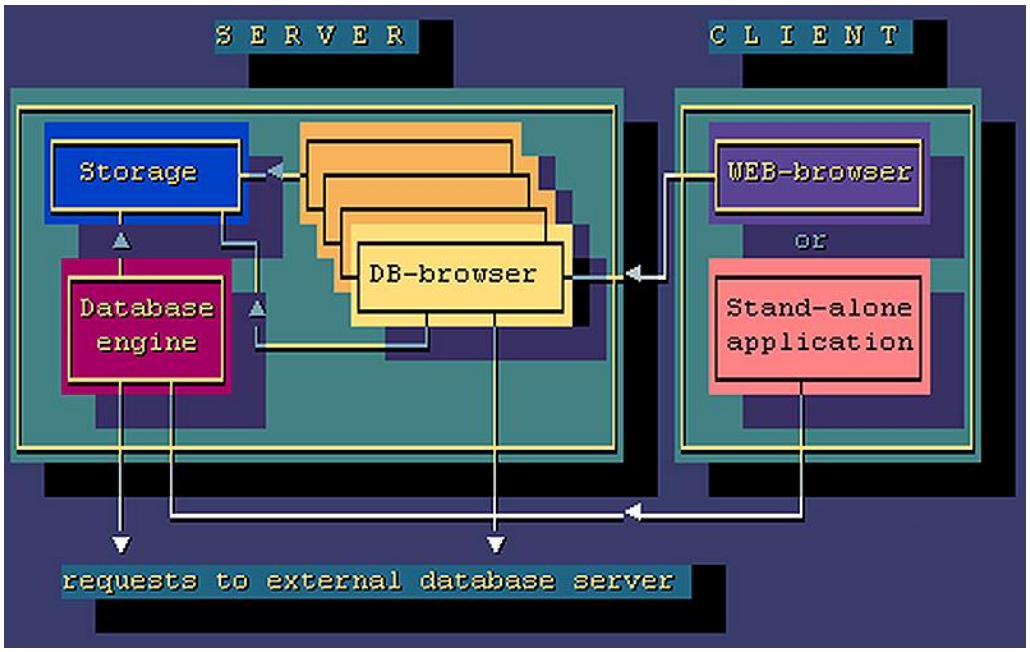
2.1.2 Εξέλιξη των βάσεων δεδομένων

Ακολουθώντας την πρόοδο της τεχνολογίας σε περιοχές όπως η επεξεργαστική ισχύς, η μνήμη του υπολογιστή, ο αποθηκευτικός χώρος και τα δίκτυα υπολογιστών, το μέγεθος, οι δυνατότητες και οι επιδόσεις των βάσεων δεδομένων και των αντίστοιχων συστημάτων διαχείρισής τους έχουν αυξηθεί κατακόρυφα. Η ανάπτυξη της τεχνολογίας των βάσεων δεδομένων χωρίζεται σε τρεις περιόδους: πλοηγούμενες, SQL/σχεσιακές και μετά-σχεσιακές. [7]

Δεκαετία του 1960 - Πλοηγούμενα DBMS

Τη δεκαετία του 1960, καθώς οι πάροχοι άρχισαν να διαφημίζουν υπολογιστικοποιημένες τεχνολογίες λογιστικών για χρήση στην παραγωγή και ευρύτερη εργαστηριακή χρήση, άρχισαν να παρουσιάζονται τα συστήματα διαχείρισης βάσεων δεδομένων, τα οποία επέτρεπαν την αποθήκευση μεγάλων ποσοτήτων δεδομένων. Το δύσκολο ζήτημα της οργάνωσης των εγγραφών στο μέσο αποθήκευσης για βέλτιστη πρόσβαση, διαχειριζόταν πλέον από ένα υποσύστημα που αποκαλούταν DBMS.

Προέκυψαν δύο πρότυπα, ένα μοντέλο προτυποποιημένο από το Σύστημα Διαχείρισης Πληροφοριών της IBM (IBM's Information Management System) και το μοντέλο «Δικτύου». Η χρήση των δύο αντιστοίχως αποδεικνύει πόσο σημαντικές γίνονταν πλέον οι βάσεις δεδομένων. Η πρώτη αναπτύχθηκε σε συνεργασία με το πρόγραμμα Apollo για να καταγραφούν σε κατάλογο τα υλικά που απαιτούνταν για τον πύραυλο σελήνης Saturn V και το διαστημικό όχημα Apollo καθώς και εργασίες παραγωγής της General Electrics. Η ώθηση της επιχείρησης του τομέα της επεξεργασίας δεδομένων άρχισε να είναι εμφανής: το συνέδριο CODASYL (Conference on Data Systems Languages), το μεγαλύτερο συνέδριο της δεκαετίας, αντί να αποτελείται από ακαδημαϊκές οντότητες, αποτελούταν κυρίως από επιχειρήσεις όπως η General Motors και η US Steel. [8]



Εικόνα 2: Σύστημα Διαχείρισης Βάσεων Δεδομένων (Εικόνα από το DBMS ConteXt)

Η προσέγγιση του CODASYL βασίστηκε στην «χειρονακτική» πλοήγηση ενός συνδεδεμένου συνόλου δεδομένων που σχηματίστηκε σε ένα μεγάλο δίκτυο. Οι εφαρμογές μπορούσαν να βρουν τις εγγραφές με ένα εκ των εξής τρόπων:

I. Χρήση ενός πρωτεύοντος κλειδιού – primary key (γνωστό και ως CALC key, συνήθως αποτέλεσμα κατακερματισμού)
II. Πλοήγηση των σχέσεων –relationships (αποκαλούμενα σύνολα – sets) από τη μία εγγραφή σε μία άλλη
III. Σαρώνοντας όλες τις εγγραφές σε ακολουθιακή σειρά

Τα μεταγενέστερα συστήματα προσέθεσαν Β-Δέντρα (B-Trees) για να παρέχουν εναλλακτικά μονοπάτια πρόσβασης. Πολλές βάσεις δεδομένων CODASYL προσέθεσαν και μία αρκετά ευθεία γλώσσα ερωτημάτων. Παρόλα αυτά, εν τέλει, η προσέγγιση του CODASYL ήταν πολύ σύνθετη και απαιτούσε σημαντική εκπαίδευση και προσπάθεια για την παραγωγή χρήσιμων εφαρμογών.

Δεκαετία του 1970 - Σχεσιακές Βάσεις Δεδομένων

Ο Edgar Codd, ο οποίος αργότερα εργάστηκε για το Εργαστήριο Έρευνας της IBM στο San Jose το 1973, ξεκίνησε το επαναστατικό του μοντέλο βάσεων δεδομένων δηλώνοντας:

Οι μελλοντικοί χρήστες μεγάλων τραπεζών δεδομένων πρέπει να προφυλάσσονται από το να πρέπει να γνωρίζουν τον τρόπο που οργανώνονται τα δεδομένα μέσα στο μηχάνημα, δηλαδή την εσωτερική αναπαράστασή τους.

Αναφερόταν έτσι, άμεσα, στα προβλήματα που είχαν αναγνωρισθεί στο πρότυπο της πλοήγησης: Ο κάθε χρήστης έπρεπε να πλοηγηθεί μέσω ενός σημαντικού μεγέθους πολυπλοκότητας για να μεταβεί στα επιθυμητά δεδομένα. Το μοντέλο του Codd, το οποίο αρθροποιήθηκε στο «A Relational Model of Data for Large Shared Data Banks», ήταν μία επαναστατική ιδέα: αντί τα δεδομένα να αντιμετωπίζονται ως απλά μέσα οργάνωσης, οι βάσεις δεδομένων μπορούσαν πλέον να χρησιμοποιούνται ως ένα εργαλείο για ερωτήματα δεδομένων για την εύρεση σχέσεων που κρύβονταν σ' αυτά.

Οι σχεσιακές βάσεις δεδομένων διαχώρισαν τα δεδομένα από τις εφαρμογές που είχαν πρόσβαση σ' αυτά, επιτρέποντας έτσι τη μεταχείριση της πληροφορίας μέσω της χρήσης μίας γλώσσας ερωτημάτων (query language), όπου η επιλογή συγκεκριμένων δεδομένων μπορούσε να επιτευχθεί αποδοτικά μέσω της δημιουργίας δηλώσεων (statements) που περιείχαν λογικούς τελεστές.

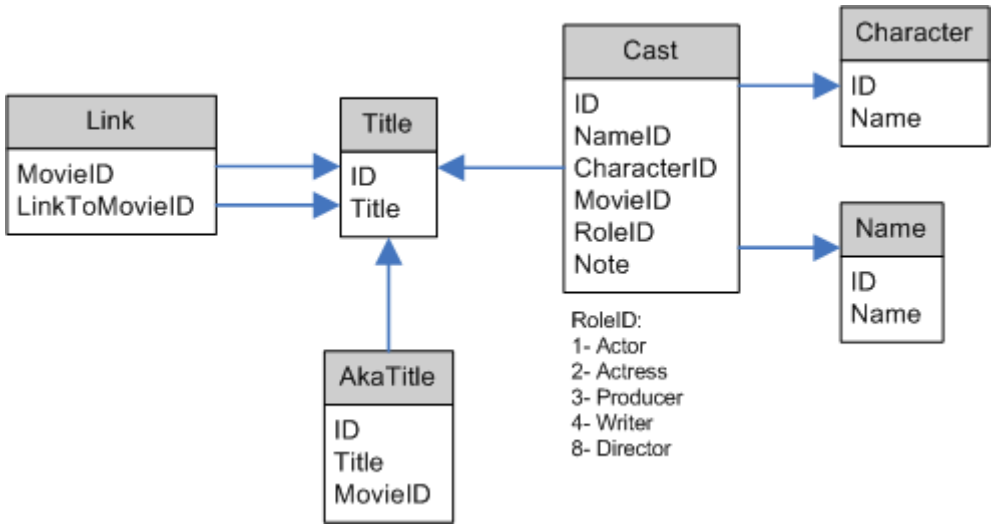
Έτσι, το 1974 η IBM ανέπτυξε ένα πρωτότυπο σχεσιακό μοντέλο βάσεων δεδομένων που αποκαλούταν System R και στη συνέχεια θα εξελισσόταν στην ευρέως χρησιμοποιούμενη βάση δεδομένων Δομημένη Γλώσσα Ερωτημάτων (Structured Query Language – SQL) το 1981. Παρόλα αυτά, η Oracle (ως «Εταιρεία Σχεσιακού Λογισμικού») ήταν η πρώτη που εμπορικοποίησε την τεχνολογία της το 1979, σε μία περίοδο που οι σχεσιακές βάσεις δεδομένων εξελίσσονταν στην κυρίαρχη μορφή αποθήκευσης για τη ψηφιακή οικονομία. [8]

Το σχεσιακό μοντέλο επέτρεπε την ελεύθερη είσοδο, διαγραφή και τροποποίηση δεδομένων στους πίνακες (tables – δηλαδή σύνολα δεδομένων με τον ίδιο τύπο

οντοτήτων το καθένα και με σταθερό μέγεθος). Το μοντέλο αυτό επέτρεπε, επίσης, στα περιεχόμενα της βάσης να εξελιχθούν χωρίς συνεχές επαναγράψιμο των συνδέσμων και των δεικτών. Το σχεσιακό μέρος προέρχεται από το γεγονός ότι οι οντότητες αναφέρονταν σε άλλες οντότητες σε αυτό που αποκαλείται σχέση ενός-προς-πολλούς, όπως τα παραδοσιακά ιεραρχικά μοντέλα, καθώς και σχέσεις πολλών-προς-πολλούς, όπως ένα πλοηγούμενο δικτυακό μοντέλο.

Ένα παράδειγμα τέτοιου συστήματος βάσεως δεδομένων είναι η καταγραφή πληροφοριών για χρήστες, το όνομά τους, πληροφορίες εισόδου, διάφορες διευθύνσεις και τηλέφωνά τους. Στην πλοηγούμενη προσέγγιση όλα αυτά θα καταγράφονταν σε μία μοναδική εγγραφή και τα μη-χρησιμοποιούμενα αντικείμενα δε θα τοποθετούνταν στη βάση. Αντίθετα, στη σχεσιακή προσέγγιση, τα δεδομένα θα κανονικοποιούνταν σε ένα πίνακα χρήστη, ένα πίνακα διευθύνσεων και ένα πίνακα τηλεφώνων. Οι εγγραφές θα δημιουργούνταν μέσα σ' αυτούς τους προαιρετικούς πίνακες μόνο αν οι διευθύνσεις ή τα τηλέφωνα ήταν πραγματικά διαθέσιμα.

Η σύνδεση των πληροφοριών μεταξύ τους είναι το κλειδί γι' αυτό το σύστημα. Στο σχεσιακό μοντέλο μερικές πληροφορίες χρησιμοποιούνται ως «κλειδί», ορίζοντας έτσι με μοναδικό τρόπο μία συγκεκριμένη εγγραφή. Όταν συλλέγονται οι πληροφορίες για ένα χρήστη, οι πληροφορίες που βρίσκονται σε προαιρετικούς πίνακες μπορεί να βρεθούν ψάχνοντας γι' αυτό το κλειδί.



Εικόνα 3: Παράδειγμα σχεσιακής βάσης δεδομένων για αρχειοθέτηση ταινιών

Δεκαετία του 2000 - Μη-σχεσιακές και Νέο-σχεσιακές Βάσεις Δεδομένων (NoSQL και NewSQL Databases)

Η γενιά μετά-σχεσιακών βάσεων δεδομένων τη δεκαετία του 2000 ονομάστηκε NoSQL βάσεις δεδομένων και περιέχει γρήγορες αποθηκεύσεις κλειδιού-τιμής και αρχειοστρεφείς βάσεις. Οι NoSQL βάσεις δεδομένων είναι συχνά πολύ γρήγορες, δεν απαιτούν προκαθορισμένα σχήματα πινάκων, αποφεύγουν λειτουργίες συνδυασμού αποθηκεύοντας από-κανονικοποιημένα δεδομένα και είναι σχεδιασμένες για οριζόντια επεκτασιμότητα (δηλαδή την προσθήκη

περισσότερων κόμβων σε ένα σύστημα, όπως την προσθήκη ενός νέου υπολογιστή σε μία κατανεμημένη εφαρμογή λογισμικού).

Οι πιο διάσημες βάσεις NoSQL είναι η MongoDB[1], η Couchbase[2], η Riak, η memcached, η Apache Cassandra και η HBase. [9] Οι βάσεις NewSQL είναι μία μοντέρνα κλάση σχεσιακών βάσεων δεδομένων που στοχεύει στην παροχή παρόμοιας επεκτάσιμης απόδοσης με τις NoSQL βάσεις για επεξεργασία online συνδιαλλαγών από φορτία, ενώ χρησιμοποιούν ακόμη την SQL και διατηρούν τις ACID πιστοποιήσεις (Ατομικότητα, Συνέπεια, Απομόνωση, Ανθεκτικότητα) ενός παραδοσιακού συστήματος διαχείρισης βάσεων δεδομένων. Τέτοιες βάσεις δεδομένων είναι οι ScaleBase, Clustrix, EnterpriseDB, MemSQL, NuoDB και VoltDB. [10]

Το χαρακτηριστικό που διαχωρίζει τις NoSQL βάσεις δεδομένων είναι η ουσιαστική απόρριψη της «σχεσιακής δόμησης των δεδομένων» που ήταν ενσωματωμένη στα σχεσιακά συστήματα διαχείρισης βάσεων δεδομένων (RDBMS). Η ώθηση των εταιρειών να στραφούν στις μη-σχεσιακές βάσεις είναι η πρόσφατη απότομη αύξηση του όγκου των συναλλαγών που πρέπει να καταγραφούν, καθώς υπάρχει μεγάλος όγκος online κίνησης. Αυτό σε συνδυασμό με την έλευση φθηνού online αποθηκευτικού χώρου (όπως προσφέρουν τα υπολογιστικά νέφη) έχει καταστήσει τις NoSQL βάσεις δεδομένων αρκετά δημοφιλείς. Παρέχουν πιο εύκολες ad-hoc αλλαγές και το δυναμισμό που χρειάζονται τα παραπάνω συστήματα, σε πολύ μεγαλύτερο βαθμό απ' ότι οι σχεσιακές βάσεις δεδομένων. Η δημιουργία μίας σχεσιακής βάσης δεδομένων περιλαμβάνει έρευνα και εξέταση του ποια δεδομένα απαιτείται να καταγραφούν, ούτως ώστε να δημιουργηθεί το «σχήμα» της βάσης δεδομένων. Αντίθετα, μία βάση δεδομένων NoSQL επιτρέπει την αποθήκευση κάθε είδους δεδομένων (ακόμη και αυτών που δεν μπορεί να προβλεφθούν κατά τη δημιουργία της βάσης). [8]

[1] http://www.mongodb.org/

[2] http://www.couchbase.com/

2.1.3 Η ανάγκη και τα οφέλη NoSQL βάσεων δεδομένων

Η τεχνολογία των NoSQL βάσεων δεδομένων αναπτύχθηκε αρχικά από τις κυρίαρχες εταιρείες διαδικτύου – μεταξύ αυτών η Google, το Facebook, το Amazon και το Linkedln – για να υπερκεράσουν τους περιορισμούς που έθετε η παλαιωμένη τεχνολογία των σχεσιακών βάσεων δεδομένων. Σήμερα, οι εταιρείες υιοθετούν τις NoSQL βάσεις δεδομένων για μεγάλο αριθμό περιπτώσεων χρήσης, μία επιλογή στην οποία οδηγήθηκαν εξαιτίας των σύγχρονων τάσεων της τεχνολογίας: Μεγάλοι Χρήστες (Big Users), Μεγάλα Δεδομένα (Big Data), η μεγάλη χρήση του Διαδικτύου και το Υπολογιστικό Νέφος (Cloud Computing).

Μεγάλοι Χρήστες (Big Users): Πριν από όχι πολύ καιρό, 1000 χρήστες ημερησίως για μία εφαρμογή θεωρούταν μεγάλο νούμερο και 10000 ακραία περίπτωση. Σήμερα, σχεδόν 3 δισεκατομμύρια χρήστες είναι συνδεδεμένοι στο διαδίκτυο και ο χρόνος που περνούν online αυξάνεται σταθερά, δημιουργώντας έτσι μία έκρηξη στον αριθμό των ταυτόχρονων χρηστών. Δεν είναι ασυνήθιστο για μία εφαρμογή να έχει εκατομμύρια διαφορετικούς χρήστες μέσα σε μία μέρα και να πρέπει να υποστηρίξει χρήστες από κάθε μεριά της γης 24 ώρες την ημέρα και 365 μέρες το χρόνο.

Η υποστήριξη μεγάλου αριθμού ταυτόχρονων χρηστών είναι σημαντική, αλλά επειδή οι απαιτήσεις της χρήσης των εφαρμογών είναι δύσκολο να προβλεφθούν, είναι σημαντικό να υποστηρίζονται δυναμικά πολλοί ταυτόχρονοι χρήστες, ο αριθμός των οποίων αυξάνεται ή μειώνεται ανάλογα. Μία εφαρμογή μπορεί να απαιτεί μεγάλο αριθμό χρηστών λόγω:

- Είναι μία εφαρμογή, η οποία μόλις κυκλοφόρησε και γίνεται ανάρπαστη, αυξάνοντας το μέγεθός της από μηδέν σε ένα εκατομμύριο χρήστες σε μία βραδιά.
- Το γεγονός πως κάποιοι χρήστες είναι συχνά ενεργοί, ενώ άλλοι χρησιμοποιούν μία εφαρμογή μερικές φορές και μετά την εγκαταλείπουν.
- Οι εορταστικές περίοδοι, όπως τα Χριστούγεννα, μπορούν να δημιουργήσουν μεγάλες απότομες αυξήσεις κίνησης για μικρές περιόδους.
- Η κυκλοφορία νέων προϊόντων και η προώθηση προϊόντων μπορούν να αυξήσουν δραματικά τη χρήση μίας εφαρμογής.

Μεγάλος αριθμός χρηστών σε συνδυασμό με τη δυναμική φύση προτύπων χρήσης οδηγεί στην ανάγκη για μία ευκόλως επεκτάσιμη τεχνολογία βάσεων δεδομένων. Με τις σχεσιακές τεχνολογίες, πολλοί προγραμματιστές βλέπουν ότι είναι δύσκολο ή ακατόρθωτο να πετύχουν τη δυναμική επεκτασιμότητα και το μέγεθος κλίμακας που απαιτούν, ενώ ταυτόχρονα θέλουν να διατηρήσουν υψηλή την απόδοσή των συστημάτων τους. Έτσι πολλοί προτιμούν τις NoSQL βάσεις δεδομένων.

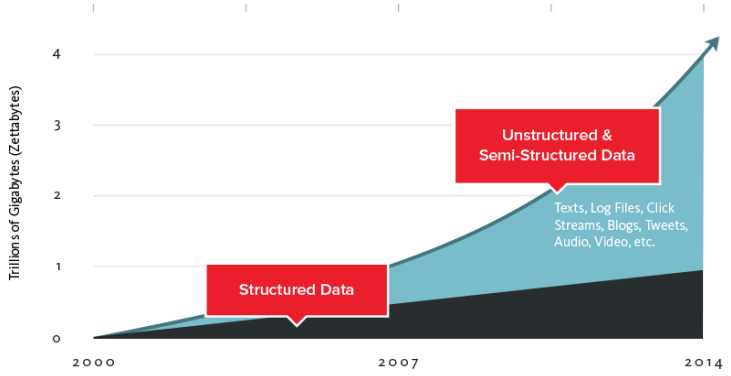
Μεγάλα Δεδομένα (Big Data): Η απότομη αύξηση της χρήσης του διαδικτύου, σε συνδυασμό με τη χρήση εφαρμογών για κινητά και κοινωνικών εφαρμογών, καθώς και οι επικοινωνίες μηχάνημα-προς-μηχάνημα (machine-to-machine communications) οδηγούν στην επανάσταση των «μεγάλων δεδομένων». Η ερευνητική εταιρεία IDC εκτιμά πως το 2013 το συνολικό παγκόσμιο μέγεθος των ψηφιακών δεδομένων ήταν 4.4 zettabytes, δηλαδή 4.4 τρισεκατομμύρια gigabytes. Αυτό το μέγεθος αναμένεται να αυξηθεί σε 44 zettabytes μέχρι το 2020.

Η σύλληψη και πρόσβαση των δεδομένων γίνεται συνεχώς πιο εύκολη μέσω τρίτων, όπως το Facebook και Dun and Bradstreet. Προσωπικές πληροφορίες χρηστών, γεωλογικά δεδομένα, κοινωνικά διαγράμματα, περιεχόμενο παραγόμενο από τους χρήστες, δεδομένα καταλόγου των μηχανημάτων και δεδομένα παραγόμενα από αισθητήρες είναι μόνο μερικά από τα παραδείγματα της διαρκώς αυξανόμενης λίστας των δεδομένων που αποθηκεύονται.

Δεν είναι έκπληξη πως οι προγραμματιστές βρίσκουν αυξανόμενη αξία στη μόχλευση των δεδομένων αυτών, με σκοπό τον εμπλουτισμό των υπαρχουσών εφαρμογών και τη δημιουργία νέων. Η διαθεσιμότητα αυτών των δεδομένων αλλάζει με γρήγορο ρυθμό τη φύση της επικοινωνίας, της αγοράς, της διαφήμισης, της ψυχαγωγίας και της διαχείρισης σχέσεων. Οι εφαρμογές που δεν τις εκμεταλλεύονται θα αποτύχουν σύντομα.

Παρόλα αυτά, η αποθήκευση και χρήση μεγάλων δεδομένων απαιτεί έναν πολύ διαφορετικό τύπο βάσης δεδομένων. Οι προγραμματιστές χρειάζονται μία πολύ ευέλικτη λύση που να μπορεί εύκολα να φιλοξενήσει οποιονδήποτε τύπο δεδομένων θα χρειαστεί να χρησιμοποιήσουν και δεν θα επηρεαστεί από δομικές αλλαγές από τρίτους παρόχους. Μεγάλο μέρος των νέων δεδομένων είναι αδόμητο ή ημι-δομημένο, οπότε οι προγραμματιστές χρειάζονται μια βάση δεδομένων για την αποδοτική αποθήκευσή τους. Δυστυχώς όμως, η σχεσιακή προσέγγιση με το άκαμπτο σχήμα της καθιστά αδύνατη την ταχεία υιοθέτηση νέων τύπων δεδομένων και δεν ταιριάζει γι' αυτού του είδους δεδομένα.

Αντίθετα, μία βάση NoSQL παρέχει ένα σαφώς πιο ευέλικτο, μοντέλο δεδομένων χωρίς-σχήμα που ταιριάζει καλύτερα στην οργάνωση των δεδομένων μιας εφαρμογής και απλοποιεί την αλληλεπίδραση μεταξύ της εφαρμογής και της βάσης, με αποτέλεσμα να απαιτείται η συγγραφή, η διόρθωση και η συντήρηση λιγότερου κώδικα.



Εικόνα 4: Η ραγδαία αύξηση του μεγέθους των δεδομένων

Η χρήση του Διαδικτύου: Ο όγκος των δεδομένων που παράγονται από μηχανήματα αυξάνεται με τον πολλαπλασιασμό της ψηφιακής τηλεμετρίας και της χρήσης του διαδικτύου. Τη σύγχρονη εποχή, περίπου 20 δισεκατομμύρια συσκευές είναι συνδεδεμένες στο διαδίκτυο – τα πάντα, από υπολογιστές tablet μέχρι οικιακές συσκευές, συστήματα στα αυτοκίνητα, σε νοσοκομεία και αποθήκες. Πολλές από αυτές τις συσκευές λαμβάνουν δεδομένα για το περιβάλλον, την τοποθεσία, την κίνηση, τη θερμοκρασία, τον καιρό και άλλα από τους 50 δισεκατομμύρια αισθητήρες τους.

Οι πρωτοπόρες εταιρείες εκμεταλλεύονται τη χρήση του διαδικτύου για την ανάπτυξη νέων προϊόντων και υπηρεσιών, τη μείωση των εξόδων και του χρόνου για αγορά, την αύξηση της αποδοτικότητας, την εξάλειψη της ζημίας και την αύξηση της ικανοποίησης των πελατών. Αυτή η δυνατότητα για πρόσβαση σε παγκόσμια, λειτουργικά δεδομένα σε πραγματικό χρόνο επιτρέπει τη δυναμική, πληροφορημένη λήψη αποφάσεων και την αύξηση της επιχειρηματικής ευελιξίας.

Οι σχεσιακές βάσεις δεδομένων, οι οποίες απαιτούν ένα προκαθορισμένο σχήμα και δομή δεδομένων, δεν μπορούν να διαχειριστούν αποτελεσματικά τα δεδομένα τηλεμετρίας. Για την αντιμετώπιση αυτών των προβλημάτων, οι πρωτοποριακές επιχειρήσεις βασίζονται στην τεχνολογία των NoSQL βάσεων δεδομένων για να επεκτείνουν την ταυτόχρονη πρόσβαση σε δεδομένα εκατομμύρια συνδεδεμένων συσκευών και συστημάτων, να αποθηκεύσουν δισεκατομμύρια σημεία δεδομένων και να επιτύχουν τις απαιτήσεις απόδοσης σε λειτουργίες κρίσιμων αποστολών και δομών.

Το Υπολογιστικό Νέφος (Cloud Computing): Σήμερα, οι περισσότερες εφαρμογές τρέχουν σε ένα δημόσιο, ιδιωτικό ή υβριδικό νέφος, υποστηρίζουν μεγάλο αριθμό χρηστών και χρησιμοποιούν αρχιτεκτονική διαδικτύου τριών-επιπέδων. Σε μία τέτοια αρχιτεκτονική, οι εφαρμογές γίνονται αντικείμενο πρόσβασης μέσω ενός περιηγητή διαδικτύου ή μία κινητή εφαρμογή που συνδέεται στο διαδίκτυο. Στο νέφος, ένας εξισορροπητής φορτίου διευθύνει την εισερχόμενη κίνηση στο επίπεδο μίας μεγαλύτερης-κλίμακας από εξυπηρετητές δικτύου/εφαρμογών για τη διεργασία της λογικής των εφαρμογών.

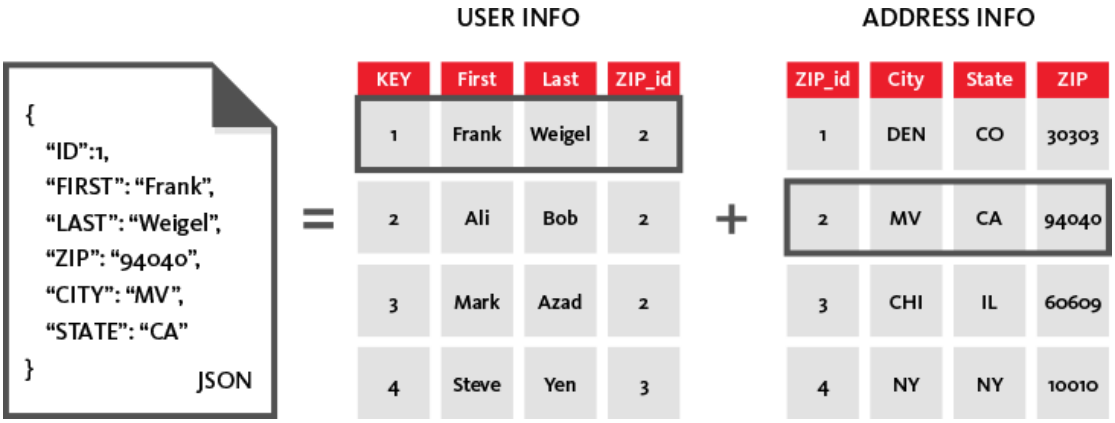
Αυτό το επίπεδο λειτουργεί υπέροχα. Για κάθε 10000 νέους χρήστες, μπορείς απλά να προσθέσεις έναν εξυπηρετητή στο επίπεδο δικτυακής εφαρμογής για να απορροφήσει το φορτίο. Στο επίπεδο βάσης δεδομένων, οι σχεσιακές βάσεις ήταν η δημοφιλής επιλογή. Η χρήση τους, όμως, ήταν προβληματική, καθώς είναι μία κεντροστρεφής τεχνολογία που τα πάντα διαμοιράζονται και κλιμακώνεται προς τα πάνω αντί προς τα έξω. Αυτό τις καθιστά κακή επιλογή για εφαρμογές που απαιτούν εύκολη και δυναμική επεκτασιμότητα, σε αντίθεση με τις NoSQL βάσεις. Οι NoSQL χτίζονται από κάτω προς τα πάνω για να είναι κατανεμημένες και επεκτάσιμες-προς-τα-έξω τεχνολογίες και συνεπώς είναι πολύ καλύτερες γι' αυτήν την αρχιτεκτονική του υπολογιστικού νέφους. [11]

2.1.4 Λειτουργικές διαφορές

Τα μοντέλα δεδομένων των σχεσιακών και των NoSQL βάσεων είναι πολύ διαφορετικά. Το σχεσιακό μοντέλο παίρνει τα δεδομένα και τα διαχωρίζει σε αλληλένδετους πίνακες που περιέχουν γραμμές και στήλες. Οι πίνακες αυτοί αναφέρονται ο ένας στον άλλον μέσω των ξένων κλειδιών (foreign keys) που είναι αποθηκευμένα σε στήλες τους. Όταν αναζητά κάποιος δεδομένα, οι επιθυμητές πληροφορίες πρέπει να συλλεγούν από πολλούς πίνακες (συχνά εκατοντάδες στις σημερινές εμπορικές εφαρμογές) και να συνδυαστούν πριν να γίνουν διαθέσιμες στην εφαρμογή. Παρόμοια, κατά την εγγραφή δεδομένων, η εγγραφή απαιτεί συνεργασία και εκτέλεση από πολλούς πίνακες.

Οι NoSQL βάσεις δεδομένων έχουν ένα πολύ διαφορετικό μοντέλο. Για παράδειγμα, μία έγγραφο-στρεφής (document-oriented) NoSQL βάση δεδομένων παίρνει τα δεδομένα που χρειάζεται να αποθηκευτούν και τα αθροίζει σε έγγραφα χρησιμοποιώντας το φορμά JSON. Το κάθε JSON έγγραφο μπορεί να θεωρηθεί ως ένα αντικείμενο που χρησιμοποιείται από την εφαρμογή. Ένα τέτοιο έγγραφο μπορεί να πάρει όλα τα δεδομένα που είναι αποθηκευμένα σε μία γραμμή που φτάνει σε 20 πίνακες μίας σχεσιακής βάσης δεδομένων και να τα συναθροίσει σε ένα μοναδικό έγγραφο/αντικείμενο.

Η άθροιση αυτών των δεδομένων μπορεί να οδηγήσει σε διπλές εγγραφές (duplication) αλλά εφόσον πλέον ο αποθηκευτικός χώρος είναι αρκετά φθηνός, η ευελιξία του προκύπτοντος μοντέλου, η αποτελεσματικότητα του στην κατανομή των προκυπτόντων εγγράφων και η βελτίωση της απόδοσης των λειτουργιών ανάγνωσης και εγγραφής καθιστούν αυτό το μοντέλο προτιμότερο για εφαρμογές βασισμένες στο διαδίκτυο.



Εικόνα 5: Παράδειγμα συνάθροισης των δεδομένων από πίνακες σε ένα έγγραφο JSON

Μία ακόμη σημαντική διαφορά είναι πως οι σχεσιακές βάσεις δεδομένων έχουν άκαμπτα σχήματα ενώ το μοντέλο των NoSQL είναι χωρίς-σχήμα. Η σχεσιακή τεχνολογία απαιτεί αυστηρό καθορισμό ενός σχήματος πριν από την αποθήκευση δεδομένων σε μία βάση. Η αλλαγή του σχήματος, αφού τα δεδομένα έχουν εισαχθεί, είναι προβληματική και συνήθως αποφεύγεται, κάτι το οποίο είναι ένα μεγάλο πρόβλημα στην εποχή των μεγάλων δεδομένων, κατά την οποία οι προγραμματιστές εφαρμογών χρειάζεται να εισάγουν συνεχώς και

ταχέως νέους τύπους δεδομένων για τον εμπλουτισμό των εφαρμογών τους. Αντιθέτως, οι βάσεις δεδομένων εγγράφων είναι χωρίς-σχήμα, γεγονός που τις καθιστά ιδανικές για την ελεύθερη προσθήκη πεδίων σε έγγραφα JSON χωρίς να απαιτείται πρώτα καθορισμός των αλλαγών. Το φορμά με το οποίο εισέρχονται τα δεδομένα μπορεί να αλλάξει οποιαδήποτε στιγμή, χωρίς τη διακοπή της εφαρμογής. [11]

Τα χαρακτηριστικά των NoSQL βάσεων δεδομένων συνοψίζονται παρακάτω:

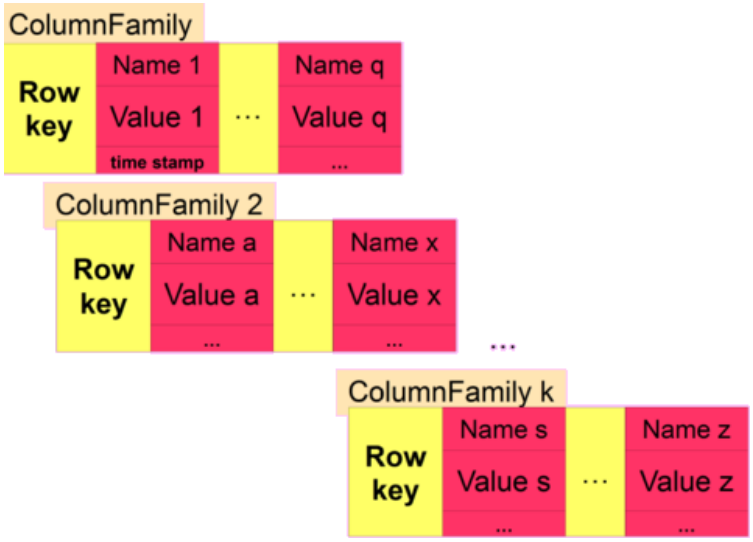
- **Μεγάλο μέγεθος:** Οι NoSQL βάσεις όπως είδαμε και παραπάνω είναι ιδανικές για περιπτώσεις μεγάλων δεδομένων.
- **Υψηλές επιδόσεις (ταχύτητες) εγγραφής:** Σε μία εποχή που το διαδίκτυο χρησιμοποιείται διαρκώς και απαιτείται η αποθήκευση τόσο μεγάλου όγκου δεδομένων, η εγγραφή των δεδομένων στις βάσεις είναι σημαντικό να γίνεται όσο το δυνατόν ταχύτερα.
- **Γρήγορη πρόσβαση κλειδιού-τιμής:** Αυτό είναι ένα από τα πιο σημαντικά στοιχεία των NoSQL βάσεων. Όταν η καθυστέρηση είναι σημαντική, μία από τις βέλτιστες επιλογές είναι ο κατακερματισμός σύμφωνα με ένα κλειδί και η ανάγνωση της τιμής απευθείας από τη μνήμη ή με μία μόνο επίσκεψη στο σκληρό δίσκο. Δεν έχουν όλες οι βάσεις NoSQL ως στόχο την ταχύτητα, αλλά σε αυτό το κομμάτι υπερτερούν.
- **Ευέλικτο σχήμα και ευέλικτες βάσεις δεδομένων:** Όπως είδαμε και με το JSON οι NoSQL βάσεις είναι ιδανικές όταν απαιτείται δυναμική και ευέλικτη αλλαγή των δεδομένων σε μία βάση.
- **Μετατροπή σχήματος:** Το γεγονός πως οι NoSQL βάσεις δεν έχουν σχήμα, τις καθιστά ως μια καλή επιλογή όταν απαιτείται μία μετατροπή σχήματος.
- **Διαθεσιμότητα για εγγραφή:** Όταν απαιτείται η επιτυχία όλων των λειτουργιών εγγραφής (write) μπορούμε να το πετύχουμε με διαμερισμό, CAP και τη συνέπεια που προσφέρουν οι NoSQL βάσεις.
- **Γενική διαθεσιμότητα παράλληλου υπολογιστή:** Ο παράλληλος υπολογιστής είναι κομμάτι του μέλλοντος.
- **Εύκολη χρήση για τους προγραμματιστές:** Η πρόσβαση στα δεδομένα θα πρέπει να είναι εύκολη. Παρότι οι σχεσιακές βάσεις είναι εύχρηστες για τους τελικούς χρήστες, για τους προγραμματιστές απαιτούν μεγαλύτερη προσπάθεια. Αντίθετα οι NoSQL είναι ιδανικές για προγραμματιστές.
- **Υποστήριξη κατανεμημένων συστημάτων:** Ένα κατανεμημένο σύστημα θα πρέπει να μπορεί να διευρύνεται με κέντρα δεδομένων, ενώ ταυτόχρονα να αντιμετωπίζει τα σφάλματα χωρίς πρόβλημα. Τα συστήματα NoSQL, επειδή επικεντρώνονται στην κλίμακα, συνηθίζουν να εκμεταλλεύονται τα διαμερίσματα, να μην χρησιμοποιούν αυστηρά πρωτόκολλα συνέπειας και είναι σωστά σχεδιασμένα να λειτουργούν σε κατανεμημένα σενάρια.

2.1.5 Είδη NoSQL Βάσεων δεδομένων

Έχουν υπάρξει αρκετές προσεγγίσεις ως προς την κατηγοριοποίηση των NoSQL βάσεων δεδομένων. Λόγω της ποικιλίας των προσεγγίσεων και των κοινών τους σημείων είναι δύσκολο να δοθεί μία συνολική εικόνα των μη-σχεσιακών βάσεων. Ένας καλός τρόπος διαχωρισμού τους είναι με βάση το μοντέλο δεδομένων που χρησιμοποιεί η κάθε μία. Με αυτό το κριτήριο ακολουθούν οι κατηγορίες των NoSQL βάσεων.

NoSQL Βάσεις με Στήλες

Μία στήλη (column) ενός κατανεμημένου αποθηκευτικού χώρου είναι ένα αντικείμενο NoSQL του χαμηλότερου δυνατού επιπέδου σε ένα χώρο-κλειδιών (keyspace). Keyspace είναι ο χώρος αποθήκευσης που διατηρεί μαζί όλες τις οικογένειες στηλών ενός σχεδίου.



Εικόνα 6: Παράδειγμα ενός keyspace, με τις οικογένειες στηλών που τον αποτελούν

Η στήλη είναι μία πλειάδα (tuple), δηλαδή ένα ζεύγος κλειδιού-τιμής, που αποτελείται από τρία στοιχεία:

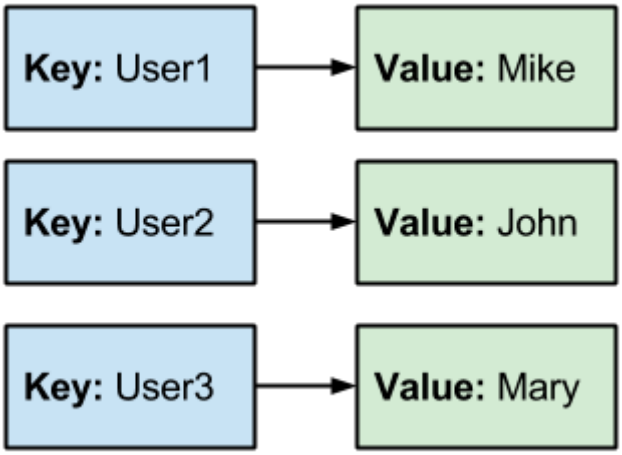
- Μοναδικό όνομα: Χρησιμοποιείται ως αναφορά για τη στήλη
- Τιμή: Είναι το περιεχόμενο της στήλης. Μπορεί να είναι τύπου AsciiType, LongType, TimeUUIDType, UTF8Type, και άλλα.
- Στάμπα χρονοσήμανσης (timestamp): Το timestamp του συστήματος που χρησιμοποιείται για να καθοριστεί η εγκυρότητα των περιεχομένων.

Παραδείγματα τέτοιων βάσεων δεδομένων είναι η HBase, η BigTable και η HyperTable.

NoSQL Βάσεις με Key-Value

Είναι η λιγότερο περίπλοκη επιλογή NoSQL. Έχουν σχεδιαστεί για την αποθήκευση δεδομένων χωρίς-σχήμα. Με την αποθήκευση τιμής-κλειδιού, όλα

τα δεδομένα αποτελούνται από ένα κλειδί που ανήκει σε ευρετήριο (indexed key) και μία τιμή.

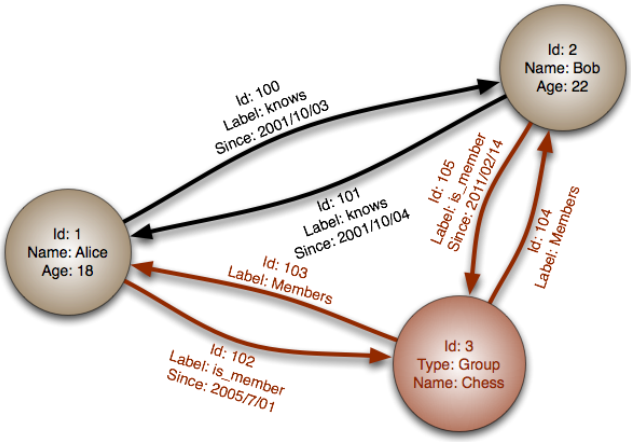


Εικόνα 7: Ένα απλό παράδειγμα Key-Value αποθήκευσης

Παραδείγματα τέτοιων βάσεων είναι η Cassandra[3], η DyanmoDB[4], η Azure Table Storage, η Riak[5] και η BerkeleyDB.

NoSQL Βάσεις με Γράφους

Οι βάσεις NoSQL με γράφους βασίζονται στη θεωρία γράφων. Αυτές οι βάσεις έχουν σχεδιαστεί για δεδομένα, των οποίων οι σχέσεις μπορεί να αναπαρασταθούν καλά από γράφους και έχουν στοιχεία που είναι διασυνδεδεμένα, με ένα μη καθορισμένο αριθμό σχέσεων μεταξύ τους.



Εικόνα 8: Παράδειγμα βάσης δεδομένων με γράφους

[3] http://cassandra.apache.org/

[4] *aws.amazon.com/**dynamodb**/*

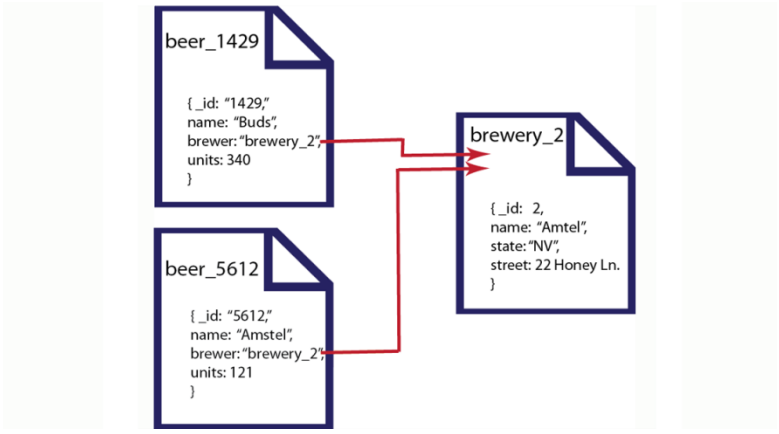
[5] *basho.com/**riak**/*

Τέτοιου είδους βάσεις δεδομένων είναι η Neo4J και η Polyglot.

NoSQL Βάσεις Εγγράφων

Αυτού του είδους οι βάσεις δεδομένων επεκτείνουν τη βασική ιδέα των key-value βάσεων, όπου τα «έγγραφα» περιέχουν πιο πολύπλοκα δεδομένα και το κάθε έγγραφο έχει ένα μοναδικό κλειδί που χρησιμοποιείται για την ανάκτησή του.

Οι βάσεις εγγράφων είναι σχεδιασμένες για την αποθήκευση, την ανάκτηση και τη διαχείριση εγγραφο-στρεφών πληροφοριών, οι οποίες αποκαλούνται και μη-δομημένες πληροφορίες. Είναι η κύρια κατηγορία των NoSQL βάσεων. Η χρησιμότητά τους περιγράφηκε και στο προηγούμενο κεφάλαιο, για την αποθήκευση μεγάλων δεδομένων, τομέα στον οποίο οι σχεσιακές βάσεις υστερούν.



Εικόνα 9: Παράδειγμα μίας απλής βάσης δεδομένων εγγράφων, για μία εταιρεία ζυθοποιίας

Σε αυτήν την κατηγορία ανήκουν η MongoDB και η CouchDB.

Μοντέλο Δεδομένων	***Επιδόσεις***	***Επεκτασι-μότητα***	***Ευελιξία***	***Πολυπλο-κότητα***	***Λειτουρ-γικότητα***
Key-Value Store	Υψηλές	Υψηλή	Υψηλή	Καμία	Ποικίλει
Column Store	Υψηλές	Υψηλή	Μέτρια	Χαμηλή	Ελάχιστη
Document Store	Υψηλές	Ποικίλει	Υψηλή	Χαμηλή	Ποικίλει
Graph Database	Ποικίλουν	Ποικίλει	Υψηλή	Υψηλή	Θεωρία Γράφων

2.2 Διαχείριση ταυτόχρονων συνδιαλλαγών

Ο σκοπός του ολοκληρωμένου συστήματος που προκύπτει είναι η διαχείριση ταυτόχρονων συνδιαλλαγών. Στο σύστημά μας, τη Διαχείριση των Συνδιαλλαγών πραγματοποιεί το κομμάτι του ολιστικού Διαχειριστή Συνδιαλλαγών (Transactions Manager, TM), όπως θα περιγραφεί στο κεφάλαιο 3.

Ο ολιστικός TM παρέχει ένα σύνολο μερών που υλοποιούν τη διαχείριση συνδιαλλαγών. Συγκεκριμένα παρέχει έναν κατάλογο, ένα μετρητή στιγμιότυπων και εύρεση συγκρούσεων. Αυτά τα μέρη είναι αρκετά αποσυνδεδεμένα μεταξύ τους, ώστε να είναι δυνατή η ανεξάρτητη οριζόντια επεκτασιμότητα τους. Ο ολιστικός διαχειριστής συνδιαλλαγών υλοποιεί απομόνωση στιγμιότυπων και έτσι ανιχνεύει τις συγκρούσεις write-write και συνεπώς όλοι οι αποθηκευτικοί χώροι που χρησιμοποιούνται σε μία συνδιαλλαγή πρέπει να παρέχουν δεδομένα πολλαπλών-εκδόσεων, ώστε οι ταυτόχρονες συνδιαλλαγές να διαβάζουν δεδομένα χωρίς να μπλοκάρονται από μία λειτουργία ενημέρωσης στο ίδιο αντικείμενο. Οι αποθηκευτικοί χώροι είτε χρησιμοποιούν το δικό τους κατάλογο, είτε παρέχουν τις πληροφορίες καταλόγου στον ολιστικό TM.

Οι εφαρμογές μπορούν να κάνουν συνδιαλλαγές απευθείας ή μέσω της κοινής γλώσσας ερωτημάτων. Και στις δύο περιπτώσεις, χρησιμοποιείται ένας ενδιάμεσος εξυπηρετητής (proxy server) για τον ολιστικό TM στην πλευρά του πελάτη. Σε κάθε αποθηκευτικό χώρο απαιτείται και ένας τοπικός TM. [3]

2.2.1 ACID Χαρακτηριστικά στις βάσεις δεδομένων – Η έννοια της συνδιαλλαγής

Συνδιαλλαγή είναι μία ακολουθία από λειτουργίες δεδομένων που εκτελούνται ατομικά. Παρέχουν τις λεγόμενες ACID ιδιότητες, δηλαδή [3]:

- **Ατομικότητα – Atomicity**: Καθιστά τις συνδιαλλαγές ως λειτουργίες που ακολουθούν το όλα-ή-τίποτα σε περίπτωση αποτυχιών. Αυτό σημαίνει, ότι το αποτέλεσμα μία συνδιαλλαγής θα ήταν το «όλα» αν επιτύχει (δηλαδή η συνδιαλλαγή κάνει commit) ή το «τίποτα» αν αποτύχει (αν απορριφθεί ή επιστρέψει σε προηγούμενη κατάσταση).
- **Συνέπεια – Consistency:** Παρέχεται από την εφαρμογή. Ο κώδικας της εφαρμογής σε μία συνδιαλλαγή θα πρέπει να εγγυάται πως αν η βάση δεδομένων έχει μια συνεπή κατάσταση, τότε η νέα κατάσταση της βάσης δεδομένων (μετά τη συνδιαλλαγή) θα είναι επίσης συνεπής-σταθερή.
- **Απομόνωση – Isolation:** Παρέχει ατομικότητα συγχρονισμού. Παρέχει δηλαδή την αίσθηση ότι ο χρήστης εκτελεί τη συνδιαλλαγή μόνος του στο σύστημα, ακόμη κι αν εκτελούνται πολλές συνδιαλλαγές ταυτόχρονα.

- **Ανθεκτικότητα – Durability:** Εγγυάται πως οι ενημερώσεις μίας επιτυχούς (και συνεπώς committed) συνδιαλλαγής δεν χάνονται ακόμα και σε περίπτωση σφαλμάτων και αποτυχιών.

Οι συνδιαλλαγές είναι ένα πολύ σημαντικό ζήτημα αφαιρετικότητας (abstraction) στο πρόγραμμα καθώς καταργούν δύο πολύ σημαντικά προβλήματα για τον προγραμματισμό εφαρμογών.

Το πρώτο είναι η αντιμετώπιση του ταυτοχρονισμού. Οι χρήστες δεν χρειάζεται να νοιάζονται για τον έλεγχό του, όταν προγραμματίζουν εφαρμογές με συνδιαλλαγές για να έχουν πρόσβαση σε διαμοιραζόμενα δεδομένα. Τα πρωτόκολλα που υλοποιούν την ιδιότητα της απομόνωσης θα χειριστούν τις ταυτόχρονες προσβάσεις.

Το δεύτερο είναι ότι οι εφαρμογές δε χρειάζεται να αντιμετωπίσουν τις αποτυχίες. Τα πρωτόκολλα ατομικότητας και ανθεκτικότητας παρέχουν αυτόματη επαναφορά σε περίπτωση αποτυχίας.

Έτσι οι ιδιότητες ACID , τις οποίες προσφέρουν οι συνδιαλλαγές, απλοποιούν σε σημαντικό βαθμό τη δουλειά των προγραμματιστών. [3]

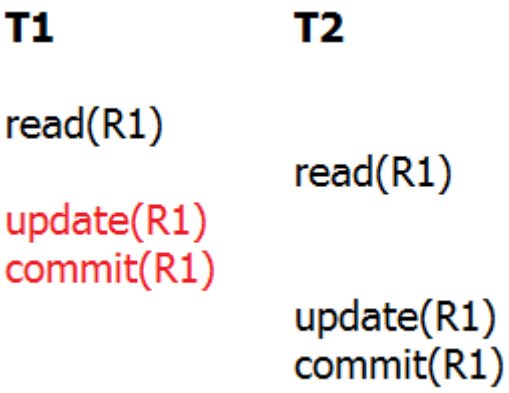
2.2.2 Επίπεδα απομόνωσης και φαινόμενα/ανωμαλίες ανάγνωσης

Τα προβλήματα που παρουσιάζουν τα παραδοσιακά DBMS που δεν υποστηρίζουν τις ταυτόχρονες συνδιαλλαγές, είναι συνήθως προβλήματα που προέρχονται από την έλλειψη των ACID ιδιοτήτων τους και κυρίως από την απουσία της απομόνωσης (isolation)

Η απομόνωση ορίζει πως η ακεραιότητα των συνδιαλλαγών είναι ορατή σε άλλους χρήστες και συστήματα. Για παράδειγμα, όταν ένας χρήστης δημιουργεί μία παραγγελία αγοράς και έχει δημιουργήσει την κεφαλή, αλλά όχι τις γραμμές, καθορίζει αν η κεφαλή είναι διαθέσιμη και ορατή για άλλα συστήματα και χρήστες ή όχι.

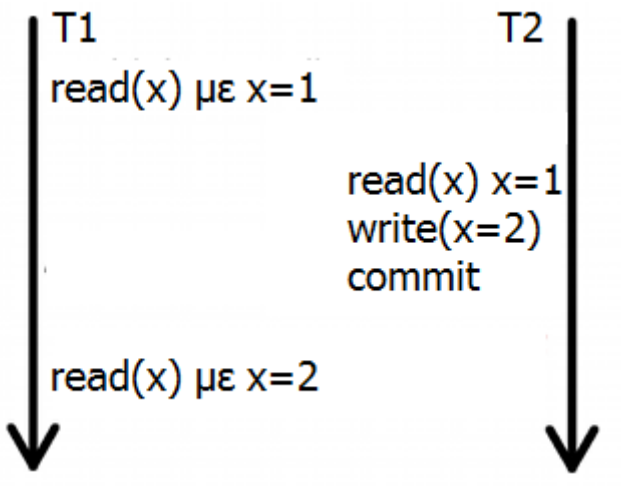
Ένα χαμηλό επίπεδο απομόνωσης αυξάνει τη δυνατότητα πολλών χρηστών να έχουν πρόσβαση στα δεδομένα την ίδια στιγμή, αλλά αυξάνει τα προβλήματα που προέρχονται από τον αριθμό των ταυτόχρονων προσβάσεων. Τέτοια είναι:

- **Dirty reads:** Το πρόβλημα με τα dirty reads προκύπτει όταν μία συνδιαλλαγή μπορεί να διαβάσει δεδομένα που έχουν μεταβληθεί αλλά όχι παραδοθεί (committed) από μία άλλη συνδιαλλαγή.
- **Χαμένες ενημερώσεις (Lost updates):** Το πρόβλημα αυτό προκύπτει όταν μία συνδιαλλαγή γράφει πάνω (over-writes) από τις αλλαγές που έγιναν από μία άλλη συνδιαλλαγή, επειδή δεν αναγνωρίζει ότι τα δεδομένα έχουν μεταβληθεί. Για παράδειγμα, εάν μία συνδιαλλαγή T1 διαβάσει την εγγραφή R1, ακολουθούμενη από μία δεύτερη συνδιαλλαγή T2 που διαβάζει την R1, τότε η πρώτη συνδιαλλαγή (T1) ενημερώνει την R1 και παραδίδει (commits) την αλλαγή, και στη συνέχεια η T2 ενημερώνει την R1 και επίσης παραδίδει την αλλαγή. Σε αυτήν την περίπτωση η ενημέρωση που έγινε από την T1 στην εγγραφή R1 «χάνεται», καθώς η T2 δεν την βλέπει ποτέ.



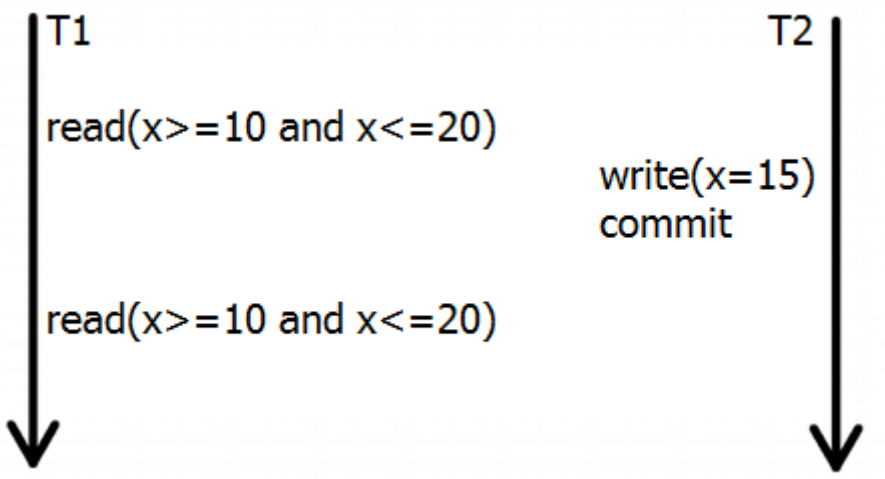
Εικόνα 10: Το πρόβλημα του Lost Update

- **Μη-επαναλαμβανόμενες αναγνώσεις (non-repeatable reads):** Το πρόβλημα αυτό εμφανίζεται όταν μία συνδιαλλαγή βρίσκει ότι μία εγγραφή, την οποία έκανε read προηγουμένως, έχει μεταβληθεί από μία άλλη συνδιαλλαγή.



Εικόνα 11: Εδώ η T1 βρίσκει αλλαγμένο το x σε δύο συνεχόμενα reads

- **Phantom reads:** Το πρόβλημα αυτό παρουσιάζεται όταν μία συνδιαλλαγή ανακαλύπτει πως το ίδιο ερώτημα SQL επιστρέφει διαφορετικά σύνολα από εγγραφές, σε διαφορετικές χρονικές στιγμές μέσα στη συνδιαλλαγή.



Εικόνα 12: Εδώ το 2ο ερώτημα για το x από την T1 μπορεί να φέρει διαφορετικά αποτελέσματα σε κάποιες περιπτώσεις στο 1ο και στο 2ο read

Γενικά, για μία βάση δεδομένων, όταν επιχειρεί να έχει το μέγιστο επίπεδο απομόνωσης υπάρχουν δύο επιλογές. Ένα παραδοσιακό σύστημα DMBS θα χρησιμοποιήσει κλειδώματα, ενώ στη δική μας περίπτωση προτιμούμε το MultiVersion Concurrency Control, το οποίο βέβαια από την μεριά του θα αποφέρει ίσως μία μείωση στις ταυτόχρονες προσβάσεις στη βάση.

Οι βάσεις δεδομένων προσφέρουν συνήθως επιλογή για το επίπεδο απομόνωσης των συνδιαλλαγών τους. Στις περισσότερες, οι συνδιαλλαγές μπορεί σε γενικές γραμμές να δημιουργούνται έτσι ώστε να αποφεύγουν τα υψηλά επίπεδα απομόνωσης, με στόχο να μειωθεί το επιπλέον κόστος (overhead) στο σύστημα.

Τα επίπεδα της απομόνωσης είναι τα εξής:

- **Serializable:** Αποτελεί το μέγιστο επίπεδο απομόνωσης. Με ένα DBMS με κλειδώματα, απαιτεί τα κλειδώματα των read και write να απελευθερώνονται στο τέλος της συνδιαλλαγής. Επίσης, χρειάζονται κλειδώματα-εύρους (range locks) όταν ένα ερώτημα SELECT χρησιμοποιήσει τη συνθήκη WHERE, για να αποφύγει τα phantom reads, συνήθως χρειάζοντας να κλειδώσει ολόκληρο τον πίνακα. Χρησιμοποιώντας MVCC, δεν εμφανίζονται phantom reads χωρίς να χρειαστεί να κλειδωθούν εγγραφές κατά τη διάρκεια της συνδιαλλαγής.
- **Repeatable Reads:** Σε αυτό το επίπεδο απομόνωσης, ένα DBMS με κλειδώματα κρατά τα κλειδώματα για read και write μέχρι τέλους της συνδιαλλαγής. Μπορεί όμως να προκύψουν phantom reads, γιατί δεν υπάρχουν κλειδώματα-εύρους.
- **Read Committed**: Σε αυτό το επίπεδο, ένα DBMS με κλειδώματα κρατάει write κλειδώματα μέχρι το τέλος της συνδιαλλαγής, αλλά τα κλειδώματα των write απελευθερώνονται στο τέλος, ενω τα reads δεν κρατιόνται για όλη τη διάρκεια της συνδιαλλαγής.
  Αυτό σημαίνει ότι εγγυάται ότι οποιοδήποτε δεδομένο γίνει read θα κάνει commit τη στιγμή που γίνεται read. Αποτρέπει να φανεί κάποιο dirty read. Αλλά, δεν εγγυάται ότι αν ξαναγίνει το read θα βρει τα ίδια δεδομένα, καθώς κάποια ταυτόχρονη συνδιαλλαγή μπορεί να άλλαξε την τιμή μιας εγγραφής, καθώς δεν υπήρχε κρατημένο ένα read lock
- **Read Uncommitted:** Σε αυτό το επίπεδο, έχουμε τη χαμηλότερη απομόνωση. Δεν αποτρέπονται τα dirty reads.

2.2.3 2Phase-Locking, κλειδώματα εγγραφών και επίδραση στην επίδοση

Στις συνδιαλλαγές, το πρωτόκολλο των κλειδωμάτων-2φάσεων (2Phase-Locking) είναι ένας τρόπος ελέγχου του ταυτοχρονισμού που εγγυάται τη σειριοποιησιμότητα. Το πρωτόκολλο αυτό χρησιμοποιεί κλειδώματα (locks), τα οποία εφαρμόζονται από μία συνδιαλλαγή στα δεδομένα, και τα οποία μπορεί να μπλοκάρουν άλλες συνδιαλλαγές από την πρόσβαση σε αυτά, κατά τη διάρκεια της συνδιαλλαγής.

Το πρωτόκολλο αυτό αποτελείται από δύο φάσεις:

- Τη φάση επέκτασης (extending phase): παίρνουν τα κλειδώματα και κανένα δεν απελευθερώνεται
- Τη φάση σμίκρυνσης(shrinking phase): τα κλειδώματα απελευθερώνονται και κανένα δεν αποκτάται.

Τα παραδοσιακά συστήματα DBMS για να ξεπεράσουν αυτά τα προβλήματα χρησιμοποιούν κλειδώματα συνήθως με το πρωτόκολλο που αναφέρθηκε. Υπάρχουν δύο είδη κλειδωμάτων:

1 Τα **μοιραζόμενα κλειδώματα (shared locks)** που χρησιμοποιούνται για να προστατεύσουν τα reads των εγγραφών. Είναι συμβατά με άλλα μοιραζόμενα κλειδώματα αλλά όχι με αποκλειστικά κλειδώματα. Γι' αυτό πολλές συνδιαλλαγές μπορούν να έχουν πολλαπλά τέτοια κλειδώματα σε μία μοναδική εγγραφή, αλλά μία συνδιαλλαγή που θέλει να βάλει αποκλειστικό κλείδωμα πρέπει να περιμένει να απελευθερωθεί από τα μοιραζόμενα.

2 Τα **αποκλειστικά κλειδώματα (exclusive locks)** που χρησιμοποιούνται για να προστατεύσουν τα writes στις εγγραφές. Μόνο μία συνδιαλλαγή μπορεί να κρατά ένα αποκλειστικό κλείδωμα σε μία χρονική στιγμή, και επιπλέον, δεν είναι συμβατά με τα μοιραζόμενα κλειδώματα. Επομένως, τελικά, ένα αποκλειστικό κλείδωμα αποτρέπει και τα write και τα read.

Το πρόβλημα που προκύπτει από τα κλειδώματα που χρησιμοποιούν τα παραδοσιακά συστήματα διαχείρισης βάσεων δεδομένων είναι ότι τα writes μπλοκάρουν πάντα τα reads. Αυτό συμβαίνει γιατί, όπως είπαμε, για τα writes χρησιμοποιείται το αποκλειστικό κλείδωμα που αποτρέπει την πρόσβαση στα κλειδωμένα δεδομένα ως προς την ανάγνωση. [2]

Αυτό το πρόβλημα επιδρά άμεσα στην επεκτασιμότητα και στις επιδόσεις μίας βάσης δεδομένων, πράγμα που καθιστά το 2Phase-locking κακή επιλογή για ένα σύστημα εξυπηρέτησης σε νέφος ή μία κατανεμημένη βάση δεδομένων. Οπότε, επιχειρούμε να επιλύσουμε το πρόβλημα χρησιμοποιώντας μία άλλη μορφή ελέγχου των ταυτόχρονων συνδιαλλαγών, τον Έλεγχο Ταυτοχρονισμού Πολλαπλών Εκδόσεων (MultiVersion Concurrency Control).

2.2.4 Η στρατηγική του Snapshot Isolation

Στον τομέα των βάσεων δεδομένων, το Snapshot Isolation (Απομόνωση Στιγμιότυπου) είναι μία στρατηγική που εγγυάται πως όλα τα reads που γίνονται σε μία συνδιαλλαγή, θα βλέπουν ένα συνεπές στιγμιότυπο της βάσης δεδομένων και ότι η ίδια η συνδιαλλαγή θα κάνει επιτυχώς commit μόνο αν δεν έχει κάνει η ίδια ενημερώσεις που να συγκρούονται με ταυτόχρονες ενημερώσεις από την εκκίνηση του στιγμιότυπου.

Υιοθετήθηκε κυρίως λόγω του ότι επιτρέπει καλύτερες επιδόσεις, λόγω της σειριοποιησιμότητας, ενώ αποφεύγει τα περισσότερα προβλήματα με τις ταυτόχρονες συνδιαλλαγές. Υλοποιείται με Έλεγχο Ταυτοχρονισμού Πολλαπλών Εκδόσεων (MVCC), όπου οι γεννήτριες τιμές κάθε αντικειμένου δεδομένων διατηρούνται.

Μία συνδιαλλαγή που λειτουργεί σε ένα σύστημα με Snapshot Isolation εμφανίζεται να λειτουργεί σε ένα ιδιωτικό στιγμιότυπο της βάσης δεδομένων, το οποίο δημιουργήθηκε στην αρχή της συνδιαλλαγής. Όταν η συνδιαλλαγή φτάσει στο τέρμα της, θα κάνει επιτυχώς commit μονάχα αν οι τιμές που ενημέρωσε δεν έχουν μεταβληθεί από εξωτερικές συνδιαλλαγές, από τη στιγμή της δημιουργίας του στιγμιότυπου και μετά. Μία τέτοια σύγκρουση write-write θα κάνει τη συνδιαλλαγή να απορριφθεί.

Παρόλα αυτά, αν η Απομόνωση Στιγμιότυπου υιοθετείται σε ένα σύστημα με MVCC, τότε επιτρέπει ώστε οι συνδιαλλαγές να προχωρούν χωρίς να ανησυχούν για ταυτόχρονες λειτουργίες και κυρίως χωρίς να απαιτείται να επαληθεύουν συνεχώς όλες τις λειτουργίες read, όταν η συνδιαλλαγή κάνει commit. Η μόνη πληροφορία που αποθηκεύεται κατά τη διάρκεια της συνδιαλλαγής είναι μία λίστα με ενημερώσεις που έγιναν, η οποία μπορεί να ανιχνεύεται για συγκρούσεις εύκολα, πριν το τελικό commit.

2.2.5 Απαιτήσεις για Snapshot Isolation: MVCC

Όπως λέει και το όνομα του, το MultiVersion Concurrency στηρίζεται σε πολλαπλές εκδόσεις των δεδομένων για την επίτευξη μεγαλύτερων επιπέδων ταυτοχρονισμού συνδιαλλαγών. Συνήθως, τα DBMS που προσφέρουν MVCC (MVDB – MultiVersion Databases) πρέπει να έχουν τα εξής χαρακτηριστικά:

a) Το DBMS θα πρέπει να μπορεί να ανακτήσει τις παλιότερες εκδόσεις μίας γραμμής.
b) Το DBMS θα πρέπει να διαθέτει ένα μηχανισμό για να καθορίζει πότε μία έκδοση μίας γραμμής είναι έγκυρη στα πλαίσια μίας συνδιαλλαγής. Συνήθως, το DBMS χρειάζεται μόνο να λάβει υπόψη του μία έκδοση που παραδόθηκε πριν την έναρξη της συνδιαλλαγής που τρέχει το ερώτημα. Για να καθοριστεί αυτό, το DBMS πρέπει να γνωρίζει ποια συνδιαλλαγή δημιούργησε μία συγκεκριμένη έκδοση μίας γραμμής και αν αυτή η συνδιαλλαγή έκανε commit πριν από την εκκίνηση της τρέχουσας συνδιαλλαγής. [2]

Για να έχουμε απομόνωση στιγμιότυπων σε ένα τέτοιο σύστημα με MVCC, ένα σημαντικό απαιτούμενο είναι η κάθε συνδιαλλαγή να χειρίζεται τις ιδιωτικές της εκδόσεις, ούτως ώστε οι συνδιαλλαγές δεν εμπλέκονται σε ιδιωτικές εκδόσεις άλλων και έτσι ισχύει ότι κάνει reads μόνο των δικών της writes. Όταν μία συνδιαλλαγή ζητά να εισάγει/ενημερώσει/διαγράψει ένα αντικείμενο δεδομένων, η αντίστοιχη λειτουργία αυτόματα προστίθεται στο προσωπικό της write-set. Τα write-sets γίνονται αντικείμενο χειρισμού στην πλευρά-πελάτη, ώστε η κάθε συνδιαλλαγή να έχει τη δική της συλλογή από write-sets που αποθηκεύονται στα πλαίσιά της. Με αυτόν τον τρόπο, μόνο ο ιδιοκτήτης ενός write-set μίας συνδιαλλαγής έχει πρόσβαση στα δεδομένα της. Σε αυτό το σημείο, τίποτα δεν αποθηκεύεται μόνιμα στον αποθηκευτικό χώρο, αλλά τα πάντα διατηρούνται στη μνήμη, από την εφαρμογή, μέχρι τη φάση του commit.

Στο ολοκληρωμένο σύστημα της MongoDB με MVCC που χρησιμοποιήσαμε, το Snapshot Isolation γίνεται μέσω της ιδιωτικής λίστας του MongoMVCC-Driver (θα περιγραφεί στο κεφάλαιο 4) και του Διαχειριστή Συνδιαλλαγών (θα περιγραφεί στο κεφάλαιο 3).

2.3 Το Υπολογιστικό Νέφος

2.3.1 Τα Χαρακτηριστικά του Υπολογιστικού Νέφους

Σύμφωνα με το NIST [12], τα βασικά χαρακτηριστικά του υπολογιστικού νέφους είναι τα παρακάτω:

- **Προσωπική εξυπηρέτηση Κατ' αίτηση (On-demand Self-service):** Ένας καταναλωτής μπορεί μονομερώς να εφοδιαστεί με υπολογιστικές δυνατότητες, όπως χρόνο εξυπηρετητή και αποθηκευτικό χώρο δικτύου, κατά τις ανάγκες του, αυτομάτως και χωρίς την ανάγκη για ανθρώπινη αλληλεπίδραση με τον κάθε πάροχο υπηρεσιών.
- **Ευρεία Πρόσβαση μέσω Δικτύου (Broad Network Access):** Οι υπολογιστικοί πόροι είναι διαθέσιμοι μέσω διαδικτύου και η πρόσβαση σ' αυτούς γίνεται μέσω καθιερωμένων μηχανισμών που προάγουν τη χρήση μέσω ετερογενών μικρών ή μεγάλων συσκευών-πελατών (όπως για παράδειγμα, κινητά τηλέφωνα, υπολογιστές tablet, φορητούς υπολογιστές και σταθμούς εργασίας).
- **Συγκέντρωση Υπολογιστικών Πόρων (Resource Pooling):** Οι υπολογιστικοί πόροι του παρόχου συγκεντρώνονται για να εξυπηρετήσουν πολλαπλούς καταναλωτές μέσω ενός μοντέλου πολλών-ενοικιαστών (multi-tenant model), με διαφορετικούς υλικούς και εικονικούς πόρους, οι οποίοι ανατίθενται δυναμικά και επανατίθενται ανάλογα με τη καταναλωτική ζήτηση. Υπάρχει ανεξαρτησία από την τοποθεσία, καθώς ο πελάτης δεν έχει γενικά έλεγχο ή γνώση για την ακριβή τοποθεσία των παρεχόμενων πόρων, αλλά μπορεί να επιλέξει την τοποθεσία σε ένα υψηλότερο επίπεδο αφαίρεσης (όπως χώρα, κρατίδιο ή κέντρο δεδομένων – datacenter). Παραδείγματα τέτοιων πόρων αποτελούν ο αποθηκευτικός χώρος, η επεξεργαστική ισχύς, η μνήμη και το εύρος ζώνης του δικτύου.
- **Ταχεία Ελαστικότητα (Rapid Elasticity):** Οι υπολογιστικοί πόροι μπορούν με ελαστικό τρόπο να δεσμευτούν και να απελευθερωθούν, σε μερικές περιπτώσεις αυτόματα, ώστε να κλιμακωθούν προς τα έξω ή προς τα μέσα ανάλογα με τη ζήτηση. Για τον καταναλωτή, οι πόροι που είναι διαθέσιμοι για δέσμευση συχνά εμφανίζονται να είναι απεριόριστοι και να μπορούν να δεσμευθούν σε οποιαδήποτε ποσότητα και κάθε στιγμή.
- **Μετρούμενη Υπηρεσία (Measured Service):** Τα συστήματα υπολογιστικών νεφών ελέγχουν αυτόματα και βελτιστοποιούν τη χρήση των πόρων διαθέτοντας ένα σύστημα μέτρησης, με πληρωμή ή χρέωση με κάθε χρήση (pay-per-use/charge-

per-use), σε κάποιο επίπεδο αφαίρεσης, ανάλογο με το είδος της υπηρεσίας (για παράδειγμα, αποθηκευτικό χώρο, επεξεργαστική ισχύ, εύρος ζώνης και ενεργούς λογαριασμούς χρήστη). Η χρήση των πόρων μπορεί να γίνει αντικείμενο παρακολούθησης, να ελεγχθεί και να αναφερθεί, παρέχοντας έτσι διαφάνεια για τον πάροχο, αλλά και για το χρήστη της υπηρεσίας.

Τα γενικά χαρακτηριστικά του υπολογιστικού νέφους, φαίνονται στον παρακάτω πίνακα:

Χαρακτηριστικά	***Περιγραφή***
Διεπαφή Προγραμματισμού Εφαρμογής (API)	Πρόσβαση σε λογισμικό που επιτρέπει στις μηχανές να αλληλεπιδρούν με λογισμικό νέφους με τον ίδιο τρόπο που η παραδοσιακή διεπαφή χρήστη επιτρέπει μεταξύ ανθρώπου και υπολογιστή. Συνήθως χρησιμοποιούνται RESTful-based APIs.
Κόστος	Το κόστος μειώνεται σύμφωνα με τους παρόχους νέφους. Ένα μοντέλο διανομής δημόσιου-νέφους (public-cloud) μετατρέπει το κεφάλαιο σε λειτουργικά έξοδα. [13]
Ευκινησία	Βελτιώνεται με τη δυνατότητα των χρηστών να επαναδεσμεύουν τους πόρους των τεχνολογικών υποδομών
Συντήρηση	Η συντήρηση γίνεται ευκολότερη, επειδή δε χρειάζεται να εγκατασταθούν στον υπολογιστή του κάθε χρήστη, ενώ μπορούν να προσπελαύνονται από διάφορες τοποθεσίες.
Απόδοση	Η απόδοση παρακολουθείται ενώ συνεπείς και χαλαρά συνδεδεμένες αρχιτεκτονικές δημιουργούνται χρησιμοποιώντας υπηρεσίες διαδικτύου ως τη διεπαφή συστήματος. [14] [15] [16]
Παραγωγικότητα	Μπορεί να αυξηθεί όταν πολλοί χρήστες μπορούν να δουλέψουν πάνω στα ίδια δεδομένα ταυτόχρονα, αντί να περιμένουν να αποθηκευτούν και να σταλούν. Μπορεί να εξοικονομηθεί χρόνος καθώς οι πληροφορίες δε χρειάζεται να επανεισαχθούν όταν τα πεδία ταιριάζουν, ούτε οι χρήστες να εγκαταστήσουν βελτιωμένες εκδόσεις εφαρμογών στον υπολογιστή τους. [17]

Αξιοπιστία	Αυξάνεται μέσω της χρήσης πολλών πλεοναζουσών σελίδων, οι οποίες καθιστούν ένα καλά σχεδιασμένο υπολογιστικό νέφος κατάλληλο για εργασιακή συνέχεια και επαναφορά μετά από καταστροφές. [18]
Ασφάλεια	Είναι δυνατόν να βελτιωθεί λόγω της συγκέντρωσης των δεδομένων, αυξημένων πόρων επικεντρωμένων στην ασφάλεια κλπ, αλλά υπάρχουν ακόμα φόβοι για την απώλεια ελέγχου ευαίσθητων δεδομένων και την έλλειψη ασφαλείας για αποθηκευμένους πυρήνες. [19]

2.3.2 Μοντέλα του Υπολογιστικού Νέφους

Υπάρχουν τέσσερα είδη για τη δημιουργία και την ανάπτυξη του υπολογιστικού νέφους:

- **Ιδιωτικό Νέφος (Private Cloud)**
  Η υποδομή του νέφους δεσμεύεται για αποκλειστική χρήση από ένα μοναδικό οργανισμό, ο οποίος περιλαμβάνει πολλούς καταναλωτές (για παράδειγμα εταιρικές μονάδες). Μπορεί να ανήκει, να γίνεται αντικείμενο διαχείρισης και να λειτουργεί υπό τον ίδιο τον οργανισμό, έναν τρίτο ή ένα συνδυασμό αυτών και μπορεί να βρίσκεται εντός ή εκτός των εγκαταστάσεών του. [12] Η ανάληψη ενός έργου ιδιωτικού νέφους απαιτεί ένα σημαντικό επίπεδο δέσμευσης για την εικονικοποίηση του εταιρικού περιβάλλοντος και απαιτεί την επανεκτίμηση των πόρων από τον οργανισμό. Όταν αυτά γίνουν σωστά, μπορεί να βελτιώσει την εργασία, αλλά το κάθε βήμα του έργου παρουσιάζει θέματα ασφαλείας, τα οποία πρέπει να αντιμετωπισθούν, ώστε να αποφευχθούν σοβαρά τρωτά σημεία. [20] Τα κέντρα δεδομένων τα οποία τρέχουν αυτόματα (self-run datacenters) επιβαρύνουν αρκετά την κεφαλαιακή επένδυση, καθώς έχουν αρκετές υλικές απαιτήσεις, εφόσον χρειάζονται δέσμευση αποθηκευτικού χώρου, υλικό και έλεγχο περιβάλλοντος.
- **Κοινοτικό Νέφος (Community Cloud)**
  Η υποδομή του νέφους δεσμεύεται για αποκλειστική χρήση από τους καταναλωτές μίας συγκεκριμένης κοινότητας, οι οποίοι προέρχονται από οργανισμούς που μοιράζονται κοινούς σκοπούς (για παράδειγμα αποστολή και απαιτήσεις ασφαλείας) . Μπορεί να ανήκει , να γίνεται αντικείμενο διαχείρισης και να λειτουργεί υπό έναν ή περισσότερους οργανισμούς της κοινότητας, τρίτους ή συνδυασμό τους και να βρίσκεται εντός ή εκτός των εγκαταστάσεων. [12]

- **Δημόσιο Νέφος (Public Cloud)**
  Η υποδομή του νέφους δεσμεύεται για ανοιχτή δημόσια χρήση. Μπορεί να ανήκει, να το διαχειρίζονται, και να λειτουργεί υπό μία εταιρεία, ακαδημαϊκό ή κυβερνητικό οργανισμό ή κάποιον συνδυασμό αυτών. Βρίσκεται εντός των εγκαταστάσεων. [12] Τεχνικά, μπορεί να υπάρχει μικρή διαφορά μεταξύ της αρχιτεκτονικής δημόσιου και ιδιωτικού νέφους, αλλά η ασφάλεια μπορεί να είναι ουσιαστικά διαφορετική για τις υπηρεσίες που γίνονται διαθέσιμες από έναν πάροχο υπηρεσιών για ένα δημόσιο κοινό και όταν η επικοινωνία γίνεται μέσω ενός δικτύου μη-έμπιστου.
- **Υβριδικό Νέφος (Hybrid Cloud)**
  Η υποδομή του νέφους είναι ένας συνδυασμός δύο ή περισσότερων διακριτών υποδομών νεφών (ιδιωτικό, κοινοτικό ή δημόσιο), οι οποίες παραμένουν μεμονωμένες οντότητες, αλλά δεσμεύονται η μία με την άλλη μέσω τυποποιημένης τεχνολογίας που επιτρέπει στα δεδομένα και τις εφαρμογές να είναι φορητές (για παράδειγμα, ριπή νέφους για την εξισορρόπηση φορτίου μεταξύ νεφών). [12]

2.3.3 Μοντέλα Υπηρεσιών του Υπολογιστικού Νέφους

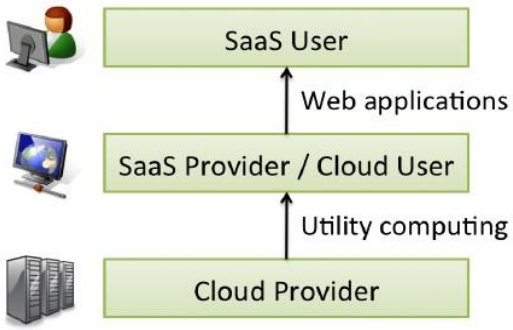
2.3.3.1 Λογισμικό-ως-Υπηρεσία (Software -as- a- Service - SaaS)

Παρέχεται στο χρήστη η δυνατότητα να χρησιμοποιήσει τις εφαρμογές του παρόχου, οι οποίες τρέχουν σε υποδομή νέφους.[6] Οι εφαρμογές είναι προσβάσιμες από διάφορες συσκευές-πελάτες (client devices), μέσω είτε μίας λεπτής διεπαφής πελάτη, όπως ένας περιηγητής διαδικτύου (για παράδειγμα υπηρεσία ηλεκτρονικού ταχυδρομείου που βασίζεται στο διαδίκτυο – web-based e-mail), είτε μία διεπαφή προγράμματος. Ο χρήστης δεν διαχειρίζεται, ούτε ελέγχει την υποκείμενη υποδομή του νέφους, στην οποία περιλαμβάνονται τα δίκτυα, οι εξυπηρετητές, τα λειτουργικά συστήματα, αποθηκευτικός χώρος ή οι πόροι της κάθε διεργασίας, με την πιθανή εξαίρεση περιορισμένων ρυθμίσεων, αναφερόμενων στο χρήστη, των εφαρμογών . [12]

Το Λογισμικό-ως-Υπηρεσία είναι μία πλατφόρμα πολλών-ενοικιαστών (multi-tenant platform). Χρησιμοποιεί κοινούς πόρους και ένα μοναδικό στιγμιότυπο τόσο του τελικού κώδικα μίας εφαρμογής καθώς και της υποκείμενης βάσης δεδομένων για να υποστηρίξει ταυτόχρονα πολλούς πελάτες. Το Λογισμικό-ως-Υπηρεσία, το οποίο συχνά καλείται και μοντέλο του Παρόχου Υπηρεσιών

[6] Υποδομή νέφους είναι η συλλογή υλικού και λογισμικού που επιτρέπει τα 5 βασικά χαρακτηριστικά του υπολογιστικού νέφους. Η υποδομή νέφους μπορεί να θεωρηθεί ότι περιέχει ένα φυσικό στρώμα, καθώς και ένα στρώμα αφαίρεσης. Το φυσικό στρώμα αποτελείται από πόρους υλικού, οι οποίοι είναι απαραίτητοι για να υποστηρίξουν τις υπηρεσίες νέφους που παρέχονται, και συνήθως περιλαμβάνει εξυπηρετητές, αποθηκευτικό χώρο και δικτυακό μέρος. Το επίπεδο αφαίρεσης αποτελείται από λογισμικό, ανεπτυγμένο κατά μήκος του φυσικού στρώματος, το οποίο καθορίζει τα βασικά χαρακτηριστικά του νέφους. Ως έννοια, το στρώμα αφαίρεσης βρίσκεται πάνω από το φυσικό.

Εφαρμογής (Application Service Provider – ASP), θεωρείται από πολλούς ως η νέα τάση στη διανομή εφαρμογών λογισμικού. [21]



Εικόνα 13: Λογισμικό-ως-Υπηρεσία

Παραδείγματα μοντέλων Λογισμικού-ως-Υπηρεσία του εμπορίου είναι τα Google Apps, Facebook και το Youtube.

- **Google Apps**
  Το Google Apps είναι μία υπηρεσία της Google, η οποία παρέχει ανεξάρτητα διαμορφώσιμες εκδόσεις διαφόρων προϊόντων της εταιρείας, χρησιμοποιώντας όνομα διαδικτυακής διεύθυνσης (domain name) το οποίο παρέχεται από το χρήστη. Προσφέρει διάφορες διαδικτυακές εφαρμογές με λειτουργικότητα παρόμοια με τις παραδοσιακές σουίτες γραφείου (office suites), συμπεριλαμβανουμένου των Gmail, Hangouts, Google Calendar, Drive, Docs, Sheets, Slides, Groups, News, Play, Sites και Vault.
  Το Google Apps διατίθεται δωρεάν για 30 μέρες, ενώ μετά απ' αυτές τις μέρες κοστίζει 5$ ανά λογαριασμό χρήστη για κάθε μήνα χρήσης ή 50$ για κάθε χρόνο. Όταν η χρήση του γίνεται για εκπαιδευτικούς ή μη-κερδοσκοπικούς λόγους, είναι δωρεάν, ενώ παρέχεται ο ίδιος αποθηκευτικός χώρος. [22]
- **Facebook**
  Το Facebook είναι μία υπηρεσία κοινωνικής δικτύωσης (social networking service). Ο χρήστης, αφού κάνει εγγραφή στη σελίδα, μπορεί να δημιουργήσει ένα προσωπικό προφίλ, να προσθέσει άλλους χρήστες ως φίλους του, να ανταλλάξει μηνύματα, να κάνει γνωστές αλλαγές στην κατάστασή του, να ανεβάσει φωτογραφίες του και να δεχθεί ειδοποιήσεις όταν οι άλλοι χρήστες ενημερώνουν το προφίλ τους. Επίσης, οι χρήστες μπορούν να γίνουν μέλη σε ομάδες κοινών-ενδιαφερόντων, οργανωμένα από το χώρο εργασίας, το σχολείο ή το κολλέγιο, ή άλλα χαρακτηριστικά και να κατηγοριοποιήσουν τους φίλους τους σε λίστες όπως «Άνθρωποι από τη δουλειά» ή «Στενοί φίλοι». Το Facebook είχε 1.3 δισεκατομμύρια ενεργούς χρήστες, τον Ιούνιο του 2014. [23]
- **Youtube**
  Το Youtube είναι μία ιστοσελίδα διαμοιρασμού βίντεο (video-sharing

website). Η σελίδα του επιτρέπει στους χρήστες να ανεβάσουν, να δουν και να μοιραστούν βίντεο. Χρησιμοποιεί τις τεχνολογίες Adobe Flash Video και HTML 5, για να εμφανίσει μία μεγάλη γκάμα από βίντεο, τα οποία έχουν δημιουργήσει οι χρήστες και τα μέσα μαζικής ενημέρωσης των εταιριών.
Οι μη εγγεγραμμένοι χρήστες μπορούν να δουν βίντεο, και οι εγγεγραμμένοι να ανεβάσουν βίντεο στα κανάλια του. Τα βίντεο, τα οποία μπορεί να περιέχουν προσβλητικό περιεχόμενο είναι διαθέσιμα μόνο σε εγγεγραμμένους χρήστες, οι οποίοι έχουν δηλώσει πως είναι άνω των 18 ετών.

2.3.3.2 Πλατφόρμα-ως-Υπηρεσία (Platform –as- a- Service – PaaS)

Παρέχεται στο χρήστη η δυνατότητα ανάπτυξης εφαρμογών, πάνω στο νέφος, τις οποίες έχει δημιουργήσει ή αποκτήσει ο χρήστης. Οι εφαρμογές αυτές έχουν δημιουργηθεί χρησιμοποιώντας προγραμματιστικές γλώσσες, βιβλιοθήκες, υπηρεσίες και εργαλεία, τα οποία υποστηρίζονται από τον πάροχο.[7] Ο καταναλωτής δεν διαχειρίζεται , ούτε ελέγχει την υποκείμενη υποδομή νέφους, περιλαμβανομένου του δικτύου, των εξυπηρετητών, των λειτουργικών συστημάτων ή του αποθηκευτικού χώρου, αλλά έχει τον έλεγχο των αναπτυσσόμενων εφαρμογών και πιθανών ρυθμίσεων για το περιβάλλον, το οποίο φιλοξενεί τις εφαρμογές. [12]

Η κεντρική ιδέα της Πλατφόρμας-ως-Υπηρεσία είναι να παρέχεται στους προγραμματιστές μία πλατφόρμα, η οποία περιλαμβάνει όλα τα συστήματα και τα περιβάλλοντα, που παρέχουν τον πλήρη κύκλο πέρας-προς-πέρας, του προγραμματισμού, δοκιμής, ανάπτυξης και φιλοξενίας εξελιγμένων εφαρμογών διαδικτύου, ως μία υπηρεσία που διανέμεται βασιζόμενη στο νέφος. [21]

Παραδείγματα Πλατφόρμας-ως-Υπηρεσία του εμπορίου είναι το Google Apps Engine, το Microsoft Azure, το GigaSpaces και το SunCloud.

2.3.3.3 Υποδομή-ως-Υπηρεσία (Infrastructure –as- a- Service – IaaS)

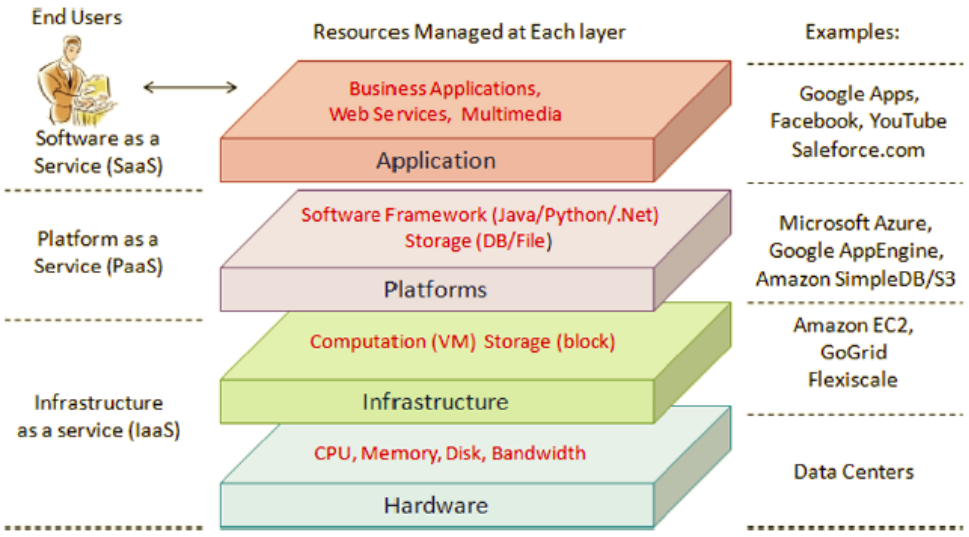
Παρέχεται στο χρήστη η δυνατότητα να δεσμεύσει επεξεργαστική ισχύ, αποθηκευτικό χώρο, δίκτυα και άλλους θεμελιώδεις υπολογιστικούς πόρους, όπου ο χρήστης μπορεί να αναπτύξει και να τρέξει αυθαίρετα λογισμικό, το οποίο περιλαμβάνει λειτουργικά συστήματα και εφαρμογές. Ο χρήστης δεν διαχειρίζεται, ούτε ελέγχει την υποκείμενη υποδομή νέφους, αλλά έχει τον έλεγχο των λειτουργικών συστημάτων, του αποθηκευτικού χώρου και των

[7] Αυτή η δυνατότητα δεν αποκλείει απαραίτητα τη χρήση συμβατών προγραμματιστικών γλωσσών, βιβλιοθηκών, υπηρεσιών και εργαλείων από άλλες πηγές.

αναπτυσσόμενων εφαρμογών και πιθανόν περιορισμένο έλεγχο πάνω σε επιλεγμένα μέρη του δικτύου (όπως το τείχος προστασίας στο υλικό που φιλοξενεί το νέφος - host firewall). [12]

Γενικά, πρόκειται για τη διανομή υποδομής υπολογιστών ως μία υπηρεσία. Εκτός της υψηλότερης ελαστικότητας, ένα πρωτεύον όφελος της Υποδομής-ως-Υπηρεσία είναι το σχήμα πληρωμής βασιζόμενο στη χρήση (usage-based payment scheme). Αυτό επιτρέπει στους πελάτες να πληρώνουν καθώς αυξάνεται η χρήση τους. Ένα άλλο μεγάλο πλεονέκτημα είναι ότι χρησιμοποιείται πάντα η τελευταία λέξη της τεχνολογίας. Οι χρήστες μπορούν να επιτύχουν μία πολύ γρηγορότερη διανομή υπηρεσίας. [21]

Παραδείγματα Υποδομής-ως-Υπηρεσία του εμπορίου είναι το Amazon Web Service, το GoGrid, το Flexiscale και το Mosso.



Εικόνα 14: Τα μοντέλα των υπηρεσιών του Υπολογιστικού Νέφους

2.3.4 Τα Πλεονεκτήματα του Υπολογιστικού Νέφους

Τα γενικά πλεονεκτήματα του υπολογιστικού νέφους είναι αρκετά για να δικαιολογήσουν τη μετάβαση μίας εταιρείας σε μοντέλο νέφους. Μέσω αυτής της μετάβασης επιτυγχάνονται [24]:

i. **Μεγάλο Κέρδος:** Παρότι η χρέωση φαίνεται πολύ χαμηλή, τα μεγάλα κέντρα δεδομένων (δεκάδες χιλιάδες υπολογιστές) μπορούν να αγοράσουν υλικό, δικτυακό εύρος ζώνης και ισχύ σε τιμές που φτάνουν το 1/5 και 1/7 των αντίστοιχων μετρίου-μεγέθους κέντρων δεδομένων (εκατοντάδες ή χιλιάδες υπολογιστές). Επιπλέον, τα σταθερά κόστη της ανάπτυξης λογισμικού μπορούν να αποσβεσθούν πάνω σε πολλά μηχανήματα. Άλλοι υποστηρίζουν ότι μία αρκετά μεγάλη εταιρεία μπορεί

να εκμεταλλευτεί τις οικονομίες κλίμακας, ώστε να προσφέρει μία υπηρεσία με κόστος αρκετά μικρότερο μίας μεσαίου-μεγέθους επιχείρησης και παρόλα αυτά να έχει ένα ασφαλές κέρδος.

ii. **Εκμετάλλευση της υπάρχουσας επένδυσης:** Η προσθήκη υπηρεσιών υπολογιστικού νέφους στην υπάρχουσα υποδομή παρέχει μία νέα πηγή κέρδους σε ιδανικά χαμηλό κόστος, βοηθώντας έτσι στην απόσβεση μεγάλων επενδύσεων σε κέντρα δεδομένων. Σύμφωνα με τον Werner Vogels, διευθυντή οικονομικών της Amazon, πολλές τεχνολογίες του Amazon Web Services αναπτύχθηκαν αρχικά για εσωτερικές λειτουργίες της Amazon.

iii. **Προστασία υπάρχοντος προϊόντος:** Καθώς οι συμβατικοί εξυπηρετητές και οι επιχειρησιακές εφαρμογές προχωρούν σε μοντέλα υπολογιστικού νέφους, πωλητές που προσφέρουν τέτοιες εφαρμογές έχουν κίνητρο να παρέχουν μία δική τους επιλογή για νέφος. Για παράδειγμα, το Microsoft Azure, προσφέρει μία άμεση επιλογή για τη μετακίνηση των υπαρχόντων πελατών των επιχειρησιακών εφαρμογών της Microsoft σε περιβάλλον νέφους.

iv. **Καταπολέμηση ενός κατεστημένου:** Μία εταιρεία με το απαιτούμενο κέντρο δεδομένων και υπολογιστικούς πόρους μπορεί να θέλει να αποκτήσει προβάδισμα σ' αυτόν τον τομέα, προτού μία άλλη μεγάλη εταιρεία εισέλθει στο χώρο. Η Google App Engine προσφέρει ένα εναλλακτικό δρόμο στην ανάπτυξη νέφους, η οποία έχει το πλεονέκτημα των πολλών χαρακτηριστικών επεκτασιμότητας και εξισορρόπησης φορτίου που διαφορετικά θα έπρεπε να δημιουργήσουν μόνοι τους.

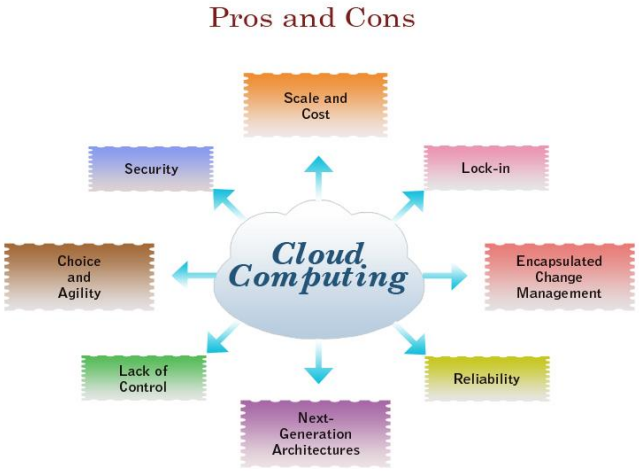
v. **Διατήρηση των σχέσεων με τους πελάτες:** Πολλοί οργανισμοί παροχής υπηρεσιών πληροφορικής, όπως η IBM Global Services, έχουν εκτενείς σχέσεις με τους πελάτες τους μέσω των προσφορών υπηρεσιών τους. Παρέχοντας μία προσφορά υπολογιστικού νέφους, δίνει σε αυτούς τους πελάτες ένα δρόμο μετακίνησης στο νέφος, χωρίς άγχος, που διατηρεί την επένδυση των δύο πλευρών σε αυτή τη σχέση.

vi. **Δημιουργία πλατφόρμας:** Η πρωτοπορία του Facebook να επιτρέψει εφαρμογές plug-in ταιριάζει απόλυτα στο υπολογιστικό νέφος, και πράγματι ένας πάροχος υποδομής για plug-in εφαρμογές του Facebook είναι η Joyent, ένας πάροχος νέφους. Το κίνητρο του Facebook ήταν να καταστήσει την εφαρμογή κοινωνικής δικτύωσής του ως μία νέα πλατφόρμα ανάπτυξης.

Επίσης, το υπολογιστικό νέφος, παρότι υπήρχε ως ιδέα εδώ και πολλά χρόνια, ωρίμασε τα τελευταία, λόγω των ιδανικών συνθηκών που δημιουργήθηκαν. Έγινε εισαγωγή σε νέες τεχνολογικές τάσεις και επιχειρησιακά μοντέλα. Ένα τέτοιο παράδειγμα είναι η εμφάνιση του Web 2.0, το οποίο ήταν στροφή από ένα μοντέλο υψηλών απαιτήσεων και δεσμεύσεων σε μία αυτό-εξυπηρέτηση με μικρές απαιτήσεις και δεσμεύσεις.

Ταυτόχρονα, δημιουργήθηκαν νέες ευκαιρίες για την ανάπτυξη εφαρμογών, οι οποίες ταιριάζουν περισσότερο στο υπολογιστικό νέφος, όπως:

- Εφαρμογές διαδραστικότητας για φορητές συσκευές
- Επεξεργασία παράλληλων δεσμίδων
- Αύξηση της ανάλυσης των δεδομένων και των στατιστικών
- Επέκταση και βελτίωση των εφαρμογών για επιτραπέζιους υπολογιστές, οι οποίες είναι αρκετά απαιτητικές



Εικόνα 15: Τα πλεονεκτήματα και τα μειονεκτήματα του Υπολογιστικού Νέφους

2.4 Η NoSQL Βάση Δεδομένων MongoDB

Η MongoDB είναι μία βάση δεδομένων εγγράφων που παρέχει μεγάλες επιδόσεις, διαθεσιμότητα και εύκολη επεκτασιμότητα.

Παρότι η MongoDB υποστηρίζει λειτουργίες μοναδικού-στιγμιότυπου (single-instance operations), η εγκατάστασή της είναι συνήθως κατανεμημένη. Τα σύνολα αντιγράφων (replica sets) παρέχουν υψηλών επιδόσεων αντιγραφή με αυτόματο μηχανισμό διόρθωσης σε περίπτωση σφάλματος, ενώ τα διαμοιρασμένα συμπλέγματα καθιστούν δυνατό τον καταμερισμό σε μεγάλα σύνολα δεδομένων πάνω σε πολλά μηχανήματα. Οι χρήστες της MongoDB συνδυάζουν αυτές τις δύο τεχνολογίες για να πετύχουν το βέλτιστο αποτέλεσμα για τις εφαρμογές τους. [25]

2.4.1 Η δομή και η λειτουργία της MongoDB

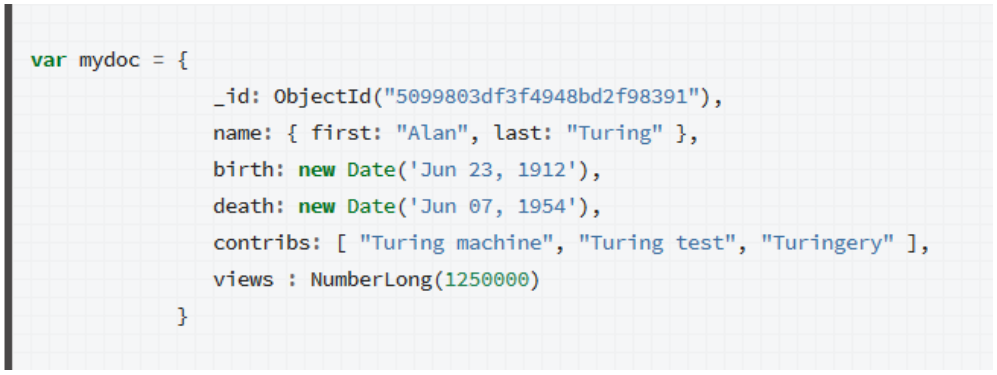
Η εγκατάσταση της MongoDB φιλοξενεί έναν αριθμό βάσεων δεδομένων. Κάθε βάση δεδομένων από αυτές περιέχει ένα σύνολο από συλλογές (collections). Κάθε συλλογή περιέχει ένα σύνολο από έγγραφα. Τέλος, το κάθε έγγραφο είναι ένα σύνολο από ζεύγη κλειδιού-τιμής. Τα έγγραφα έχουν δυναμικό σχήμα (dynamic schema). Αυτό σημαίνει ότι τα έγγραφα της ίδιας συλλογής δεν

χρειάζεται να έχουν το ίδιο σύνολο πεδίων ή δομής και τα κοινά πεδία μίας συλλογής εγγράφων μπορούν να περιέχουν διαφορετικούς τύπους δεδομένων.

Εικόνα 16: Ένα έγγραφο JSON με τη μορφή που τα αποθηκεύει η MongoDB

Η MongoDB αποθηκεύει τα έγγραφα στο δίσκο σε φορμά σειριοποίησης BSON. Το BSON είναι μία δυαδική αναπαράσταση των JSON εγγράφων, αλλά περιέχει πολλούς περισσότερους τύπους δεδομένων από το JSON. Το κέλυφος JavaScript *mongo* και οι οδηγοί γλώσσας MongoDB (MongoDB language drivers) κάνουν τη μετάφραση μεταξύ BSON και της αναπαράστασης των εγγράφων με βάση τη γλώσσα προγραμματισμού. Στη δική μας περίπτωση η γλώσσα προγραμματισμού είναι η Java.

Η τιμή ενός πεδίου μπορεί να είναι οποιουδήποτε τύπου δεδομένων BSON, συμπεριλαμβανουμένων άλλων εγγράφων, πινάκων και πινάκων από έγγραφα. Για παράδειγμα το έγγραφο που φαίνεται στην εικόνα περιέχει τιμές από ποικίλους τύπους.

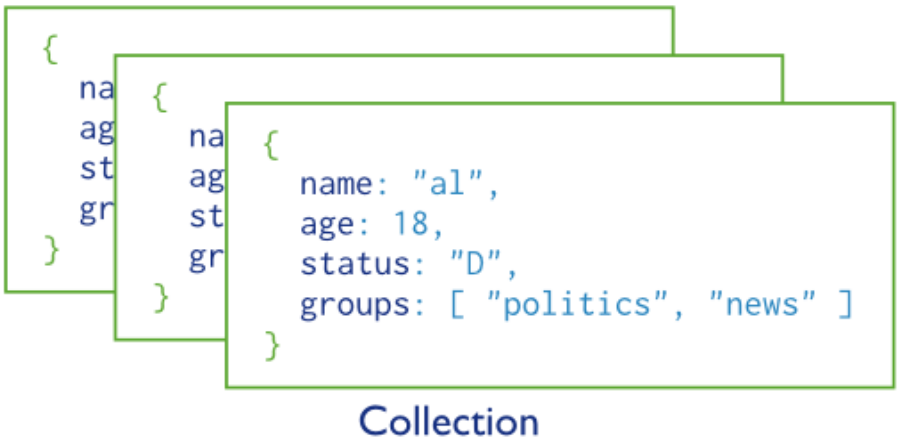


Εικόνα 17: Ένα πιο περίπλοκο έγγραφο της MongoDB

Εδώ βλέπουμε πως:

- Το **_id** περιέχει ObjectId
- Το **name** περιέχει ένα ενσωματωμένο έγγραφο που περιέχει τα πεδία **first** και **last**
- Τα πεδία **birth** και **death** περιέχουν τιμές τύπου Date
- Το **contribs** περιέχει έναν πίνακα από συμβολοσειρές (array of strings)
- Το **views** περιέχει μία τιμή τύπου NumberLong

Όσον αφορά στο πεδίο **_id**, εξ' ορισμού η MongoDB δημιουργεί ένα μοναδικό ευρετήριο γι' αυτό κατά τη δημιουργία της συλλογής. Είναι πάντα το πρώτο πεδίο στα έγγραφα και μπορεί να περιέχει κάθε τύπο BSON πλην των πινάκων.



Εικόνα 18: Συλλογή εγγράφων της MongoDB

Ο χρήστης MongoDB χρησιμοποιεί την τελεία για να έχει πρόσβαση στα στοιχεία ενός πίνακα και στα πεδία ενός ενσωματωμένου εγγράφου. Δηλαδή σε γενικές γραμμές θα πρέπει να χρησιμοποιήσει κάτι τέτοιο:

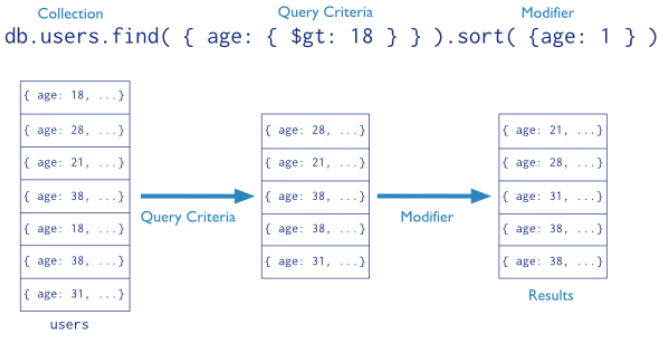
`'<array>.<index>' ή '<embedded document>.<field>'`

για πίνακα και για ενσωματωμένο έγγραφο, αντίστοιχα.

Για τα ερωτήματα (queries) η MongoDB παρέχει ένα σύνολο από τελεστές για να καθορίσουν πως η μέθοδος find() επιλέγει τα δεδομένα από μία συλλογή με βάση ένα έγγραφο προσδιορισμού ερωτημάτων που χρησιμοποιεί ένα συνδυασμό ακριβών ταιριασμάτων ισότητας και συνθηκών, με τη χρήση ενός τελεστή ερωτημάτων.

Ένα ερώτημα στοχεύει σε μία συγκεκριμένη συλλογή εγγράφων. Τα ερωτήματα καθορίζουν τα κριτήρια ή τις συνθήκες, τα οποία ή οι οποίες ορίζουν τα έγγραφα που η MongoDB θα επιστρέψει στο χρήστη. Μπορούμε να μεταβάλλουμε προαιρετικά τα ερωτήματα, ούτως ώστε να επιβάλλουμε όρια, παραβλέψεις και σειρά ταξινόμησης.

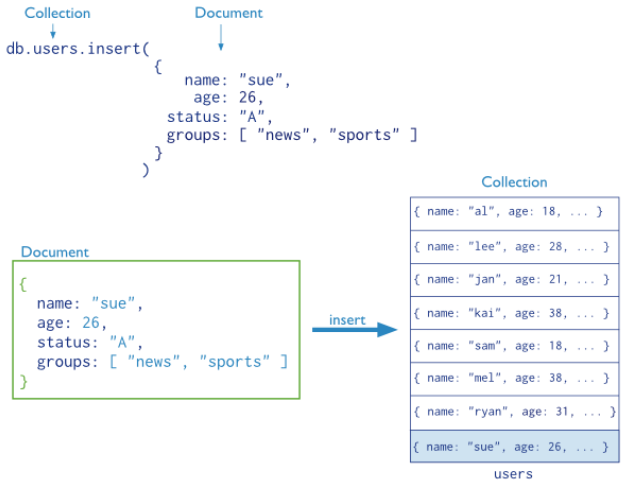
Ένα παράδειγμα τέτοιου ερωτήματος φαίνεται στην εικόνα που ακολουθεί



Εικόνα 19: Ερώτημα για την εύρεση των ατόμων με ηλικία πάνω από 18 και ταξινόμηση με βάση την ηλικία

Εκτός της ανάγνωσης των αποτελεσμάτων (read - μέσω της find() και των ερωτημάτων), μπορούμε και να τροποποιήσουμε τα δεδομένα, κάνοντας μία λειτουργία CRUD (Create, Update, Delete – Δημιουργία, Ενημέρωση, Διαγραφή δεδομένων). Για την ενημέρωση και τη διαγραφή τα κριτήρια της επιλογής των εγγράφων μπορούν να καθοριστούν. [25]

Παράδειγμα της εισαγωγής δεδομένων σε μία συλλογή φαίνεται παρακάτω:



Εικόνα 20: Εισαγωγή του εγγράφου που φαίνεται πάνω, στη συλλογή *users*

2.4.2 Χαρακτηριστικά και πλεονεκτήματα της MongoDB

Η MongoDB ως βάση δεδομένων NoSQL φέρει πρακτικά όλα τα πλεονεκτήματα αυτού του είδους βάσεων, και πιο συγκεκριμένα τα πλεονεκτήματα των βάσεων με έγγραφα. Κάποια κύρια χαρακτηριστικά και δυνατότητές της είναι:

- **Ερωτήματα ad hoc**
  Η MongoDB υποστηρίζει την αναζήτηση πεδίου, ερωτήματα εύρους (range queries), αλλά και αναζητήσεις κανονικών εκφράσεων (regular expressions). Έχουν τη δυνατότητα να επιστρέψουν συγκεκριμένα πεδία και να περιέχουν και λειτουργίες JavaScript που ορίστηκαν από το χρήστη.
- **Ευρετήρια**
  Οποιοδήποτε πεδίο σε ένα έγγραφο MongoDB μπορεί να δεικτοδοτηθεί. Υπάρχει επίσης και δυνατότητα για δευτερεύοντα ευρετήρια.
- **Αντίγραφα**
  Η MongoDB παρέχει μεγάλη διαθεσιμότητα μέσω των συνόλων αντιγράφων της. [26]
- **Εξισορρόπηση Φορτίου**
  Η MongoDB υποστηρίζει οριζόντια επεκτασιμότητα (horizontal scalability), χρησιμοποιώντας θρυμματισμό (sharding). [27] Ο χρήστης

επιλέγει ένα κλειδί θρύμματος (shard key) που καθορίζει τον τρόπο με τον οποίο τα δεδομένα θα κατανεμηθούν μέσα σε μία συλλογή. Τα δεδομένα χωρίζονται σε εύρη (βασιζόμενα σε αυτό το κλειδί) και κατανέμονται κατά μήκος πολλών θρυμμάτων. Η MongoDB μπορεί να τρέξει σε πολλούς εξυπηρετητές, εξισορροπώντας το φορτίο και δημιουργώντας αντίγραφα των δεδομένων για τη διατήρηση του συστήματος σε ενεργή κατάσταση σε περίπτωση βλάβης.

- **Αποθήκευση Αρχείων**
  Η MongoDB μπορεί να χρησιμοποιηθεί και ως ένα σύστημα αρχείων, εκμεταλλευόμενη την εξισορρόπηση φορτίου και τα αντίγραφα δεδομένων της.
- **Συσσωμάτωση**
  Η MapReduce μπορεί να χρησιμοποιηθεί για την επεξεργασίας παρτίδων των δεδομένων και λειτουργίες συσσωμάτωσης.
- **Εκτέλεση της πλευράς-εξυπηρετητή JavaScript**
  Η JavaScript μπορεί να χρησιμοποιηθεί για ερωτήματα, λειτουργίες άθροισης και να αποσταλεί απευθείας στη βάση δεδομένων για να εκτελεστεί.
- **Συλλογές με ανώτατο όριο**
  Η MongoDB υποστηρίζει και συλλογές σταθερού μεγέθους (fixed-size collections). Αυτού του είδους οι συλλογές λειτουργούν ως μία κυκλική ουρά.

2.4.3 Περιπτώσεις χρήσης της MongoDB

Η MongoDB, ως η πιο δημοφιλής NoSQL βάση δεδομένων, βρίσκει καθημερινά πολλές περιπτώσεις χρήσης στις σύγχρονες εφαρμογές. Ιδανικά χρησιμοποιείται για μοντέρνες διαδικτυακές εφαρμογές και οι καλύτερες περιπτώσεις για να την προτιμήσει κανείς είναι:

Περίπτωση Χρήσης MongoDB	***Λόγος Προτίμησης***
Προφίλ χρήστη και λογαριασμού	Μπορεί να αποθηκεύσει πίνακες διευθύνσεων με μεγάλη ευκολία
CMS (Σύστημα Διαχείρισης Περιεχομένου)	Το ευέλικτο σχήμα της είναι ιδανικό για ετερογενείς συλλογές από τύπους περιεχομένων
Δεδομένα φόρμας	Καθιστά εύκολη την εξέλιξη της δομής των δεδομένων φόρμας με την πάροδο του χρόνου
Blogs και περιεχόμενο παραγόμενο από χρήστες	Μπορεί να διατηρήσει τα δεδομένα με πολύπλοκες σχέσεις μαζί σε ένα αντικείμενο
Μηνύματα (Messaging)	Αποθήκευση των μετα-δεδομένων εύκολα ανά μήνυμα ή τύπο μηνύματος, χωρίς την ανάγκη διατήρησης ξεχωριστών συλλογών ή σχημάτων
Προτιμήσεις συστήματος	Δυνατό με έναν απλό γράφο αντικειμένων αποτελούμενο από τιμές προτιμήσεων
Δεδομένα καταλόγου οποιουδήποτε τύπου	Τα δομημένα δεδομένα καταλόγου είναι σημαντικά για το μέλλον
Γράφοι	Απαιτούνται μόνο αντικείμενα και δείκτες, οπότε η MongoDB ταιριάζει απόλυτα
Δεδομένα βασισμένα στην τοποθεσία	Η MongoDB αντιλαμβάνεται τις γεω-χωρικές συντεταγμένες

[28]

3

Ανάλυση Απαιτήσεων Συστήματος

Το ολοκληρωμένο σύστημα της MongoDB της υλοποίησής μας θα χρησιμοποιεί για την εγγύηση των ιδιοτήτων ACID των συνδιαλλαγών έναν εξωτερικό Διαχειριστή Συνδιαλλαγών, συγκεκριμένα τον ολιστικό Διαχειριστή Συνδιαλλαγών του CoherentPaaS [3]. Παρέχει API στις αποθήκες δεδομένων ώστε να χρησιμοποιείται σε συνδυασμό με τη διαδικασία ταυτόχρονων συνδιαλλαγών του CoherentPaaS [1].

3.1 Υλοποίηση των ιδιοτήτων των συνδιαλλαγών

Οι ιδιότητες των συνδιαλλαγών επιτυγχάνονται μέσα από ένα συνδυασμό διαφορετικών πρωτοκόλλων. Η ατομικότητα παρέχει την απόρριψη όλων των ενεργειών και αποτελεσμάτων μίας αποτυχημένης συνδιαλλαγής (undo). Η ανθεκτικότητα επιτρέπει την επαναφορά των ενεργειών και αποτελεσμάτων μίας επιτυχημένης συνδιαλλαγής (redo). Η ατομικότητα και η ανθεκτικότητα μαζί απαιτούν αφαιρετικότητα στο επίπεδο των δεδομένων, συνήθως στη μορφή ενός καταλόγου, δηλαδή υλοποιούνται πάνω σε ένα μηχανισμό καταλόγου. Ο κατάλογος (log) πρέπει να υλοποιηθεί με ένα πρωτόκολλο επαναφοράς που εκτελείται σε περίπτωση αποτυχίας, όταν απαιτείται επαναφορά. Το πρωτόκολλο αυτό εκτελείται πριν η βάση δεδομένων γίνει διαθέσιμη μετά από μία αποτυχία και ευθύνεται για την επαναφορά της συνεκτικότητας της βάσης δεδομένων, απορρίπτοντας ενημερώσεις που έχουν γίνει από αποτυχημένες συνδιαλλαγές και φέρνοντας πίσω ενημερώσεις από συνδιαλλαγές που έχουν κάνει commit.

Η ατομικότητα απαιτεί υπονοούμενο έλεγχο ταυτοχρονισμού. Το υψηλότερο επίπεδο απομόνωσης είναι γνωστό ως «σειριοποιησιμότητα – serializability». Αυτή εγγυάται ότι η ταυτόχρονη εκτέλεση των συνδιαλλαγών ταυτίζεται με μία σειριακή εκτέλεσή τους. Επομένως, το αποτέλεσμα που προκύπτει είναι το ισοδύναμο από το να μην υπάρχουν ταυτόχρονες συνδιαλλαγές κ να εκτελούνται σειριακά η μία μετά την άλλη. Αυτό είναι σαν όλα τα read και write μίας συνδιαλλαγής να γίνονταν σε μία μόνο χρονική στιγμή.

Παρόλα αυτά, είναι προφανές ότι η σειριοποιησιμότητα επιδρά δραματικά στη δυνατότητα ταυτόχρονων συνδιαλλαγών, καθώς υπάρχουν συγκρούσεις μεταξύ read κατηγορημάτων (για παράδειγμα ένα ερώτημα SELECT) και writes. Βασικά, ένα τέτοιο read συγκρούεται με οποιοδήποτε write του ίδιου πίνακα, εκτός αν το κατηγόρημα αποτελείται μόνο από στήλες που ανήκουν σε ευρετήριο και το λόγο κατηγορήματος. Έτσι, έχουν προταθεί άλλα επίπεδα απομόνωσης όπως τα ANSI και το Snapshot Isolation. Τα άλλα επίπεδα ANSI μειώνουν υπερβολικά την απομόνωση, οπότε υπάρχει ο κίνδυνος για πολλές ανωμαλίες στα reads και writes. Αντίθετα, το Snapshot Isolation εισάγει μόνο ενός είδους ανωμαλία, η οποία είναι γνωστή ως write skew, και την οποία οι περισσότερες εφαρμογές δεν την προκαλούν. Το Snapshot Isolation ουσιαστικά χωρίζει την απομόνωση του συγχρονισμού μίας συνδιαλλαγής, σε δύο λογικά (logical) μέρη: την **εκκίνηση** της συνδιαλλαγής, κατά την οποία όλα τα reads γίνονται λογικά και το **τέρμα** της συνδιαλλαγής, όπου όλα τα writes γίνονται λογικά.

Το Snapshot Isolation παρέχει ένα πολύ υψηλό επίπεδο απομόνωσης, λόγω του γεγονότος πως οι συνδιαλλαγές κάνουν read από ένα στιγμιότυπο της βάσης δεδομένων, το οποίο βρίσκεται στην κατάσταση της βάσης τη στιγμή που ξεκίνησε η συνδιαλλαγή. Απαιτεί τη χρήση MultiVersion Concurrency Control. Αυτό σημαίνει ότι αντί να αποθηκεύει μία μοναδική έκδοση του κάθε αντικειμένου δεδομένων, ο μηχανισμός βασίζεται στη δημιουργία μίας νέας έκδοσης του ενημερωμένου αντικειμένου δεδομένων κατά τη διάρκεια του commit της συνδιαλλαγής. Έτσι, για ένα αντικείμενο δεδομένων μπορεί να συνυπάρχουν πολλαπλές εκδόσεις του. Αυτές οι εκδόσεις θα πρέπει να έχουν ετικέτες, τέτοιες ώστε να υπάρχει η δυνατότητα της επιλογής των σωστών δεδομένων, για μία συνδιαλλαγή που προσπαθεί να κάνει read στο αντικείμενο αυτό. Συνήθως, για αυτήν τη σηματοδότηση χρησιμοποιούνται λογικές στάμπες χρονοσήμανσης (timestamps).

Το Snapshot Isolation αποφεύγει όλες τις συγκρούσεις read-write, όπως την προαναφερθείσα. Παρόλα αυτά, απαγορεύει τις συγκρούσεις write-write, κάτι το οποίο απαιτεί τον έλεγχο για αυτές τις συγκρούσεις από κάποιο σύστημα διαχείρισης συγκρούσεων. [3]

Κεντρική Επεξεργασία Συνδιαλλαγών

Θεωρούμε τη συνδιαλλαγή ως μία ακολουθία από λειτουργίες read και write σε εγγραφές δεδομένων. Μία λειτουργία read μπορεί να διαβάσει μεμονωμένες

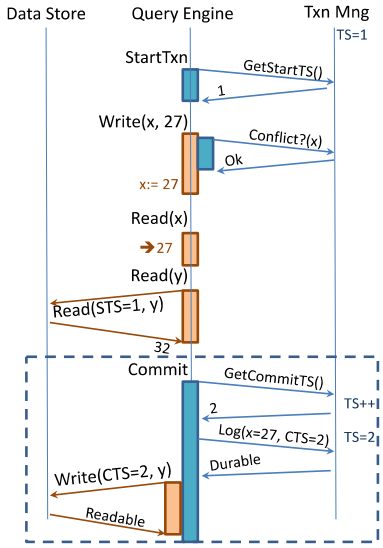
εγγραφές ή συλλογές από εγγραφές που έχουν επιλεγεί από ένα κατηγόρημα. Παρουσιάζεται μία λύση βασιζόμενη στο Snapshot Isolation, το οποίο αποφεύγει τις συγκρούσεις μεταξύ read και write και μπορεί να χρησιμοποιηθεί τόσο για παραδοσιακές σχεσιακές βάσεις, όσο και για NoSQL βάσεις.

Έστω ένα σύστημα πολλαπλών εκδόσεων όπου η κάθε λειτουργία write $w_i(x_i)$ της συνδιαλλαγής T_i στην εγγραφή x δημιουργεί μία νέα ιδιωτική έκδοση x, και κάθε λειτουργία read $r_i(x_j)$ της συνδιαλλαγής T_i διαβάζει την τελευταία έκδοση του x, και η x_j έχει δημιουργηθεί από την T_j, με $j < i$ και δεν υπάρχει άλλη συνδιαλλαγή T_z, η οποία να χει γίνει commit με $j < z < i$. Με ένα τέτοιο σύστημα πολλαπλών εκδόσεων, το Snapshot Isolation απαιτεί τις ιδιότητες για snapshot read και snapshot write.

Το snapshot read απαιτεί η T_i να διαβάζει ένα στιγμιότυπο της βάσης δεδομένων που αντικατοπτρίζει τις τελευταίες εκδόσεις των εγγραφών που έγιναν commit, όταν ξεκίνησε η T_i. Συγκεκριμένα, αυτό σημαίνει ότι αν η T_i κάνει ένα read r_i στη x, τότε διαβάζει είτε μία ιδιωτική έκδοση της T_i, η οποία δημιουργήθηκε προηγουμένως, είτε την έκδοση x_j που δημιουργήθηκε από την T_j, με την T_j να είναι η τελευταία συνδιαλλαγή που έγραψε στη x και έκανε commit, πριν εκκινήσει η T_i.

Το snapshot write απαιτεί δύο ταυτόχρονες συνδιαλλαγές να μην ενημερώνουν την ίδια οντότητα. Αν συμβαίνει αυτό, τότε μία από τις δύο συνδιαλλαγές θα απορριφθεί. Όταν μία συνδιαλλαγή κάνει commit, τότε το timestamp του commit της αυξάνεται και όλες οι ιδιωτικές εκδόσεις μίας συνδιαλλαγής σημειώνονται με αυτό το timestamp. Αν η συνδιαλλαγή αποτύχει, τότε διαγράφονται οι ιδιωτικές εκδόσεις.

Στις περισσότερες υλοποιήσεις του Snapshot Isolation, ο διαχειριστής συνδιαλλαγών διατηρεί ένα μετρητή, ο οποίος χρησιμοποιείται για να σημειώσει τις συνδιαλλαγές με timestamps του commit και της εκκίνησης. Η τιμή του μετρητή αντικατοπτρίζει το commit timestamp της τελευταίας συνδιαλλαγής, η οποία έγινε commit.



Εικόνα 21: Η επεξεργασία συνδιαλλαγών με Snapshot Isolation

Στην αρχή μίας συνδιαλλαγής T_i, ο διαχειριστής συνδιαλλαγών (Txn και Mxn στην εικόνα) ορίζει την τρέχουσα τιμή του μετρητή, TS, ως το timestamp εκκίνησης. Στην εικόνα, η αρχική τιμή του μετρητή είναι 1 (δηλαδή TS=1). Όταν μία συνδιαλλαγή T_i θέλει να κάνει write σε μία εγγραφή x, τότε η λειτουργία write θα εκτελέσει πρώτα έναν έλεγχο συγκρούσεων, με το διαχειριστή συνδιαλλαγών. Μία σύγκρουση θα συμβεί αν υπάρχει μία ταυτόχρονη συνδιαλλαγή T_j και για παράδειγμα, η T_j δεν έχει κάνει commit ή το commit timestamp της είναι μεγαλύτερο από το timestamp εκκίνησης της T_i. Αν δεν υπάρχει σύγκρουση, τότε το write μπορεί να προχωρήσει, δημιουργώντας μία νέα ιδιωτική έκδοση του x, η οποία προς το παρόν είναι ορατή μόνο στην ίδια την T_i. Αν υπάρχει μία σύγκρουση, τότε η T_i πρέπει να απορριφθεί, ώστε να διασφαλιστεί η ιδιότητα του snapshot write. Αυτό σημαίνει, ότι διαγράφονται απλώς οι ιδιωτικές εκδόσεις που έχει δημιουργήσει μέχρι τώρα η T_i.

Όταν μία συνδιαλλαγή T_i απαιτεί να κάνει read μία εγγραφή x, τότε η βάση δεδομένων πρέπει να παρέχει την εγγραφή που δημιουργήθηκε από την T_j με commit timestamp $C(T_j)$, τέτοιο ώστε $C(T_j) \leq S(T_i)$ και δεν υπάρχει έκδοση της x που να δημιουργήθηκε από κάποια T_k, με $C(T_j) < C(T_k) < S(T_i)$. Αυτό διασφαλίζει την ιδιότητα του snapshot read.

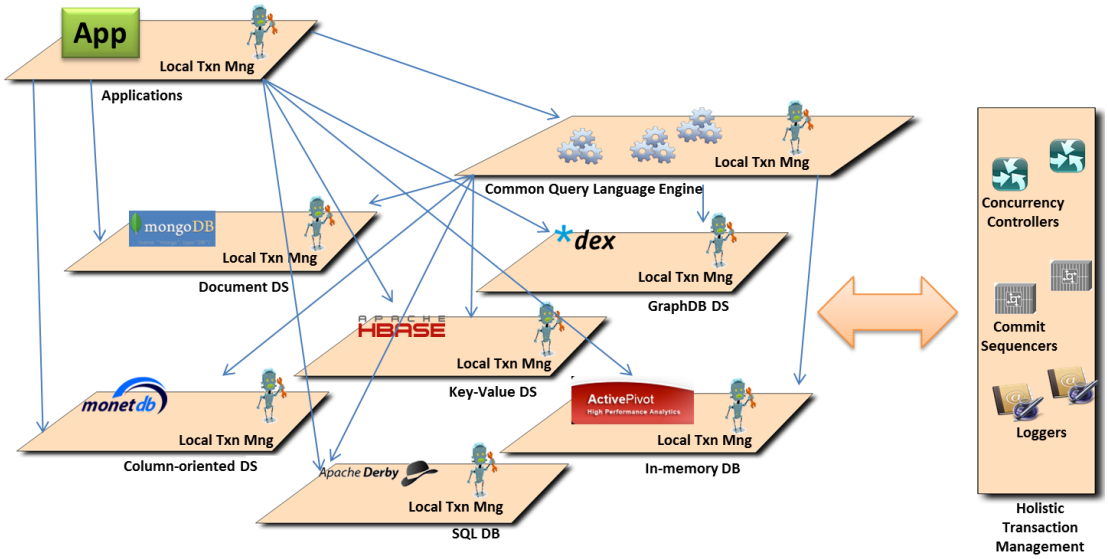
Στο χρόνο του commit μίας συνδιαλλαγής T_i πρέπει να επιτευχθούν αρκετά πράγματα. Πρώτα, η συνδιαλλαγή απαιτεί ένα commit timestamp $C(T_i)$ από το διαχειριστή συνδιαλλαγών, το οποίο γίνεται αναθέτοντας την επόμενη τιμή του μετρητή. Έπειτα, όλες οι εκδόσεις των εγγραφών που δημιουργήθηκαν από την T_i πρέπει να σημειωθούν (σταμπαριστούν) με το timestamp $C(T_i)$. Τότε, οι αλλαγές θα πρέπει να γίνουν ανθεκτικές, κάτι το οποίο συνήθως συμβαίνει διατηρώντας τον κατάλογο redo σε σταθερό αποθηκευτικό χώρο. Μόνο τότε, το commit επιβεβαιώνεται στο χρήστη. Στην ουσία αυτή η επεξεργασία του commit θα πρέπει να είναι μία ατομική ενέργεια. Συγκεκριμένα, η αύξηση του μετρητή στο διαχειριστή συνδιαλλαγών και η ολοκλήρωση των νέων εκδόσεων στον αποθηκευτικό χώρο συνδέονται άμεσα, καθώς μόλις η νέα τιμή του μετρητή $C(T_i)$ ανατίθεται ως timestamp εκκίνησης μίας νέας συνδιαλλαγής T_k, τότε αυτή θα πρέπει να μπορεί να δει τις ενημερώσεις που συμβαίνουν από την T_i. [3]

3.2 *Εκτελώντας συνδιαλλαγές στο νέο περιβάλλον*

Το Coherent PaaS έχει ένα σύνολο από υποσυστήματα που διαδραματίζουν σημαντικό ρόλο την επεξεργασία των συνδιαλλαγών. Αυτά τα μέρη είναι ο Ολιστικός Διαχειριστής Συνδιαλλαγών, η Αποθήκη Δεδομένων και η Μηχανή Κοινών Ερωτημάτων. Υπάρχουν δύο ειδών αποθήκες δεδομένων **α)** Αυτές οι οποίες αφήνουν την επεξεργασία των συνδιαλλαγών εξολοκλήρου στον ολιστικό Διαχειριστή Συνδιαλλαγών και **β)** Αυτές που εκτελούν εσωτερικά επεξεργασία συνδιαλλαγών και αφήνουν μόνο το συντονισμό των συνδιαλλαγών στον ολιστικό Διαχειριστή Συνδιαλλαγών (το σύστημά μας είναι του δεύτερου είδους).

Όλα τα μέρη που έχουν σχέση με τις συνδιαλλαγές έχουν κι έναν τοπικό Διαχειριστή Συνδιαλλαγών (Local TM – LTM). Η πρόσβαση σε αυτόν γίνεται μέσω ενός προγράμματος-πελάτη, του LTMClient Ο LTMClient εκθέτει το API για τη διαχείριση συνδιαλλαγών και ενεργεί ως μία διεπαφή προς τον ολιστικό TM.

Η πρόσβαση στις αποθήκες δεδομένων γίνεται, επίσης, μέσω ενός ενδιάμεσου προγράμματος-πελάτη (proxy client, αναλαμβάνει ρόλο παρόμοιο με τον οδηγό JDBC για τις σχεσιακές βάσεις δεδομένων), ο οποίος επίσης συνυπάρχει με την εφαρμογή-πελάτη και τη μηχανή κοινών ερωτημάτων. Η μηχανή αυτή έχει και το δικό της proxy client που συνυπάρχει με την εφαρμογή. [3]



Εικόνα 22: Η αλληλεπίδραση μεταξύ των υποσυστημάτων της διαχείρισης συνδιαλλαγών

Το API του Διαχειριστή Συνδιαλλαγών

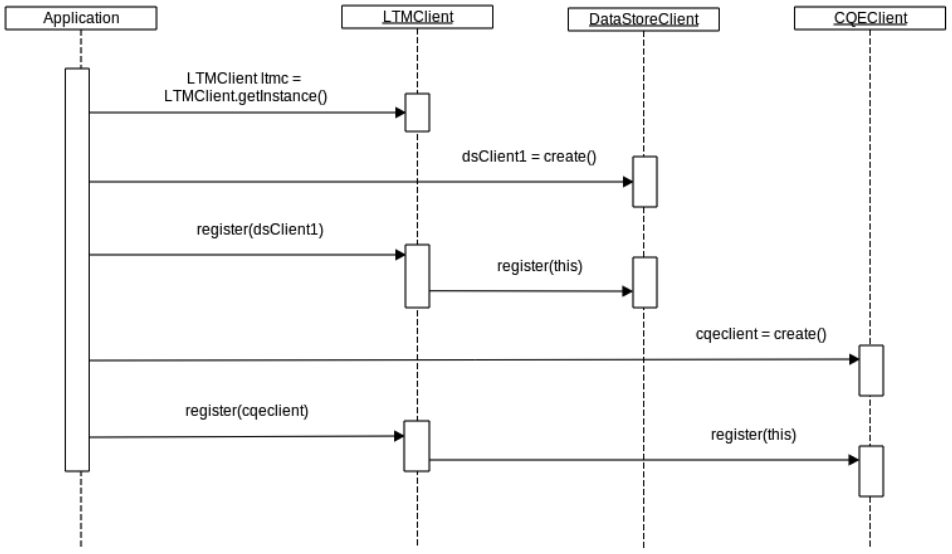
Στο σύστημά μας, μία εφαρμογή μπορεί να έχει πρόσβαση στις αποθήκες δεδομένων είτε μέσω της Κοινής Γλώσσας Ερωτημάτων είτε απευθείας μέσω προγραμμάτων-πελατών στις αποθήκες δεδομένων.

Για την εκτέλεση μίας συνδιαλλαγής απαιτούνται τα εξής βήματα:

1. Αρχικοποίηση του LTMClient
2. Καταχώρηση των προγραμμάτων-πελατών των αποθηκών δεδομένων με τον LTMClient
3. Απόκτηση μίας σύνδεσης συνδιαλλαγών
4. Εκτέλεση μίας ή περισσότερων συνδιαλλαγών, μέσω της σύνδεσης

Το πρώτο βήμα επιτρέπει την πρόσβαση στον LTM, όπως θα έκανε κι ένας JDBC οδηγός. Το δεύτερο βήμα εκτελείται για να επιτραπεί στον LTM να διαδώσει διαφανώς το πλαίσιο της συνδιαλλαγής, απελευθερώνοντας την εφαρμογή από αυτό το βάρος. Το τρίτο βήμα προσφέρει το χειρισμό συνδιαλλαγών για μία περίοδο του CoherentPaaS.

Βήμα 1ο - Αρχικοποίηση και Βήμα 2ο - Καταχώρηση



Εικόνα 23: Το βήμα της καταχώρησης

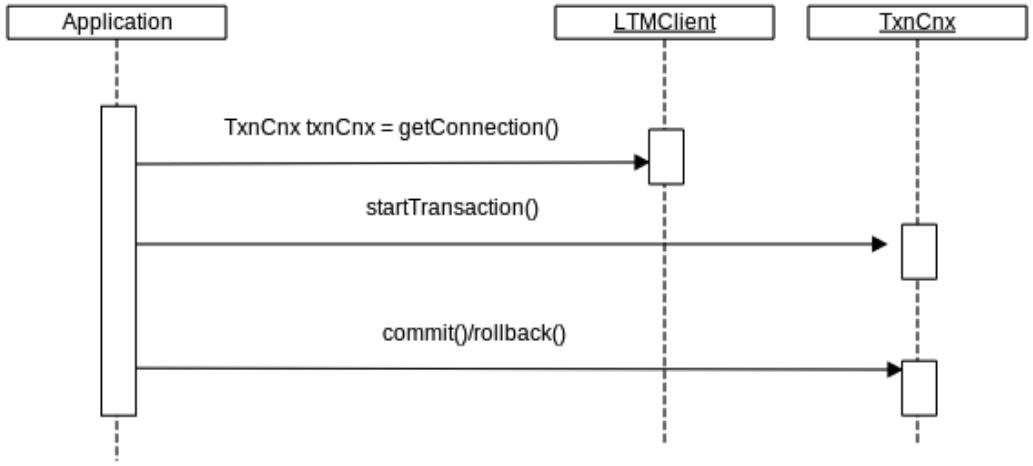
Τα πρώτα βήματα αποτελούνται από την αρχικοποίηση του συστήματος και την καταχώρηση στον TM. Η εικόνα 23 δείχνει πως η εφαρμογή αρχικοποιεί τον LTMClient. Η εφαρμογή θα έχει πρόσβαση τόσο στις αποθήκες δεδομένων απευθείας, όσο και μέσω της Μηχανής Κοινών Ερωτημάτων.

Η εφαρμογή αρχικά ζητά ένα στιγμιότυπο (instance) του LTMClient χρησιμοποιώντας τη μέθοδο getInstance(). Εσωτερικά, ο LTMClient συνδέεται

με τον LTM. Τότε, η εφαρμογή αρχικοποιεί ένα ή περισσότερα προγράμματα-πελάτη των αποθηκών δεδομένων και τους καταχωρεί στον LTMClient. Αυτό σημαίνει ότι ο LTMClient είναι ενήμερος για την ύπαρξη τους. [3]

Βήμα 3ο – Η απόκτηση σύνδεσης

Η εφαρμογή θα πρέπει να αποκτήσει μία σύνδεση πριν να μπορεί να υποβάλλει κάποια λειτουργία συνδιαλλαγών. Η σύνδεση συνδιαλλαγής (TxnCnx) αποκτάται από τον LTMClient και είναι ανάλογη με μία σύνδεση JDBC. Η σύνδεση της συνδιαλλαγής θα πρέπει να διατηρεί το περιεχόμενο (context) της συνδιαλλαγής (TxnCtx). Επίσης, ευθύνεται για την αλληλεπίδραση με τον LTMClient, με έναν τρόπο που θα είναι διαφανής στην εφαρμογή. Η εφαρμογή επικοινωνεί άμεσα με τα προγράμματα-πελάτη των βάσεων δεδομένων, χωρίς να γνωρίζει την ύπαρξη της σύνδεσης συνδιαλλαγής. Τα προγράμματα-πελάτη αυτά χρησιμοποιούν τη σύνδεση για να αλληλεπιδράσουν με τον TM. Υπάρχει μόνο ένα πλαίσιο συνδιαλλαγής ενεργό, κάθε στιγμή, για κάθε συνδιαλλαγή.



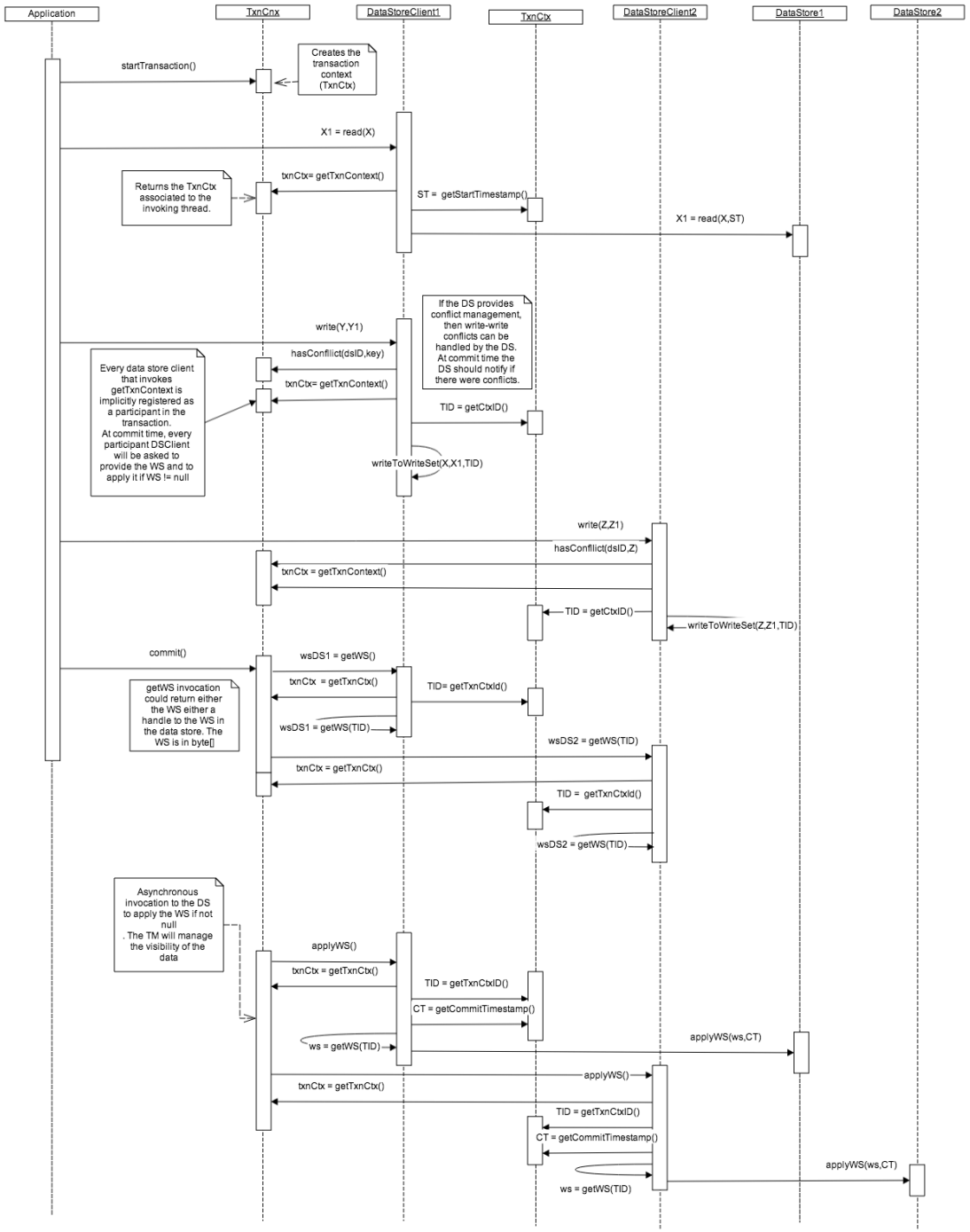
Εικόνα 24: Απόκτηση Σύνδεσης

Η Εικόνα 24 δείχνει την αλληλεπίδραση με τη σύνδεση συνδιαλλαγής. Πρώτα, η εφαρμογή ζητά από τον LTMClient μία σύνδεση. Στη συνέχεια, αυτός επιστρέφει μία σύνδεση συνδιαλλαγής και πλέον η εφαρμογή μπορεί να εκτελέσει συνδιαλλαγές στο σύστημα του CoherentPaaS. [3]

Βήμα 4ο – Εκτέλεση των Συνδιαλλαγών

Στο σύστημά μας χρησιμοποιούμε την εκτέλεση συνδιαλλαγών με απευθείας πρόσβαση στις βάσεις δεδομένων.

Στην εικόνα που ακολουθεί φαίνεται ένα παράδειγμα, όπου η εφαρμογή εκτελεί μία συνδιαλλαγή μέσω του TM του CoherentPaaS και δύο βάσεων δεδομένων, της DataStore1 και της DataStore2, ενώ φαίνονται οι αλληλεπιδράσεις αφού έχει αποκτηθεί η σύνδεση.



Εικόνα 25: Εκτέλεση συνδιαλλαγών

Η εφαρμογή ξεκινά τη συνδιαλλαγή επικαλούμενη την εκκίνηση. Τότε, διαβάζει μία εγγραφή X από τον DataStoreClient1, κάνοντας read(X) στον DataStoreClient1, μέσω του API που παρέχεται από την DataStore1. Εσωτερικά, ο DataStoreClient1 ζητά το πλαίσιο της συνδιαλλαγής (TxnCtx), αποκτά πρόσβαση σε αυτό μέσω μίας τοπικής μεταβλητής νήματος και πλέον έχει όλες τις πληροφορίες που χρειάζεται για τη συνδιαλλαγή, όπως το αναγνωριστικό της (transaction ID – TID) και το timestamp εκκίνησης (ST). ο DataStoreClient1 παίρνει το timestamp εκκίνησης από το TxnCtx μέσω της getStartTimestamp(). Τέλος, επικοινωνεί με τη DataStore1 για να κάνει read τη σωστή έκδοση

(read(X,TS)). Λειτουργίες read σε διαφορετικές αποθήκες εκτελούνται με όμοιο τρόπο.

Οι λειτουργίες write είναι παρόμοιες με τις read, αλλά πρέπει να γίνει έλεγχος για συγκρούσεις. Ο DataStoreClient1 ζητά το TxnCtx από τη σύνδεση συνδιαλλαγής, TxnCnx. Τότε, παίρνει το timestamp εκκίνησης (ST) από το TxnCtx. Έπειτα, ρωτά ξανά τη σύνδεση για να ελέγξει αν υπάρχουν συγκρούσεις write-write (hasConflict), παρέχοντας το αναγνωριστικό του DataStoreClient1 και το κλειδί προς μεταποίηση. Αν υπάρχει σύγκρουση, τότε δεν θα εκτελεστεί. Αν δεν υπάρχει, ο DataStoreClient1 ζητά ξανά το πλαίσιο της συνδιαλλαγής, TxnCtx, για το TID και αποθηκεύει σε έναν εσωτερικό buffer (writetoWriteSet) την λειτουργία write με κλειδί Y, τιμή Y1 και το TID.

Τέλος, η συνδιαλλαγή κάνει commit. Για να γίνει το commit, η εφαρμογή επικαλείται τη διαδικασία commit του LTMClient. Το πρώτο βήμα της φάσης του commit είναι να καταλογιστούν τα write-sets. Η σύνδεση συνδιαλλαγής, TxnCnx ζητά κάθε client βάσης δεδομένων που συμμετέχει στη συνδιαλλαγή (DataStoreClient1 και DataStoreClient2) το write-set τους (getWS). Ο DataStoreClient1 ζητά πάλι το TxnCtx και παίρνει το timestamp του commit, CS_j της συνδιαλλαγής. Οι clients των βάσεων δεδομένων μπορούν να παρέχουν είτε το write-set, είτε να διαχειριστούν το write-set στον εσωτερικό τους κατάλογο, αν έχουν τέτοιο. Όταν ο LTMClient συλλέξει όλα τα write-sets, τα παρέχει στον LTM και περνούν όλα στον κατάλογο του CoherentPaaS. Τότε η συνδιαλλαγή σημειώνεται ως committed και ο LTMClient επιστρέφει τον έλεγχο στην εφαρμογή.

Ο LTMClient ζητά από κάθε client βάσης δεδομένων που συμμετέχει στη συνδιαλλαγή να εφαρμόσει το write-set (applyWS) στην αντίστοιχη αποθήκη δεδομένων, ώστε να το καταστήσει ορατό σε μελλοντικές συνδιαλλαγές. Πάλι, κάθε client χρησιμοποιεί το TxnCtx για να πάρει το TID. Έπειτα, παίρνουν το αντίστοιχο write-set και εφαρμόζουν κάθε αλλαγή που έγινε από τη συνδιαλλαγή στην υποκείμενη βάση δεδομένων. Όταν τελειώσει αυτή η φάση, τότε ο LTMClient ενημερώνει τον LTM ότι η συνδιαλλαγή είναι ανθεκτική και αναγνώσιμη. Ο LTM θα αυξήσει το μετρητή στιγμιότυπου, όταν αυτό είναι δυνατόν.

4

Σχεδίαση Συστήματος

Για την εκτέλεση ενός συστήματος DBMS με MVCC θα πρέπει:

- Αν πολλαπλές εκδόσεις αποθηκεύονται στη βάση δεδομένων, ένας αποδοτικός μηχανισμός συλλογής σκουπιδιών απαιτείται για να συλλέξει και να αφαιρέσει τις παλιές εκδόσεις, οι οποίες δεν χρειάζονται πλέον.
- Το σύστημα διαχείρισης της βάσης δεδομένων πρέπει να παρέχει αποδοτικές μεθόδους πρόσβασης που αποφεύγουν να ελέγχουν πλεονάζουσες εκδόσεις.
- Το σύστημα διαχείρισης της βάσης δεδομένων πρέπει να αποφεύγει τις ακριβές αναζητήσεις όταν καθορίζει το σχετικό χρόνο του commit μίας συνδιαλλαγής.

Υπάρχουν ουσιαστικά δύο τρόποι για να προσεγγίσει κανείς το MVCC. Ο πρώτος είναι η αποθήκευση πολλαπλών εκδόσεων των εγγραφών στη βάση δεδομένων και η συλλογή σκουπιδιών των εγγραφών που δε χρειάζονται περαιτέρω.

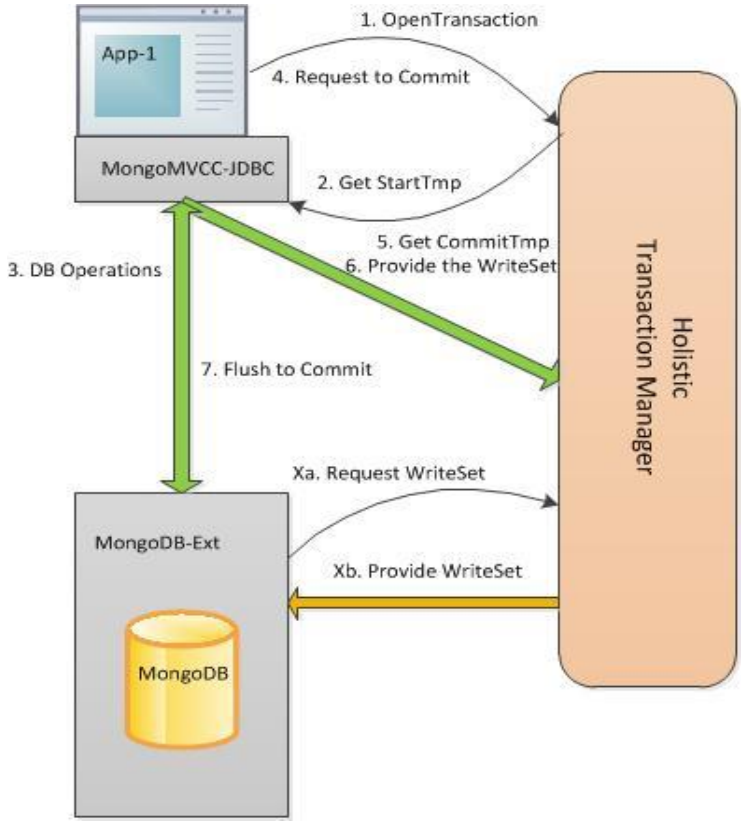
Ο δεύτερος τρόπος είναι η διατήρηση μόνο της τελευταίας έκδοσης των δεδομένων στη βάση δεδομένων, όπως στις εκτελέσεις μίας έκδοσης, άλλα η επαναδημιουργία παλιότερων εκδόσεων των δεδομένων δυναμικά, ανάλογα με το πώς απαιτείται, εκμεταλλευόμενοι τις πληροφορίες του καταλόγου για Μελλοντική Εγγραφή (Write Ahead log). [2]

Στη δική μας έκδοση της MongoDB MVCC χρησιμοποιείται η πρώτη προσέγγιση, δηλαδή της αποθήκευσης των πολλαπλών εκδόσεων στη βάση δεδομένων.

4.1 Γενική Αρχιτεκτονική για τη MongoDB

Για να γίνει δυνατή η ενσωμάτωση του Διαχειριστή Συνδιαλλαγών (Transaction Manager – TM) του CoherentPaaS, ο οποίος καθιστά δυνατές τις ταυτόχρονες συνδιαλλαγές και να διαβεβαιώνει τη σημασιολογία ACID (Atomicity, Consistency, Isolation και Durability), απαιτούταν η MongoDB να έχει τη δυνατότητα να χειριστεί πολλές εκδόσεις των ίδιων αντικειμένων. Επίσης, έχει ενσωματωθεί και ο τοπικός TM που μπορεί να χειριστεί ταυτόχρονες συνδιαλλαγές και να λύνει τις συγκρούσεις write-write. Μεταβάλαμε τέλος το μηχανισμό επαναφοράς της MongoDB για να μπορεί να στηρίζεται στον TM για να καταστήσει τις συνδιαλλαγές ανθεκτικές, παρέχοντας τις δυνατότητες του write-set του.

Αρχικά η MongoDB έπρεπε να υιοθετήσει όλες εκείνες τις απαραίτητες λειτουργίες για να μπορεί να ενσωματώσει πλήρως τον ολιστικό διαχειριστή συνδιαλλαγών του CoherentPaaS. Αυτό επιτρέπει σε έναν προγραμματιστή εφαρμογών να αναπτύξει εφαρμογές που μπορούν να εκμεταλλευτούν τις φυσικές ικανότητες της MongoDB, ενισχυμένες από τον ACID διαχειριστή συνδιαλλαγών που προσφέρεται από την πλατφόρμα CoherentPaaS. Αυτή η αρχιτεκτονική παρουσιάζεται στην παρακάτω Εικόνα:



Εικόνα 26: Η αρχιτεκτονική του συστήματος της MongoDB MVCC

Στην εικόνα παρατηρούμε τα εξής μέρη του συστήματος:

- **App-1 – Application:** Είναι η εφαρμογή που θα χρησιμοποιήσει τη MongoDB, ολοκληρωμένη με το CoherentPaaS.
- **MongoMVCC-Driver:** Είναι ο οδηγός JDBC που θα χρησιμοποιηθεί από την εφαρμογή για να έχει πρόσβαση στη MongoDB.
- **Holistic Transaction Manager:** Είναι το κύριο κομμάτι του CoherentPaaS, το οποίο προσφέρει τη σημασιολογία ACID συνδιαλλαγών.
- **MongoDB:** Ο φυσικός MongoDB εξυπηρετητής.
- **MongoDB-Ext:** Αυτό το μέρος περιέχει όλες τις απαραίτητες επεκτάσεις του φυσικού MongoDB εξυπηρετητή που επιβεβαιώνει την ολοκλήρωση της MongoDB από το CoherentPaaS και υλοποιεί όλες τις πρόσθετες λειτουργίες.

Κατά τη διάρκεια μίας συνδιαλλαγής, από το παραπάνω σχήμα, προκύπτουν οι εξής ενέργειες για την ολοκλήρωσή της:

1. Η εφαρμογή χρειάζεται να δημιουργήσει μία νέα συνδιαλλαγή. Αρχικά επικοινωνεί με τον ολιστικό Διαχειριστή Συνδιαλλαγών, μέσω μίας τοπικής υλοποίησης προγράμματος-πελάτη που περιέχεται στον Οδηγό MongoMVCC-Driver, για την εκκίνηση της συνδιαλλαγής.
2. Ο ολιστικός Διαχειριστής Συνδιαλλαγών επιστρέφει τα πλαίσια της συνδιαλλαγής (transaction's context) και το πιο σημαντικό τη στάμπα χρονοσήμανσης εκκίνησης (start timestamp) και το αναγνωριστικό της (identifier).
3. Σε αυτό το σημείο, η εφαρμογή έχει τη δυνατότητα να αλληλεπιδράσει με τη MongoDB, χρησιμοποιώντας τον έλεγχο της MultiVersion Concurrency που προσφέρει την αντίστοιχη έκδοση των αντικειμένων δεδομένων, σύμφωνα με το timestamp εκκίνησης της συνδιαλλαγής. Κάθε λειτουργία ενημέρωσης (εισαγωγή, ενημέρωση, διαγραφή) διατηρεί μία ιδιωτική έκδοση του αντικειμένου δεδομένων που επηρεάζεται στο ιδιωτικό πλαίσιο της συνδιαλλαγής, ανάλογα με το χειρισμό του MongoMVCC-Driver.
4. Όταν η εφαρμογή είναι έτοιμη να παραδώσει (commit), ενημερώνει τον ολιστικό TM, ούτως ώστε αυτός να μπορέσει να διαπράξει όλες τις απαραίτητες ενέργειες για να ετοιμάσει τη φάση παράδοσης (commit phase).
5. Αν δεν υπάρχουν συγκρούσεις write σε αυτή τη φάση, τότε ο TM ενημερώνει το πλαίσιο της συνδιαλλαγής με τη στάμπα χρονοσήμανσης της παράδοσης (commit timestamp) και ενημερώνει την εφαρμογή για να ετοιμαστεί να παραδώσει.

6. Πριν το commit, ο MongoMVCC-Driver πρέπει να περάσει το ιδιωτικό σύνολο εγγραφής (write-set) στον TM, ώστε αυτός να το καταστήσει ανθεκτικό για να είναι δυνατή η κατάσταση επαναφοράς σε περίπτωση ενδεχόμενης αποτυχίας. Η MongoDB δε θα στηριχθεί στο φυσικό μηχανισμό της, καθώς αυτός εισάγει πολύ μικρά χρονικά διαστήματα, κατά τα οποία τα δεδομένα δεν μπορεί να θεωρούνται ανθεκτικά και μπορεί να οδηγήσουν σε απώλειες write. Συνεπώς, η MongoDB θα παρέχει στον TM το πλήρες ιδιωτικό write-set (όλες οι ενημερωμένες εκδόσεις των δεδομένων στα πλαίσια της συνδιαλλαγής) ώστε αυτός να καταστήσει δυνατή την ιδιότητα της ανθεκτικότητας.
7. Αφού η ανθεκτικότητα διασφαλιστεί, για να τελειοποιηθεί το commit, ο MongoMVCC-Driver καθαρίζει (flushes) όλες τις αλλαγές στη MongoDB. Η MongoDB-Ext καταχωρεί όλες τις εκκρεμούσες συνδιαλλαγές, ούτως ώστε να μπορεί να διαπράξει ενέργειες επαναφοράς, σε περίπτωση αποτυχίας.
8. Αν προκύψει μία αποτυχία κατά την ολοκλήρωση της φάσης του commit, τότε η MongoDB-Ext την αναγνωρίζει και απαιτεί το write-set της αποτυχούσας συνδιαλλαγής από τον TM.
9. Ο TM προσφέρει το απαιτούμενο write-set και η MongoDB-Ext προχωρά στις απαραίτητες ενέργειες επαναφοράς. [1]

4.2 Επιλογές υλοποίησης

Για να λειτουργήσει το σύστημά μας με το MultiVersion Concurrency και το CoherentPaaS πρέπει όλες οι συνδιαλλαγές να μπορούν να δουν διαφορετικές εκδόσεις των δεδομένων, σύμφωνα με τα timestamp εκκίνησής τους. Το MVCC δεν υποστηρίζεται από τη φυσική MongoDB και έτσι η προσέγγισή μας βασίζεται σε δουλειά που έχει γίνει πάνω σε άλλες βάσεις δεδομένων, κυρίως στην PostgreSQL. [2]

Το κάθε δεδομένο στη MongoDB MVCC είναι της μορφής:

(_id, _dataID, _cmtTmstmp, _nextCmtTmstmp, _isDeleted, document-Data)

όπου έχουμε:

- **_id:** Είναι το απαιτούμενο, για κάθε έγγραφο που αποθηκεύεται στη βάση, μοναδικό κλειδί της MongoDB. Αποτελείται από 20 bytes, τα οποία είναι συνένωση των μοναδικών _dataID και _cmtTmstmp. Αυτή η συνένωση επιτρέπει εκδόσεις του ίδιου αντικειμένου δεδομένων να αποθηκεύονται μαζί.
- **_dataID:** Είναι ένα αναγνωριστικό δεδομένων, 12 bytes, του τύπου «ObjectId», το οποίο παρέχεται από τις φυσικές βιβλιοθήκες της MongoDB, η οποία αυτόματα παράγει αυτή την τιμή με έναν

κατανεμημένο τρόπο. Καθιστά βέβαιη τη μοναδικότητα του κλειδιού μεταξύ των διαφόρων στιγμιότυπων. Το αναγνωριστικό μοιάζει με την κλάση UUID της Java, η οποία επίσης χρησιμοποιείται για την παροχή μοναδικών κλειδιών σε κατανεμημένες εφαρμογές.

- **_cmtTmstmp:** Πρόκειται για μία στάμπα χρονοσήμανσης μήκους 8 bytes, η οποία παρέχεται από τον TM κατά το commit.
- **_nextCmtTmstmp:** Είναι μία στάμπα χρονοσήμανσης μήκους 8 bytes, η οποία παρέχεται από τον TM. Αν το έγγραφο είναι στην τελευταία έκδοσή του, τότε αυτό το πεδίο θα πρέπει να χει την τιμή *null.* Σε διαφορετική περίπτωση, δείχνει στο επόμενο commit timestamp της αμέσως επόμενης ενημερωμένης έκδοσης.
- **_isDeleted:** Αυτό το πεδίο-σημαία (flag) δείχνει αν ένα αντικείμενο είναι ή όχι διαγεγραμμένο.

Η MongoDB υποστηρίζει όλες τις βασικές λειτουργίες CRUD (Create, Read, Update, Delete). Η κάθε μία από αυτές θα πρέπει να υλοποιηθεί και για τη MongoDB MVCC. Στη δική μας υλοποίηση αυτές οι λειτουργίες παίρνουν την εξής μορφή:

Read

Το read παρέχεται από μία προκαθορισμένη συνθήκη που εφαρμόζεται από τη MongoDB πάνω στον αποθηκευτικό χώρο. Τα έγγραφα που ικανοποιούν αυτήν τη συνθήκη επιλέγονται και ανακτώνται. Στη MongoDB MVCC οι λειτουργίες read απαιτούν επιπλέον μία τιμή timestamp εκκίνησης και επίσης, τα ανακτώμενα έγγραφα θα πρέπει να ικανοποιούν τη συνθήκη:

```
(cmtTmstmp <= startTmstmp)
AND ((nextCmtTmstmp > startTmp) OR (nextCmtTmstmp is null))
AND (isDeleted is null)
```

Μέσω αυτής της συνθήκης, είναι πάντα βέβαιο πως μόνο η τελευταία μη-διαγεγραμμένη έκδοση που ικανοποιεί και την προκαθορισμένη συνθήκη, θα ανακτηθεί. Ο εσωτερικός μηχανισμός ερωτημάτων της MongoDB είναι υπεύθυνος για την επιλογή των κατάλληλων δεδομένων στο εύρος, χωρίς να πρέπει να εισαχθεί ένα επιπλέον φορτίο απόδοσης.

Insert

Όταν εισάγεται ένα νέο αντικείμενο στη MongoDB, το cmtTmstmp παίρνει την τιμή του commit timestamp της συνδιαλλαγής, ενώ το nextCmtTmstmp και το isDeleted έχουν τιμή *null.*

Update

Όταν ένα αντικείμενο δεδομένων ενημερώνεται, ο MongoMVCC-Driver πρέπει να κατευθύνει στη MongoDB την ενημέρωση, πρώτα, της τιμής του nextCmtTmstmp της τελευταίας έκδοσης στο commit timestamp της τρέχουσας συνδιαλλαγής και ύστερα να εισάγει μία νέα έκδοση του αντικειμένου.

_id	_dataID	Εργαζόμενος	Αμοιβή	_cmtTmstmp	_nextCmtTmstmp	_isDeleted
…	…	…	…	…	…	…
…	…	…	…	…	…	…
1010	10	Γιώργος	500	10	25	*null*
1025	10	Γιώργος	1000	25	28	*null*
1028	10	Γιώργος	2000	28	*null*	*null*

Εικόνα 27: Παράδειγμα update μίας εγγραφής σε βάση δεδομένων εργαζομένων

Delete

Όταν ένα αντικείμενο δεδομένων διαγράφεται, ο MongoMVCC-Driver πρέπει να επικαλεσθεί όλες τις λειτουργίες που γίνονται και στη διαδικασία ενημέρωσης και στη συνέχεια να θέσει τη σημαία isDeleted στην τιμή «1» για την τελευταία έκδοση του αντικειμένου.

Η διατήρηση πολλαπλών εκδόσεων αντικειμένων δεδομένων στη βάση, δημιουργεί το πρόβλημα της σημαντικής αύξησης του όγκου της. Παρόλα αυτά, ενόσω οι τρέχουσες συνδιαλλαγές τερματίζονται, μερικές εκδόσεις γίνονται αχρείαστες και δεν υπάρχει λόγος για τη διατήρησή τους. Η MongoDB-Ext πρέπει να ενημερώνεται περιοδικά από τον ολιστικό TM για το ποιες εκδόσεις είναι αχρείαστες για να διαγραφούν από το συλλέκτη σκουπιδιών.

Επιλογές Υλοποίησης για το Διαχειριστή Συνδιαλλαγών

Η φυσική έκδοση της MongoDB δεν καθιστά βέβαιο τον ACID χειρισμό των συνδιαλλαγών. Παρόμοια, η έκδοση της MongoDB MVCC που θα ολοκληρώνεται με το CoherentPaaS δεν παρέχει ACID συνδιαλλαγές από μόνη της. Αυτό το κάνει ο ολιστικός TM, καθώς ο MongoMVCC-Driver τον ελέγχει για να αναγνωρίσει συγκρούσεις write-write. Όπως περιγράφηκε και πιο πάνω, μία νέα συνδιαλλαγή δέχεται μία timestamp εκκίνησης και το αναγνωριστικό της συνδιαλλαγής. Κατά τη διάρκεια μίας λειτουργίας read, κάθε συνδιαλλαγή επιλέγει δεδομένα περιλαμβάνοντας επιπροσθέτως την timestamp εκκίνησης, ώστε η MongoDB να μπορεί να ανιχνεύσει για τις κατάλληλες εκδόσεις. Κατά τη διάρκεια του commit, ο TM καθορίζει μία νέα timestamp για το commit που θα χαρακτηρίζει τη νέα εισαχθείσα έκδοση των αντικειμένων δεδομένων. Για την κατάλληλη διαχείριση των συνδιαλλαγών έχουν προστεθεί κάποιες λειτουργίες στον TM και έχουν γίνει βελτιώσεις στην αρχική του έκδοση:

Reads Στιγμιότυπων και Ιδιωτική Διαχείριση Εκδόσεων

Όσον αφορά στο Snapshot Isolation του συστήματός μας, του οποίου οι απαιτήσεις αναφέρθηκαν στο κεφάλαιο 2:

Όταν εκτελείται μία λειτουργία ενημέρωσης/διαγραφής, μία νέα έκδοση ενός αντικειμένου δεδομένων εισάγεται, το οποίο απαιτεί όπως ο MongoMVCC-Driver να ελέγξει τον TM για πιθανές write-write συγκρούσεις. Μία σύγκρουση μπορεί

να προκύψει λόγω μίας τρέχουσας συνδιαλλαγής, η οποία αρχικοποιείται από τον MongoMVCC-Driver, ο οποίος έχει ήδη μία αίτηση για ενημέρωση του ίδιου αντικειμένου ή εξαιτίας μίας τρέχουσας συνδιαλλαγής που αρχικοποιείται από μία αποθήκη δεδομένων σε ένα σενάριο πολλαπλών-αποθηκών δεδομένων, η οποία επίσης ζητά μία ενημέρωση. Ο MongoMVCC-Driver εισάγει τη μέθοδο «hasConflict» στο τοπικό πρόγραμμα-πελάτη διαχείρισης, για κάθε αντικείμενο δεδομένων που προσπαθεί να διαπράξει μία ενημέρωση. Αυτή η μέθοδος αναμένει το αναγνωριστικό από την αντίστοιχη αποθήκη δεδομένων και το κλειδί του αντικειμένου δεδομένων για να ελεγχθεί για συγκρούσεις σε έναν πίνακα από bytes. Το κλειδί σχηματίζεται από τη συνένωση του ονόματος της βάσης δεδομένων που χρησιμοποιείται και το όνομα της αντίστοιχης συλλογής συν το «_dataID» κλειδί από το αντικείμενο δεδομένων, μετατρέποντάς το σε έναν πίνακα από bytes. Αν μία σύγκρουση write-write αναγνωρίζεται από τον TM, μία εξαίρεση TransactionManagerException θα αρθεί, η οποία θα επιβάλει στη συνδιαλλαγή να πάει σε μία προηγούμενη κατάσταση. Παρόλα αυτά, όταν μία λειτουργία εισαγωγής λαμβάνει χώρα, ο MongoMVCC-Driver δεν χρειάζεται να ελέγξει για συγκρούσεις, αφού δεν υπάρχει πρόσβαση ποτέ στο νέο εισαχθέν αντικείμενο από άλλες ταυτόχρονες συνδιαλλαγές.

Όσον αφορά στα reads στιγμιότυπων (snapshot reads), ο MongoMVCC-Driver υποβάλλει ένα read (στην ορολογία της MongoDB αυτό αναφέρεται σε μία λειτουργία find) ερώτημα , παρέχοντας επίσης την timestamp εκκίνησης της συνδιαλλαγής. Η MongoDB χειρίζεται το ερώτημα και επιστρέφει έναν iterator τύπου DBCursor, για πρόσβαση στα αποτελέσματα. Η MongoDB αποθηκεύει μόνο αντικείμενα που είναι δημοσίως ορατά (publicly visible), αποτρέποντας έτσι την πρόσβαση μίας συνδιαλλαγής στο ιδιωτικό σύνολο μίας άλλης. Παρόλα αυτά, μία ανοιχτή συνδιαλλαγή μπορεί να έχει ήδη ενημερώσει κάποια έγγραφα και συνεπώς, κάποια αποτελέσματα από το επιστρέφον σύνολο μπορεί να είναι παλιότερης-έκδοσης (outdated), από τη σκοπιά της συνδιαλλαγής αυτής. Γι' αυτό το λόγο, η επιστροφή των αποτελεσμάτων γίνεται σε δύο στάδια: Πρώτα, Ο MongoMVCCCursor διατρέχει αυξανόμενος (iterates) το ιδιωτικό write-set της συνδιαλλαγής. Χρησιμοποιεί τη δική του υλοποίηση για την επαλήθευση των εκφράσεων, για να ελέγξει αν ένα αντικείμενο δεδομένων που είναι αποθηκευμένο στη βάση επαληθεύει τις συνθήκες του ερωτήματος. Αν ναι, τότε το επιστρέφει πίσω στην εφαρμογή. Διαφορετικά, το παραβλέπει και ο iterator προχωρά στο επόμενο αντικείμενο. Όταν τελειώσει την ανίχνευση του write-set, τότε διατρέχει το σύνολο αποτελεσμάτων της MongoDB. Για κάθε αντικείμενο που περιλαμβάνεται σ' αυτό το σύνολο, ελέγχει αν το μοναδικό κλειδί (το πεδίο _dataID δηλαδή) περιέχεται ήδη στο write-set της συνδιαλλαγής. Αν όχι, τότε επιστρέφεται στην εφαρμογή. Αν ήδη περιέχεται, τότε παραβλέπεται το αντικείμενο και ο iterator προχωρά στο επόμενο αντικείμενο. Αυτό συμβαίνει γιατί αυτό το αντικείμενο θα έχει ήδη εξετασθεί από το δρομέα (cursor) στο πρώτο στάδιο, οπότε το αντικείμενο αυτό είναι παλιότερης έκδοσης, όσον αφορά σ' αυτήν τη συνδιαλλαγή. [1]

Υποστήριξη Ευρετηρίου

Η MongoDB παρέχει, εξ' ορισμού, ένα Β-Δέντρο (B-Tree) ευρετήριο για το πεδίο «_id», το οποίο χρησιμοποιείται για την αναγνώριση του κλειδιού του κάθε εγγράφου που είναι αποθηκευμένο στη βάση δεδομένων. Επιπλέον, επιτρέπει

στον προγραμματιστή της εφαρμογής να καθορίσει πρόσθετα ευρετήρια σε άλλα πεδία, ώστε να μπορεί να εκτελεί αποτελεσματικές, ως προς τις επιδόσεις, ανιχνεύσεις-αναζητήσεις. Αντίστοιχα, βασιζόμενοι στο φυσικό μηχανισμό της MongoDB, χρησιμοποιώντας ένα κοινό κλειδί από τα «dataID» και «cmtTmstmp» εφαρμόζεται παρόμοιος μηχανισμός και στη MongoMVCC της υλοποίησής μας. Κατά τη δημιουργία μίας νέας συλλογής, ο MongoMVCC-Driver επιβάλλει στη MongoDB να δημιουργήσει αυτό το ευρετήριο, εξ' ορισμού. [1]

Οριστικοποίηση του commit

Όταν η εφαρμογή κάνει commit για μία ανοιχτή συνδιαλλαγή, επικαλείται το τοπικό πρόγραμμα-πελάτη για να το κάνει. Αυτό, με τη σειρά του, ζητά από τον MongoMVCC-Driver να παρέχει το πλήρες write-set των ιδιωτικών εκδόσεων της συνδιαλλαγής, ώστε να το καταστήσει ανθεκτικό. Η ολοκληρωμένη MongoDB δε βασίζεται στο φυσικό μηχανισμό της για την επαναφορά σε περιπτώσεις αποτυχίας. Αντί αυτού, θα ζητήσει το write-set των αποτυχημένων συνδιαλλαγών από τον TM. Σε αυτό το σημείο, ο MongoMVCC-Driver παράγει έναν πίνακα από bytes για όλα τα αντικείμενα δεδομένων που είναι αποθηκευμένα στο ιδιωτικό write-set. Αυτό επιτυγχάνεται μέσω της ακόλουθης διαδικασίας: Πρώτα, ένα HashMap της Java δημιουργείται για κάθε συλλογή, το οποίο περιέχει αντικείμενα, των οποίων το κλειδί είναι η συνένωση των ονομάτων της βάσης δεδομένων της συλλογής και του ονόματος της συλλογής και του αναγνωριστικού του εγγράφου. Η τιμή του είναι η νέα έκδοση του εγγράφου που δημιουργήθηκε. Αυτά τα αντικείμενα HashMap μετατρέπονται σε πίνακα byte χρησιμοποιώντας το ByteArrayOutputStream της Java για κάθε συλλογή και επιστρέφονται στον TM ώστε να μπορέσει να κάνει ανθεκτικό το write-set. Με αυτή τη διαδικασία είναι πιο βολικό να επαναδημιουργηθεί το write-set σε περίπτωση αποτυχίας της συνδιαλλαγής, για ένα μη-ντετερμινιστικό λόγο, και να εκτελεστούν και οι απαραίτητες επανορθωτικές ενέργειες από το μέρος του Mongo-Ext. Όταν το write-set γίνει επιτυχώς commit στη MongoDB, τότε ο TM ειδοποιείται ώστε να μπορέσει να καταστήσει τις νέοεισαχθείσες εκδόσεις ορατές. [1]

Μηχανισμός Καταλόγου και Επαναφοράς

Η ολοκληρωμένη MongoDB δε βασίζεται στο φυσικό μηχανισμό για καταλογισμό και επαναφορά, καθώς εισάγει ένα σημαντικό διάστημα που θα μπορούσε να οδηγήσει σε απώλεια δεδομένων και ασυνέπεια. Η MongoDB υποβάλλει όλα τα write-sets της σε ένα ενδιάμεσο αρχείο, ώστε να αποφύγει πολλαπλές προσβάσεις στο χώρο αποθήκευσης, κάτι το οποίο θα εισήγαγε ένα επιπρόσθετο λεπτό σημείο συνωστισμού για την επίδοση (bottleneck). Το αρχείο αυτό αποθηκεύεται μόνιμα στο σκληρό δίσκο σε κάθε περίοδο του χρονικού αυτού διαστήματος. Αυτό καθιστά τα τροποποιημένα δεδομένα ευάλωτα σε απώλειες.

Έτσι, η ολοκληρωμένη MongoDB βασίζεται στο μηχανισμό καταλογισμού του TM και ζητά από αυτόν το write-set για να το εφαρμόσει ξανά σε περίπτωση αποτυχίας συνδιαλλαγής. [1]

Αλλαγές και Βελτιώσεις από τις αρχικές εκδόσεις του CoherentPaaS

Η φυσική MongoDB δεν παρέχει MVCC λειτουργίες, όπως αναφέρθηκε και προηγουμένως, κάτι το οποίο είναι απαραίτητο για τον ολιστικό TM του CoherentPaaS. Έτσι η MVCC έπρεπε να σχεδιαστεί εξαρχής.

Αρχικά δεν υπήρχε το πεδίο του nextCmtTmstmp. Αυτό εισήχθη λόγω των αργών επιδόσεων των ερωτημάτων εύρους σε ένα πεδίο διαφορετικό από το πρωτεύον κλειδί του εγγράφου. Επιπλέον, υπήρχαν περιπτώσεις που η τελευταία έκδοση ενός αντικειμένου δεν ικανοποιεί ένα τέτοιο ερώτημα, αλλά μία παλαιότερη το ικανοποιεί. Έτσι, έχοντας μόνο το cmtTmstmp, δεν είναι δυνατό να αναγνωρίσουμε ποια έκδοση είναι η τελευταία, χωρίς σημαντική μείωση των επιδόσεων.

Για παράδειγμα, στον παρακάτω πίνακα:

_id	_dataID	Εργαζόμενος	Αμοιβή	_cmtTmstmp	_isDeleted
…	…	…	…	…	…
…	…	…	…	…	…
1010	10	Γιώργος	400	10	*null*
1025	10	Γιώργος	500	25	*null*
1028	10	Γιώργος	1000	28	*null*

ένα ερώτημα με καθορισμό του (Αμοιβή < 800) στην έκδοση 30, θα επιστρέψει τις 1010 και 1025 και από αυτές, η MongoDB θα επιλέξει το Γιώργο με αμοιβή 500 ως μία έγκυρη απάντηση στο ερώτημα, πράγμα το οποίο είναι λάθος.

Αντίθετα στην περίπτωση που έχουμε το nextCmtTmstmp :

_id	_dataID	Εργαζόμενος	Αμοιβή	_cmtTmstmp	_nextCmtTmstmp	_isDeleted
…	…	…	…	…	…	…
…	…	…	…	…	…	…
1010	10	Γιώργος	400	10	25	*null*
1025	10	Γιώργος	500	25	28	*null*
1028	10	Γιώργος	1000	28	*null*	*null*

δεν παρουσιάζεται αυτό το πρόβλημα, καθώς ο Γιώργος φαίνεται πως έχει αμοιβή μεγαλύτερη των 800, στην τελευταία έκδοση του αντικειμένου. [1]

5

Υλοποίηση

Στο κεφάλαιο αυτό γίνεται αναφορά στις τεχνολογίες και τα προγράμματα που χρησιμοποιήθηκαν και έκαναν δυνατή αυτήν την εργασία στη βάση της. Επίσης, στο τελευταίο κομμάτι παρουσιάζονται οι σημαντικότερες κλάσεις των υποσυστημάτων του ολοκληρωμένου συστήματος της MongoDB MVCC, όπως αυτό παρουσιάστηκε στο προηγούμενο κεφάλαιο.

5.1 Πλατφόρμες και προγραμματιστικά εργαλεία

5.1.1 Η Γλώσσα Προγραμματισμού Java

5.1.1.1 Αντικειμενοστρεφής Προγραμματισμός

Ο Αντικειμενοστρεφής προγραμματισμός είναι ένα θεμελιώδες ύφος προγραμματισμού υπολογιστών, το οποίο βασίζεται στην έννοια των «αντικειμένων». Τα αντικείμενα είναι δομές δεδομένων που περιέχουν δεδομένα, στη μορφή πεδίων, τα οποία είναι συνήθως γνωστά ως γνωρίσματα (attributes); και κώδικα, στη μορφή διαδικασιών, οι οποίες συνήθως αποκαλούνται μέθοδοι (methods). Ένα διακριτικό χαρακτηριστικό των αντικειμένων είναι ότι οι διαδικασίες ενός αντικειμένου μπορούν να έχουν πρόσβαση και συχνά να μεταβάλλουν τα πεδία δεδομένων του αντικειμένου με το οποίο είναι συσχετισμένα. Στον αντικειμενοστρεφή προγραμματισμό, τα προγράμματα υπολογιστών σχεδιάζονται φτιάχνοντάς τα από αντικείμενα τα οποία αλληλεπιδρούν μεταξύ τους. Υπάρχει μεγάλη ποικιλία στον αντικειμενοστρεφή προγραμματισμό, αλλά οι πιο δημοφιλείς γλώσσες είναι βασισμένες σε κλάσεις

(class-based), το οποίο σημαίνει ότι τα αντικείμενα είναι στιγμιότυπα των κλάσεων, κάτι το οποίο συνήθως καθορίζει και τον τύπο τους. [29] [30]

Τα θεμελιώδη χαρακτηριστικά του αντικειμενοστρεφούς προγραμματισμού, συναντιούνται στις περισσότερες αντικειμενοστρεφείς γλώσσες [31]:

- **Δυναμική Αποστολή :** όταν μία μέθοδος επικαλείται σε ένα αντικείμενο, το ίδιο το αντικείμενο καθορίζει ποιο κομμάτι κώδικα εκτελείται. Αυτό γίνεται βρίσκοντας τη μέθοδο σε έναν πίνακα που σχετίζεται με το αντικείμενο, κατά το χρόνο εκτέλεσης. Το συγκεκριμένο χαρακτηριστικό διακρίνει ένα αντικείμενο από έναν αφηρημένο τύπο δεδομένων (abstract data type), ο οποίος έχει μία στατική εφαρμογή των λειτουργιών όλων των στιγμιότυπων.
- **Ενθυλάκωση (Encapsulation)**
- **Πολυμορφισμός Υπο-τύπων (Subtype polymorphism)**
- **Κληρονομία Αντικειμένων (Object Inheritance)**
- **Ανοικτή Αναδρομή (Open Recursion):** μία ειδική μεταβλητή (μπορεί συντακτικά να είναι λέξη-κλειδί), η οποία συνήθως είναι η this ή self, επιτρέπει το σώμα μίας μεθόδου να επικαλεστεί ένα άλλο σώμα μεθόδου του ίδιου αντικειμένου.
- **Αφαίρεση (Abstraction)**

Οι πιο γνωστές γλώσσες προγραμματισμού, οι οποίες ακολουθούν το ύφος του αντικειμενοστρεφούς προγραμματισμού είναι οι C++, Objective-C, Smalltalk, Delphi, Java, JavaScript, C#, Perl, Python, Ruby και PHP.

5.1.1.2 Χαρακτηριστικά και Πλεονεκτήματα της Java

Η γλώσσα προγραμματισμού Java είναι μία από τους κύριους εκπροσώπους του ύφους του αντικειμενοστρεφούς προγραμματισμού.

Η Java έχει πληθώρα χαρακτηριστικών που την καθιστούν ιδανική για προγράμματα όπως αυτά τα οποία χρησιμοποιούνται για τη διαχείριση των νεφών. Μερικά από αυτά φαίνονται παρακάτω:

- Η Java είναι **απλή.** Αυτό συμβαίνει γιατί βασίζεται στη C++, οπότε είναι ευκολότερη στη μάθηση μετά από τη C++, ενώ έχουν αφαιρεθεί πολλά χαρακτηριστικά που προκαλούν σύγχυση και δε χρησιμοποιούνται συχνά (όπως πχ δείκτες, υπερφόρτωση τελεστών κλπ). Επίσης, δε χρειάζεται να διαγράφονται τα αντικείμενα που δεν έχουν αναφορές λόγω της αυτόματης συλλογής σκουπιδιών της Java (Automatic Garbage Collection).
- Είναι **αντικειμενοστρεφής**, συνεπώς έχει όλα τα χαρακτηριστικά που περιγράφηκαν στην προηγούμενη ενότητα.

- Έχει **ανεξαρτησία από την πλατφόρμα (Platform Independent).** Η πλατφόρμα είναι το περιβάλλον υλικού ή λογισμικού, στο οποίο εκτελείται το πρόγραμμα. Η πλατφόρμα της Java διαφέρει από τις περισσότερες άλλες πλατφόρμες με την έννοια ότι είναι μία πλατφόρμα βασιζόμενη σε λογισμικό, η οποία τρέχει πάνω σε άλλες βασιζόμενες σε υλικό. Έχει δύο μέρη: το *περιβάλλον χρόνου εκτέλεσης (runtime environment*) και *το API (Application Programming Interface)*. Ο κώδικας της Java μπορεί να εκτελεστεί σε πολλαπλές πλατφόρμες όπως πχ. τα Windows, το Linux, το Solaris της Sun κλπ. Μεταγλωττίζεται από το μεταγλωττιστή και μετατρέπεται σε bytecode. Το bytecode είναι ανεξάρτητο από την πλατφόρμα.
- Είναι **ασφαλής**, λόγω της απουσίας δεικτών, αλλά και του γεγονότος ότι τα προγράμματα εκτελούνται μέσα στο virtual machine sandbox. Επίσης, έχει κάποιες λειτουργίες με τις οποίες αυξάνεται η ασφάλειά της, ενώ την ασφάλεια μπορεί να την ενισχύσει κι ο ίδιος ο προγραμματιστής με δικές του μεθόδους.
- Η Java είναι **δυνατή.** Χρησιμοποιεί δυνατή διαχείριση μνήμης. Απουσιάζουν οι δείκτες, ενώ έχει αυτόματη συλλογή σκουπιδιών. Ακόμη, υπάρχει χειρισμός των εξαιρέσεων (exception handling) και μηχανισμός ελέγχου τύπων.
- Είναι **ουδέτερη ως προς την αρχιτεκτονική.** Δεν περιλαμβάνει χαρακτηριστικά εξαρτημένα, όπως για παράδειγμα το μέγεθος των πρωτόγονων τύπων είναι σταθερό.
- **Φορητότητα**, καθώς μπορεί να γίνει μεταφορά του java bytecode σε κάθε πλατφόρμα.
- **Υψηλή απόδοση,** αφού η Java είναι γρηγορότερη από τον παραδοσιακό τρόπο διερμηνείας, καθώς ο byte code είναι πιο «κοντά» στον τοπικό κώδικα αλλά πιο αργή, παρόλα αυτά, από μία μεταγλωττισμένη γλώσσα.
- Η Java είναι **κατανεμημένη.** Μπορούμε να δημιουργήσουμε κατανεμημένες εφαρμογές, ενώ μπορούμε να έχουμε πρόσβαση σε αρχεία, καλώντας μεθόδους από κάθε μηχάνημα στο διαδίκτυο.
- Είναι γλώσσα **πολύ-νηματική (multi-threaded)**. Ένα νήμα είναι σαν ένα ξεχωριστό πρόγραμμα, το οποίο εκτελείται ταυτόχρονα. Μπορούμε να γράψουμε προγράμματα Java, τα οποία αντιμετωπίζουν πολλές εργασίες, καθορίζοντας πολλαπλά νήματα. Το κύριο πλεονέκτημα του πολύ-νηματισμού είναι ότι μοιράζει τη μνήμη. Τα νήματα είναι πολύ σημαντικά για πολλά είδη εφαρμογών.

5.1.1.3 Χρήση της Java στην παρούσα εργασία

Η Java στην εργασία χρησιμοποιήθηκε ουσιαστικά παντού. Είναι η γλώσσα στην οποία ήταν ο κώδικας του YCSB. Μέσω της abstract class του DB, την οποία κάναμε implement για τη MongoDB και την MongoMVCCDB, μέσω override των μεθόδων για τις βασικές λειτουργίες, όπως αναφέρθηκε παραπάνω (read(), insert(), update(), scan()).

Η Java χρησιμοποιήθηκε ακόμη στον MongoMVCC-Driver και στην επέκταση της MongoDB σε MongoDB MVCC, όπως και στον Transactional Manager.

5.1.2 Άλλες Τεχνολογίες που Χρησιμοποιήθηκαν

5.1.2.1 Okeanos Service

Το ~okeanos είναι μία υπηρεσία Υποδομής – ως - Υπηρεσία (IaaS). Αυτό σημαίνει ότι μπορεί κάποιος να χτίσει το δικό του υπολογιστή, ο οποίος να είναι μόνιμα συνδεδεμένος στο διαδίκτυο, χωρίς να ανησυχεί για σφάλματα υλικού, μπλεγμένα καλώδια, προβλήματα στη συνδεσιμότητα και στο υλικό. Έτσι, μπορεί να φτιάξει κάποιος τα δικά του Εικονικά Μηχανήματα (Virtual Machines - VMs) και Εικονικά Δίκτυα (Virtual Networks), τα οποία μπορεί να διαχειριστεί, να καταστρέψει, να συνδεθεί σε αυτά και διάφορες άλλες ενέργειες, τόσο μέσω του περιηγητή του, όσο και μέσω του REST API του ~okeanos. Ακόμη, διαθέτει και αποθηκευτικό χώρο, για την αποθήκευση αρχείων μέσω διαδικτύου, τη διαμοίρασή τους με άλλους και πρόσβαση σε αυτά, οποτεδήποτε και από οπουδήποτε, είτε και μέσω των VMs.

Το ~okeanos σχεδιάστηκε και αναπτύχθηκε από το Εθνικό Δίκτυο Έρευνας & Τεχνολογίας (Greek Research and Technology Network – GRNET). Το GRNET τρέχει την υπηρεσία του ~okeanos από τα κέντρα δεδομένων του. Παρέχει ποιοτικό IaaS στην ελληνική Ακαδημαϊκή και Ερευνητική Κοινότητα. Φοιτητές, καθηγητές και ερευνητές μπορούν να το χρησιμοποιήσουν δωρεάν και να αποκτήσουν την πλήρη ισχύ των εικονικών υποδομών (υπολογιστική ισχύ, δίκτυο, αποθηκευτικό χώρο).

Σε ένα φοιτητή δίνει τη δυνατότητα να δοκιμάσει διαφορετικά είδη λογισμικού σε ένα μηχάνημα που δε χρειάζεται όταν πλέον τελειώσει τη δουλειά του. Επίσης, καθιστά εύκολη την επαφή με διαφορετικές τεχνολογίες χωρίς να τον επιβαρύνει οικονομικά. Είναι σαν ένα ζωντανό πεδίο δοκιμών για τις φοιτητικές ανάγκες.

Στη δική μας περίπτωση συνδεόμαστε μέσω SSH (όπως θα παρουσιαστεί παρακάτω) με τα VMs που χρησιμοποιούμε από τον Okeanos. Η επαλήθευση του χρήστη γίνεται μέσω ενός αρχείου προσωπικού κλειδιού (ppk – PuTTY Private Key).

Τα VMs που χρησιμοποιούνται έχουν διευθύνσεις IP:

- 83.212.84.241 (Okeanos1)
- 83.212.84.242 (Okeanos2)
- 83.212.84.243 (Okeanos3)
- 83.212.84.244 (Okeanos4)

Το κάθε μηχάνημα έχει τη MongoDB εγκατεστημένη και σε λειτουργία (up) και ακούει στη θύρα 27017, ενώ μέσω SCP φορτώσαμε το μεταγλωττισμένο YCSB με την επέκταση για τη MongoDB και τη MongoDB MVCC.

Συνήθως, κατά τις μετρήσεις χρησιμοποιήθηκε το μηχάνημα Okeanos1 για την εκτέλεση του YCSB και το μηχάνημα Okeanos2 για τη φιλοξενία (host) της βάσης δεδομένων, στην οποία έγιναν οι μετρήσεις.

Αξίζει να σημειωθεί πως ο Transactional Manager της MVCC έκδοσης της βάσης φιλοξενείται στο μηχάνημα Okeanos4.

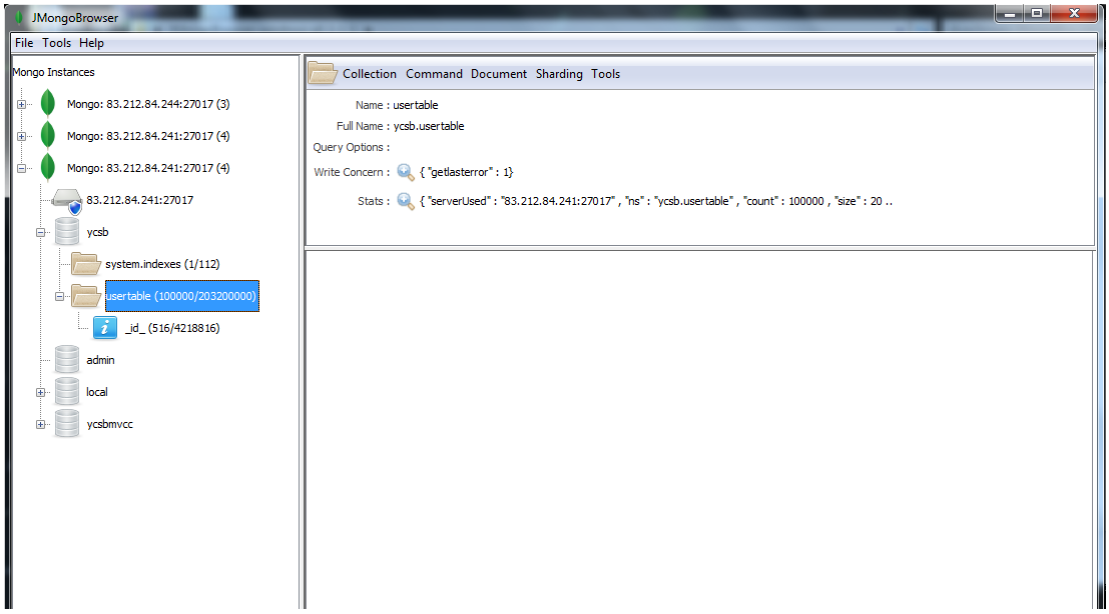
5.1.2.2 JMongoBrowser

Το JMongoBrowser είναι μία εφαρμογή Γραφικής Διεπαφής Χρήστη (GUI) που μπορεί να δείξει τα περιεχόμενα, να κάνει ερωτήματα και να διαχειριστεί ένα σύμπλεγμα MongoDB.

Για την εργασία αυτή, χρησιμοποιείται η έκδοση του JMongoBrowser για Windows. Κατά την εκκίνηση του προγράμματος, συνδεόμαστε στη βάση του μηχανήματος, στο οποίο θα φορτώσουμε ή έχουμε φορτώσει τη βάση δεδομένων του workload.

Συγκεκριμένα, το JMongoBrowser χρησιμοποιείται για τον έλεγχο των δεδομένων στη βάση (αν όλα εισήχθησαν σωστά για παράδειγμα), για ερωτήματα ώστε να διαπιστωθεί η αποτελεσματικότητα του εκάστοτε πειράματος, αλλά κυρίως για τη διαγραφή της κάθε βάσης δεδομένων, προτού περάσουμε στο επόμενο πείραμα. Αυτό γίνεται χρησιμοποιώντας την επιλογή του προγράμματος «Drop Database», στη βάση που δημιουργήσαμε.

Αξίζει να σημειωθεί πως η εφαρμογή αυτή μπορεί να συνδεθεί ταυτόχρονα σε όλα τα μηχανήματα που χρησιμοποιούνται.

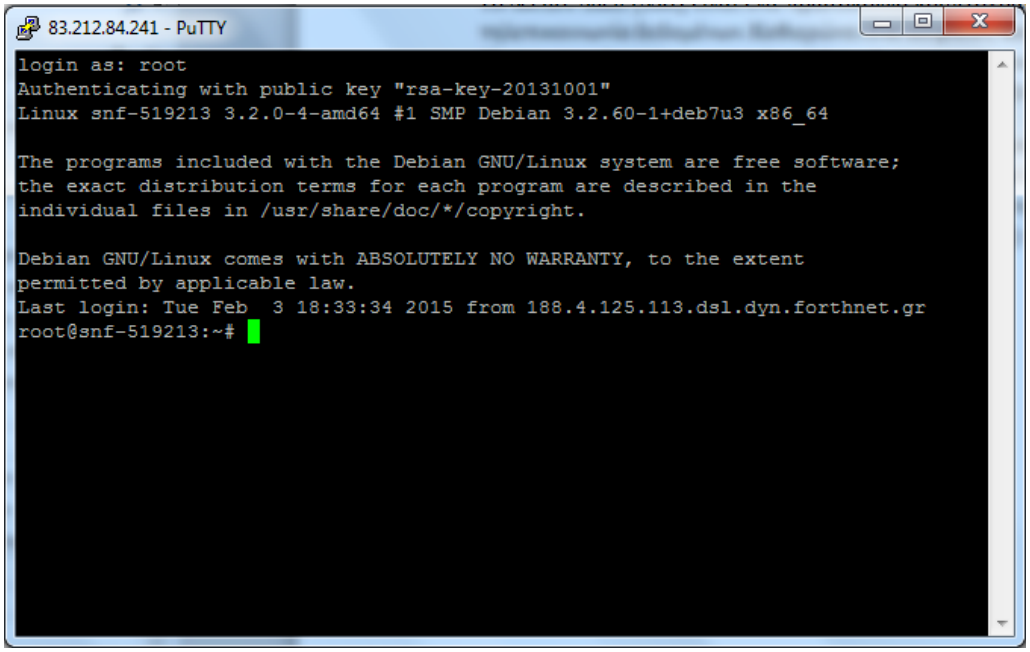


Εικόνα 28: Παράδειγμα χρήσης του JMongoBrowser, όπου φαίνονται οι βάσεις του μηχανήματος Okeanos1, ενώ είναι συνδεδεμένο και με τα Okeanos4 και Okeanos2

5.1.2.3 Secure Shell (SSH)

Το Secure Shell (SSH) είναι ένα πρωτόκολλο κρυπτογραφίας δικτύων για την ασφαλή τηλεπικοινωνία δεδομένων. Καθιερώνει ένα ασφαλές κανάλι πάνω σε ένα μη ασφαλές δίκτυο σε μία αρχιτεκτονική πελάτη-εξυπηρετητή, συνδέοντας μία εφαρμογή SSH πελάτη (το PuTTY στην περίπτωσή μας) σε έναν SSH εξυπηρετητή.

Το PuTTY που χρησιμοποιείται στην παρούσα διπλωματική εργασία είναι το πρόγραμμα πελάτη που αναφέρθηκε. Με αυτό συνδεόμαστε στα VMs του Okeanos, χρησιμοποιώντας το πρωτόκολλο SSH και στη συνέχεια μπορούμε να χρησιμοποιήσουμε απομακρυσμένα το κέλυφος (shell) του linux συστήματος του VM, για την εκτέλεση των πειραμάτων.



Εικόνα 29: Σύνδεση μέσω SSH με το Putty στο μηχάνημα Okeanos1

Το Secure Copy (SCP) είναι ένα μέσο για την ασφαλή μεταφορά αρχείων μεταξύ ενός τοπικού host και ενός απομακρυσμένου ή μεταξύ δύο απομακρυσμένων hosts. Βασίζεται στο SSH πρωτόκολλο.

Για τη χρήση του SCP στην εργασία αυτή χρησιμοποιείται το πρόγραμμα WinSCP. Είναι ένα πρόγραμμα πελάτη ανοιχτού κώδικα για Windows, το οποίο χρησιμοποιήσαμε για τη μεταφορά του μεταγλωττισμένου (compiled) YCSB με τις επεκτάσεις του. Όταν απαιτούνταν αλλαγές στο YCSB, τότε μεταφέραμε στα απομακρυσμένα VMs τις νέες εκδόσεις.

5.1.3 Το Ολοκληρωμένο Περιβάλλον Προγραμματισμού Netbeans

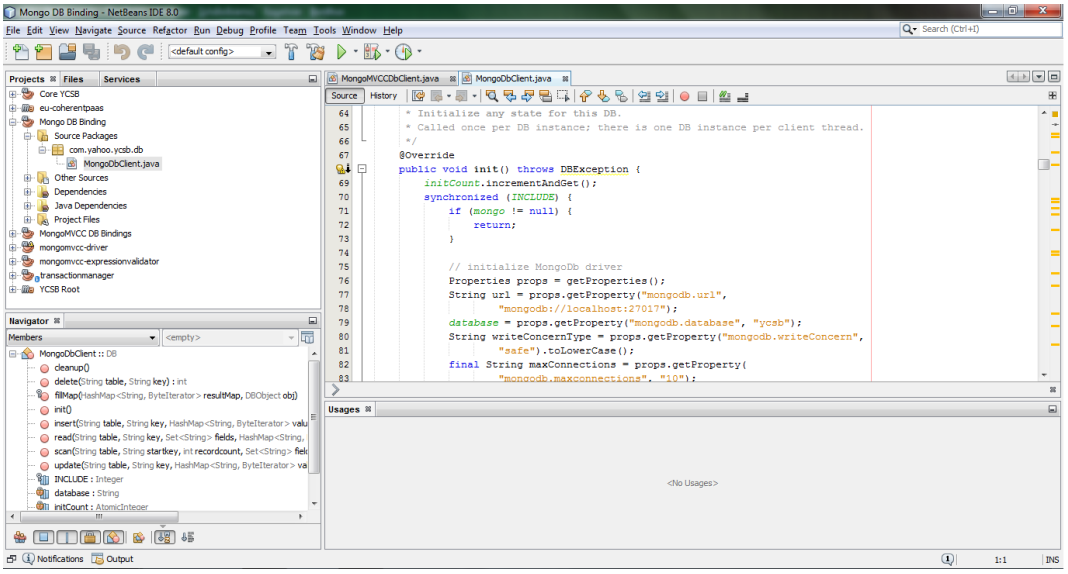
Το Netbeans είναι το επίσημο ολοκληρωμένο περιβάλλον προγραμματισμού (Integrated Development Environment – IDE) της Java 8. Με το πρόγραμμα επεξεργασίας κειμένου-κώδικα, τους αναλυτές κώδικα και τους μετατροπείς που προσφέρει, μπορεί κανείς να το χρησιμοποιήσει για την αναβάθμιση των εφαρμογών του.

Ένα IDE προσφέρει πολύ περισσότερα από έναν επεξεργαστή κειμένου. Ο επεξεργαστής του Netbeans προσφέρει αυτόματη μορφοποίηση στον κώδικα, ταιριάζει λέξεις, παρενθέσεις και άγκιστρα, και τονίζει τον πηγαίο κώδικα συντακτικά και σημασιολογικά. Παρέχει επίσης μοτίβα κώδικα, μικρές συμβουλές για την εγγραφή κώδικα και εργαλεία επανόρθωσης.

Η διατήρηση μίας καθαρής συνολικής όψης μεγάλων εφαρμογών, με μεγάλο αριθμό από φακέλους και αρχεία, και εκατομμύρια γραμμές κώδικα, είναι

δύσκολη υπόθεση. Το Netbeans όμως την καθιστά ευκολότερη, καθώς παρέχει διαφορετικούς τρόπους ως προς την όψη των δεδομένων, από πολλαπλά παράθυρα για ένα project μέχρι βοηθητικά εργαλεία για την εγκατάσταση και τη διαχείριση των εφαρμογών. Έτσι, όταν κάποιος νέος προγραμματιστής γίνει μέλος του project, μπορεί να καταλάβει αμέσως τη δομή της εφαρμογής, καθώς ο κώδικας είναι πολύ καλά δομημένος.

Το Netbeans παρέχει ακόμη το Netbeans GUI Builder, το ποίο αυτόματα αναλαμβάνει τη σωστή μορφοποίηση των κενών και της στοίχισης του κώδικα. Τέλος, προσφέρει ένα δυνατό Debugger που καθιστά ευκολότερη και πιο αποτελεσματική την αντιμετώπιση των διάφορων προβλημάτων και σφαλμάτων στον κώδικα.

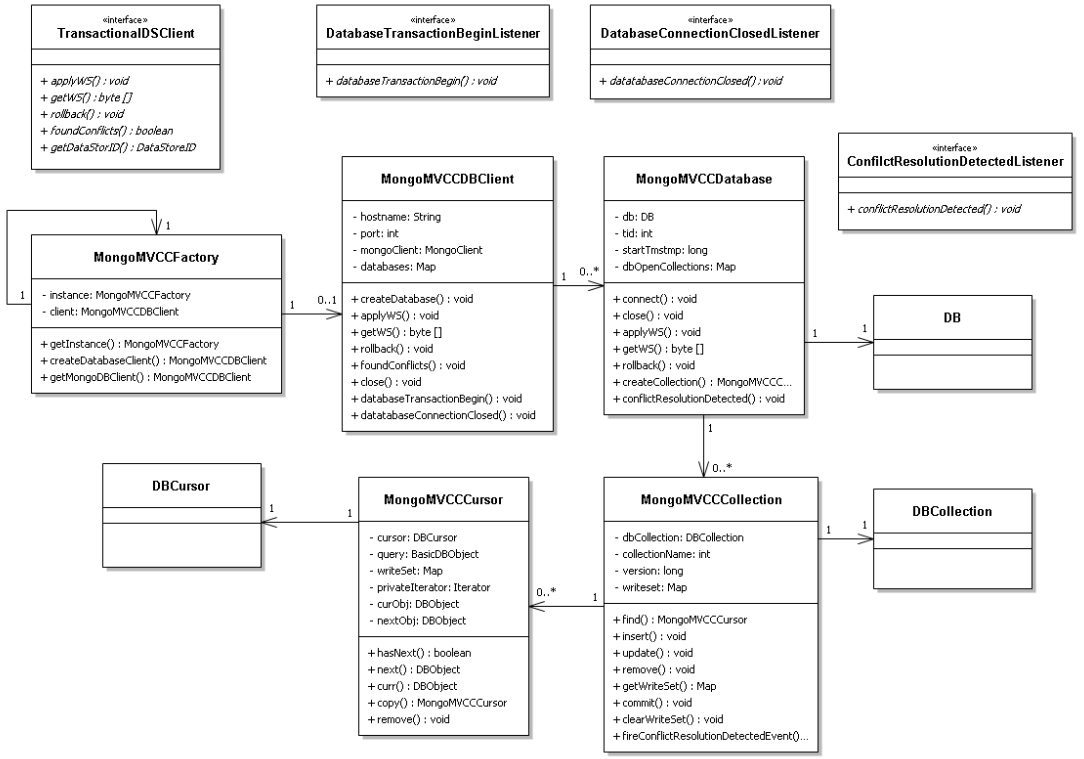


Εικόνα 30: Το περιβάλλον του Netbeans

5.2 Περιγραφή Κλάσεων

Η υλοποίηση της ολοκληρωμένης MongoDB αποτελείται από δύο κύρια μέρη: τον **MongoMVCC-Driver**, ο οποίος επεκτείνει τον επίσημο JDBC οδηγό της MongoDB, ώστε να μπορεί να χειριστεί λειτουργίες MVCC και ιδιωτικές εκδόσεις των συνδιαλλαγών, ενώ επίσης υλοποιεί τις υποχρεωτικές λειτουργίες για να υποστηρίζεται η αλληλεπίδραση με τον ολιστικό TM. Το δεύτερο μέρος είναι η **MongoDB-Ext** που επεκτείνει τη φυσική MongoDB και τις λειτουργίες της, με σκοπό την υποστήριξη της MVCC και την αποτελεσματική διαδικασία επαναφοράς σε περίπτωση αποτυχίας.

Το διάγραμμα κλάσεων του MongoMVCC-Driver φαίνεται παρακάτω [1]:



Εικόνα 31: Το Διάγραμμα Κλάσεων του MongoMVCC-Driver, σύμφωνα με τη UML

Στο διάγραμμα φαίνονται τα εξής βασικά μέρη-κλάσεις, τις οποίες χρησιμοποιεί ο κώδικας:

MongoMVCCFactory

Είναι το κύριο «εργοστάσιο» (factory) για την παραγωγή νέων στιγμιότυπων της κλάσης MongoMVCCDBClient. Πρόκειται για μία κλάση Singleton (δηλαδή μία κλάση με ένα μοναδικό αντικείμενο, με σκοπό τον έλεγχο της δημιουργίας αντικειμένων), την οποία ο προγραμματιστής θα πρέπει να χρησιμοποιήσει για να αρχικοποιήσει τον οδηγό της Mongo. Παρέχεις τις εξής μεθόδους [1]:

- **getInstance**: Αρχικοποιεί το εργοστάσιο και επιστρέφει το στιγμιότυπό του.
- **createDatabaseClient:** Αρχικοποιεί ένα αντικείμενο της κλάσης MongoMVCCDBClient, το οποίο είναι το πρόγραμμα-πελάτη JDBC που θα χρησιμοποιείται για την εκτέλεση λειτουργιών μέσα στη βάση.
- **getMongoDBClient:** Επιστρέφει το στιγμιότυπο του MongoMVCCDBClient.

MongoMVCCDBClient

Η κλάση αυτή προτείνεται να χρησιμοποιείται ως Singleton μέσα στο JVM (Java Virtual Machine) που εκτελεί την εφαρμογή. Αποθηκεύει όλες τις λεπτομέρειες σχετικά με τη φιλοξενία της MongoDB (MongoDB' s host – για παράδειγμα το όνομα του μηχανήματος φιλοξενίας – hostname, τη θύρα – port κλπ) και αρχικοποιεί ένα στιγμιότυπο του MongoClient της MongoDB. Αυτός χειρίζεται όλες τις συνδέσεις στη βάση δεδομένων. Διατηρεί και διαχειρίζεται μία επαρκή και ρυθμιζόμενη ποσότητα από συνδέσεις στη βάση και τις αναθέτει όπως απαιτείται από κάποιο νήμα. Η MongoDB προτείνει να χρησιμοποιείται μόνο ένα στιγμιότυπο αυτής της κλάσης, ακόμη και σε ένα πολύνηματικό περιβάλλον, ώστε να μειώνεται ο αριθμός των ενεργών συνδέσεων που θα πρέπει να διαχειριστεί ένα αντικείμενο της κλάσης MongoClient. Αυτός είναι ο λόγος που ο MongoDBClient πρέπει να δημιουργείται μόνο μία φορά από την κλάση-εργοστάσιο του.

Υλοποιεί επίσης τη διεπαφή (interface) TransactionalDSClient, ώστε να μπορεί ο τοπικός TM να τον επικαλείται για την εκτέλεση των απαραίτητων ενεργειών ως προς τη διαχείριση ταυτόχρονων συνδιαλλαγών. Υλοποιεί ακόμη τις DatabaseConnectionClosedListener και DatabaseTransactionBeginListener διεπαφές, οι οποίες επιτρέπουν τη διαχείριση πολλαπλών ταυτόχρονων συνδιαλλαγών σε ένα πολύνηματικό περιβάλλον. Οι μέθοδοι που υλοποιούνται είναι [1]:

- **createDatabase:** Δημιουργεί ένα νέο στιγμιότυπο της κλάσης MongoMVCCDatabase και το προσθέτει σε ένα προσωπικό χάρτη κατακερματισμού (hashmap).
- **applyWS:** Σε αυτήν τη μέθοδο γίνεται επίκληση όταν μία συνδιαλλαγή εισέρχεται στη φάση της οριστικοποίησης του commit. Όταν ο τοπικός TM δίνει την εντολή, ο MongoMVCCDBClient παίρνει το TID (Transaction ID – Αναγνωριστικό Συνδιαλλαγής) από το πλαίσιο της τρέχουσας συνδιαλλαγής και επιβάλλει στο αντίστοιχο αντικείμενο MongoMVCCDatabase να εφαρμόσει το ιδιωτικό της write-set στη MongoDB.
- **getWS:** Γίνεται επίκληση σε αυτήν τη μέθοδο όταν μία συνδιαλλαγή επιχειρεί να κάνει commit. Όταν ο τοπικός TM δώσει την εντολή, ο MongoMVCCDBClient παίρνει το TID της τρέχουσας συνδιαλλαγής και επιβάλλει στο αντίστοιχο αντικείμενο MongoMVCCDatabase να

επιστρέψει την αντίστοιχη αναπαράσταση του ιδιωτικού write-set της σε έναν πίνακα από bytes.

- **rollback:** Είναι η μέθοδος την οποία επικαλούμαστε, όταν μία συνδιαλλαγή πρέπει να αναιρέσει μερικές ενέργειές της και να επιστρέψει σε μία προηγούμενη κατάστασή της (rollback). Όταν ο τοπικός TM δώσει την εντολή, ο MongoMVCCDBClient παίρνει το TID από την τρέχουσα συνδιαλλαγή και επιβάλλει στο αντίστοιχο αντικείμενο MongoMVCCDatabase να κάνει rollback.
- **foundConflicts:** Αυτή η μέθοδος επιστρέφει πάντα false, καθώς η MongoDB δεν ελέγχει εσωτερικά αν υπάρχουν συγκρούσεις. Αυτή τη δουλειά την έχει αναλάβει ο TM.
- **close:** Η συγκεκριμένη μέθοδος κλείνει τον MongoClient και διαγράφει όλα τα αποθηκευμένα στιγμιότυπα MongoMVCCDBClient.
- **databaseTransactionBegin:** Όταν ένα στιγμιότυπο MongoMVCCDatabase ανοίγει μία νέα συνδιαλλαγή, προκαλεί ένα αντίστοιχο γεγονός (event), το οποίο οδηγεί στην επίκληση της μεθόδου αυτής. Ο MongoMVCCDBClient αποθηκεύει τη νέα συνδιαλλαγή που άνοιξε στην ιδιωτική του λίστα από ανοιχτές συνδιαλλαγές.
- **databaseConnectionClosed:** Όταν ένα στιγμιότυπο της κλάσης MongoMVCCDatabase κλείνει, τότε προκαλεί ένα αντίστοιχο γεγονός (event) που οδηγεί στην επίκληση αυτής της μεθόδου. Τότε ο MongoMVCCDBClient αφαιρεί όλες τις ανοιχτές συνδιαλλαγές αυτού του στιγμιότυπου από την ιδιωτική του λίστα με τις συνδιαλλαγές.

MongoMVCCDatabase

Αυτή η κλάση επεκτείνει (extends) την κλάση DB της MongoDB, ώστε να μπορεί να διαχειρίζεται συνδιαλλαγές και αντικείμενα δεδομένων πολλαπλών εκδόσεων. Αντιπροσωπεύει μία συνδιαλλαγή στη MongoDB. Αν πολλαπλές συνδιαλλαγές πρέπει να ανοίξουν από την εφαρμογή στην ίδια βάση δεδομένων, τότε πολλαπλά στιγμιότυπα της κλάσης πρέπει να δημιουργηθούν από τον MongoMVCCDBClient. Υλοποιεί επίσης τη διεπαφή ConflictResolutionDetectedListener. Οι κύριες μέθοδοι που παρέχει η MongoMVCCDatabase είναι [1]:

- **connect:** Όταν ένα αντικείμενο αυτής της κλάσης δεχθεί την εντολή να συνδεθεί, τότε ανακτά το πλαίσιο της αντίστοιχης συνδιαλλαγής (πχ το TID και τη στάμπα χρονοσήμανσης εκκίνησης) και το αποθηκεύει εσωτερικά. Τότε πυροδοτεί ένα αντίστοιχο γεγονός (event) ώστε το στιγμιότυπο του MongoMVCCDBClient μπορεί να πληροφορηθεί για το χειρισμό της νέας συνδιαλλαγής.
- **close:** Κλείνει όλα τα αντικείμενα της κλάσης MongoMVCCCollection που προηγουμένως είχαν δημιουργηθεί και πυροδοτεί ένα αντίστοιχο γεγονός ώστε το στιγμιότυπο του MongoMVCCDBClient να μπορεί να

πληροφορηθεί για να το αφαιρέσει από την ιδιωτική του λίστα συνδιαλλαγών.

- **applyWS:** Όταν επικαλείται αυτή η μέθοδος για να εφαρμόσει το ιδιωτικό της write-set, πρώτα ανακτά τη στάμπα χρονοσήμανσης του commit της τρέχουσας συνδιαλλαγής και έπειτα, επιβάλλει σε κάθε στιγμιότυπο της κλάσης MongoMVCCCollection στις οποίες είχε πρόσβαση να κάνει commit.
- **getWS:** Επιστρέφει την αναπαράσταση του ιδιωτικού write-set της συνδιαλλαγής σε έναν πίνακα από bytes.
- **rollback:** Όταν δίνονται εντολές για rollback, τότε αυτή η μέθοδος επιβάλλει κάθε στιγμιότυπο της MongoMVCCCollection να αποσύρει το ιδιωτικό του write-set και να κλείσει.
- **createCollection:** Δημιουργεί ένα νέο αντικείμενο της κλάσης MongoMVCCCollection και το αποθηκεύει σε μία ιδιωτική εσωτερική λίστα.
- **conflictResolutionDetected:** Αν ένα γεγονός τύπου ConflictResolutionDetected πυροδοτηθεί από ένα αντικείμενο της κλάσης MongoMVCCCollection, τότε το MongoMVCCDatabase πρέπει να κάνει rollback.

MongoMVCCCollection

Αυτή η κλάση επεκτείνει την κλάση DBCollection της MongoDB. Παρέχει τις ίδιες λειτουργίες με την κλάση DBCollection και μερικές επιπλέον μεθόδους, οι οποίες στοχεύουν στη διαχείριση συνδιαλλαγών (για παράδειγμα να εκτελούν rollback). Επίσης, αποθηκεύει εσωτερικά τις ιδιωτικές εκδόσεις μίας συνδιαλλαγής σε μία συλλογή πίνακα κατακερματισμού και τη στάμπα χρονοσήμανσης εκκίνησης της συνδιαλλαγής που δημιουργείται από αυτό το αντικείμενο. Οι κύριες μέθοδοι που παρέχονται από αυτήν την κλάση είναι [1]:

- **find:** Κάνει override τη μέθοδο find της DBCollection, προσθέτοντας την υποβολή της στάμπας χρονοσήμανσης εκκίνησης της τρέχουσας συνδιαλλαγής, ώστε να μπορούν να ανακτηθούν οι αντίστοιχες έγκυρες εκδόσεις που ικανοποιούν τις συνθήκες του ερωτήματος. Επιστρέφει ένα αντικείμενο τύπου MongoMVCCCursor, ώστε να μπορεί να διατρέξει αυξανόμενο το σύνολο αποτελεσμάτων.
- **insert:** Κάνει override τη μέθοδο insert της DBCollection. Αντί να αποθηκεύει τα νέα αντικείμενα δεδομένων που δημιουργήθηκαν κατευθείαν στον αποθηκευτικό χώρο της MongoDB, τα προσθέτει στο ιδιωτικό write-set της συνδιαλλαγής.
- **update:** Κάνει override τη μέθοδο update της DBCollection. Αντί να ενημερώνει απευθείας τα αντικείμενα δεδομένων που αντιστοιχούν στο ερώτημα που υποβάλλεται, δημιουργεί νέες εκδόσεις και τα αποθηκεύει στο ιδιωτικό write-set του αντικειμένου.

- **remove:** Κάνει override τη μέθοδο delete της DBCollection. Αντί να διαγράψει απευθείας τα αντικείμενα δεδομένων που αντιστοιχούν στο ερώτημα, δημιουργεί νέες εκδόσεις, καθιστά τη σημαία τους isDeleted αληθή και τις προσθέτει στο ιδιωτικό write-set.
- **commit:** Σπρώχνει τα στοιχεία του ιδιωτικού write-set του αντικειμένου στη MongoDB. Αν τα αντικείμενα είναι νέα, τα προσθέτει απλώς στο χώρο αποθήκευσης θέτοντας τη στάμπα χρονοσήμανσης του commit τους στην αντίστοιχη της συνδιαλλαγής. Αν υπάρχουν νέες εκδόσεις ήδη υπαρχόντων αντικειμένων δεδομένων, τότε πρώτα ενημερώνει τη στάμπα χρονοσήμανσης του commit της προηγούμενης έκδοσης της συνδιαλλαγής και μετά τα προσθέτει στο χώρο αποθήκευσης των δεδομένων.
- **getWrite-set:** Επιστρέφει το ιδιωτικό write-set του αντικειμένου.
- **clearWrite-set:** Αποσύρει το ιδιωτικό write-set του αντικειμένου. Γίνεται επίκληση σε αυτήν τη μέθοδο, κυρίως όταν μία συνδιαλλαγή θα πρέπει να κάνει rollback.
- **fireConflictResolutionDetectedEvent:** Αν γίνει άρση μίας εξαίρεσης TransactionManagerException από τον τοπικό TM, κατά την ενημέρωση ή διαγραφή ενός αντικειμένου, τότε γίνεται επίκληση σε αυτήν τη μέθοδο, ώστε να πυροδοτηθεί το αντίστοιχο γεγονός που θα ενημερώσει τη MongoMVCCDatabase για να κάνει rollback.

MongoMVCCCursor

Αυτή η κλάση κάνει επέκταση της κλάσης DBCursor της MongoDB. Αυτή επιστρέφεται όταν μία DBCollection επικαλείται τη μέθοδο find(). Διατρέχει αυξανόμενος το σύνολο αποτελεσμάτων ενός ερωτήματος και επιστρέφει τα ανακτημένα έγγραφα σε μορφή DBObject της MongoDB. Ο MongoMVCCCursor κάνει override αυτή τη μέθοδο διατρέχοντας πρώτα μέσω του ιδιωτικού write-set της συλλογής, και μετά μέσω του συνόλου αποτελεσμάτων του υποβληθέντος ερωτήματος, ενώ ελέγχει αν τα ανακτώμενα έγγραφα είναι απηρχαιωμένα στα πλαίσια της τρέχουσας συνδιαλλαγής. Χρησιμοποιεί, εσωτερικά, δύο δείκτες που δείχνουν στο τρέχον και στο επόμενο αντικείμενο του συνόλου αποτελεσμάτων. Οι μέθοδοι που παρέχει είναι [1]:

- **hasNext:** Απαντά στην ερώτηση αν υπάρχει ή όχι ένα νέο αντικείμενο στο σύνολο αποτελεσμάτων. Αν ο δείκτης nextObj δείχνει ήδη σε κάποιο αντικείμενο, τότε επιστρέφει αληθές (true). Αλλιώς, ελέγχει αν ένα νέο αντικείμενο υπάρχει ή όχι στο σύνολο αποτελεσμάτων, και αν ναι θέτει το δείκτη nextObj να το δείχνει.
- **next:** Επιστρέφει το αντικείμενο στο οποίο δείχνει ο δείκτης nextObj. Στη συνέχεια, θέτει το δείκτη currObj σε αυτό το αντικείμενο και επίσης θέτει το δείκτη nextObj να δείχνει *null*. Αν ο τελευταίος ήδη έδειχνε *null*, τότε πρώτα επικαλείται την hasNext και στη συνέχεια επιστρέφει την αντίστοιχη τιμή.

- **curr:** Επιστρέφει το αντικείμενο στο οποίο δείχνει ο δείκτης currObj (τρέχον αντικείμενο)
- **copy:** Δημιουργεί ένα ακριβές αντίτυπο του αντικειμένου MongoMVCCCursor.
- **remove**: Αφαιρεί το τρέχον αντικείμενο, στο οποίο δείχνει ο δείκτης currObj.

6

Έλεγχος

Ένα σύστημα σαν αυτό της Mongo MVCC με τον ολιστικό TM, προσδίδει στη βάση μας όλα τα πλεονεκτήματα των ACID ιδιοτήτων, αλλά με τίμημα τις επιδόσεις του συστήματος, οι οποίες δεν μπορούν προφανώς να ανταγωνιστούν τις αντίστοιχες ενός φυσικού συστήματος MongoDB.

Το ζήτημα σε αυτήν την περίπτωση είναι πόσο αξίζει μία τέτοια υλοποίηση. Δηλαδή, πόσο σημαντικά είναι τα πλεονεκτήματα σε σχέση με τα μειονεκτήματα. Ο βέλτιστος τρόπος για να διαπιστωθεί αυτό είναι πειραματικές μετρήσεις στη βάση, με κατάλληλα διαμορφωμένα workloads (φόρτο εργασίας), τα οποία μπορούν να εμφανίσουν τα πιο κρίσιμα σημεία και προβλήματα ενός συστήματος, σε συνδυασμό βέβαια με ένα πρόγραμμα-εργαλείο για μετρήσεις (benchmark).

Στην περίπτωση του συστήματος του CoherentPaaS, χρησιμοποιήσαμε το εργαλείο της Yahoo, YCSB, αφού πρώτα το επεκτείναμε ώστε να υποστηρίζει τη MongoDB και την έκδοση με MVCC. Το εργαλείο αυτό παρουσιάζεται στο πρώτο μέρος αυτού του κεφαλαίου, με το δεύτερο μέρος να αποτελεί τα αποτελέσματα των πειραμάτων και η παρουσίασή τους.

6.1 Μεθοδολογία ελέγχου: Yahoo! Cloud Serving Benchmark

Τα τελευταία χρόνια έχει υπάρξει μία έκρηξη νέων συστημάτων για την αποθήκευση δεδομένων και τη διαχείριση των υπολογιστικών νεφών (clouds). Πολλά από αυτά τα συστήματα «νέφους» αποκαλούνται και «χώροι αποθήκευσης key-value» και «συστήματα NoSQL» και μοιράζονται όλα το στόχο της μαζικής κλιμάκωσης «κατ' αίτηση» (ελαστικότητα) και τον απλοποιημένο προγραμματισμό εφαρμογών και υλοποίησης.

Η μεγάλη ποικιλία έχει καταστήσει την επιλογή του κατάλληλου συστήματος δύσκολη υπόθεση. Οι μεγαλύτερες διαφορές είναι μεταξύ των διαφόρων μοντέλων δεδομένων, όπως είδαμε στο κεφάλαιο 2. Παρόλα αυτά τα μοντέλα δεδομένων μπορεί να αρχειοποιηθούν και να συγκριθούν ποιοτικά. Η σύγκριση των επιδόσεων διαφόρων συστημάτων είναι ένα δύσκολο πρόβλημα. Μερικά συστήματα έχουν σχεδιαστεί για τη βελτιστοποίηση των writes χρησιμοποιώντας δομές πάνω-στο-δίσκο (on-disk structures) που μπορεί να διατηρηθούν χρησιμοποιώντας ακολουθιακή I/O (Input/Output), ενώ άλλα συστήματα έχουν βελτιστοποιηθεί για τυχαία reads, χρησιμοποιώντας μία πιο παραδοσιακή αρχιτεκτονική με buffer-pool. Επιπλέον, αποφάσεις για τον καταμερισμό, την αντιγραφή, την τοποθέτηση των δεδομένων και τη συνέπεια των συνδιαλλαγών έχουν αντίκτυπο στις επιδόσεις.

Η κατανόηση των επιπλοκών των επιδόσεων, σε αυτές τις αποφάσεις, είναι αρκετά δύσκολη. Οι προγραμματιστές των συστημάτων αναφέρουν τους αριθμούς των επιδόσεων για τα βέλτιστα workloads (φορτία εργασίας) για το σύστημά τους, κάτι το οποίο δεν ταιριάζει το workload για το στόχο μίας εφαρμογής. Επομένως, οι προγραμματιστές πρέπει συχνά να κατεβάσουν και να εκτιμήσουν μόνοι τους πολλαπλά συστήματα. Έχουν υπάρξει πολλά παραδείγματα για τη Yahoo! και αυτή η διαδικασία είναι πολύ χρονοβόρα και ακριβή.

Ο στόχος της Yahoo!, όσον αφορά στο YCSB (Yahoo! Cloud Serving Benchmark), είναι η δημιουργία ενός τυποποιημένου εργαλείου δοκιμών (benchmark) και πλαισίου δοκιμών (benchmark framework) για τη διευκόλυνση της αξιολόγησης διαφορετικών συστημάτων νέφους. Επικεντρώθηκε στα συστήματα εξυπηρέτησης (serving systems), τα οποία παρέχουν online πρόσβαση read και write στα δεδομένα. Αυτό σημαίνει, ότι ένας χρήστης διαδικτύου περιμένει για μία σελίδα να φορτώσει, ενώ τα reads και writes στη βάση δεδομένων εκτελούνται διαρκώς, ως μέρος της κατασκευής και της διανομής της σελίδας.

Έτσι παρουσιάστηκε το YCSB πλαίσιο, το οποίο χρησιμοποιείται για το benchmarking βάσεων δεδομένων νέφους. Αποτελείται από ένα πρόγραμμα-πελάτη που παράγει workloads και ένα πακέτο από τυποποιημένα workloads που καλύπτουν σημαντικά μέρη του χώρου επιδόσεων. Ένα σημαντικό πλεονέκτημα του YCSB είναι η επεκτασιμότητά του. Η γεννήτρια των workloads καθιστά εύκολο τον καθορισμό νέων τύπων workloads και είναι, επίσης, πολύ εύκολη η προσαρμογή του προγράμματος πελάτη, ώστε να μπορεί να αξιολογήσει και άλλα συστήματα δεδομένων. [4]

Αυτό κάναμε στην περίπτωση της MongoDB και της MongoDB MVCC, οι οποίες δεν υποστηρίζονται εξ' ορισμού στο YCSB, αλλά των οποίων η αξιολόγηση καθίσταται δυνατή μέσω αλλαγών του YCSB για αυτές.

6.1.1 Τα Επίπεδα του Benchmark

Υπάρχουν δυο επίπεδα για την αξιολόγηση των επιδόσεων και της επεκτασιμότητας των συστημάτων εξυπηρέτησης νέφους.

Επίπεδο 1 - Επιδόσεις

Το επίπεδο επιδόσεων (performance tier) του benchmark επικεντρώνεται στην καθυστέρηση (latency) των αιτήσεων, όταν η βάση δεδομένων βρίσκεται υπό μεγάλο φορτίο. Η καθυστέρηση είναι πολύ σημαντική για τα συστήματα εξυπηρέτησης, καθώς υπάρχει συνήθως ένας άνθρωπος που περιμένει για μία σελίδα να φορτώσει. Παρόλα αυτά, υπάρχει μία εγγενής εξάρτηση των επιδόσεων μεταξύ καθυστέρησης και του throughput (δηλαδή των λειτουργιών που γίνονται ανά δεδομένη χρονική στιγμή). Σε ένα δεδομένο σύστημα, όσο η ποσότητα του φορτίου αυξάνεται, η καθυστέρηση των μεμονωμένων αιτήσεων αυξάνεται, αφού υπάρχει μεγαλύτερος ανταγωνισμός για το δίσκο, το CPU και το δίκτυο. Συνήθως, οι προγραμματιστές πρέπει να αποφασίσουν σε μία αποδεκτή καθυστέρηση και να παρέχουν αρκετούς εξυπηρετητές, ώστε να επιτευχθεί το επιθυμητό throughput χωρίς η καθυστέρηση να υπερβεί την αποδεκτή. Ένα σύστημα με καλύτερες επιδόσεις, το επιτυγχάνει αυτό με λιγότερους εξυπηρετητές.

Αυτό το επίπεδο στοχεύει στο χαρακτηρισμό αυτής ακριβώς της ανταλλαγής ώστε να επιτυγχάνεται ο βέλτιστος συγκερασμός μεταξύ throughput και καθυστέρησης. Για τη λειτουργία αυτού του επιπέδου χρειάζεται μία γεννήτρια φόρτων εργασίας (workload generator) που έχει δύο σκοπούς: τον καθορισμό του συνόλου δεδομένων και τη φόρτωσή του στη βάση δεδομένων, καθώς και την εκτέλεση λειτουργιών, μετρώντας παράλληλα τις επιδόσεις.

Επίπεδο 2 - Επεκτασιμότητα

Ένας σημαντικός παράγοντας των συστημάτων νέφους είναι η ικανότητά τους να κλιμακώνονται ελαστικά, ώστε να διαχειρίζονται περισσότερο ή λιγότερο φορτίο, ανάλογα με τη ζήτηση των εφαρμογών. Αυτό το επίπεδο εξετάζει τις επιπτώσεις της προσθήκης μηχανημάτων στο σύστημα πάνω στις επιδόσεις.

Η **επεκτασιμότητα προς τα πάνω** δείχνει το πώς αντιδρά η βάση δεδομένων στην αύξηση των μηχανημάτων. Για την εξέτασή της, φορτώνουμε ένα συγκεκριμένο αριθμό εξυπηρετητών με δεδομένα και τρέχουμε το φόρτο εργασίας. Στη συνέχεια, διαγράφουμε τα δεδομένα, προσθέτουμε εξυπηρετητές στο σύμπλεγμα και ξανατρέχουμε το φόρτο εργασίας. Αν η βάση δεδομένων έχει καλές ιδιότητες επεκτασιμότητας, οι επιδόσεις θα πρέπει να διατηρούνται σταθερές, και το throughput και η ποσότητα των δεδομένων να αυξάνονται ανάλογα.

Η **ελαστική αύξηση της ταχύτητας** δείχνει τις επιδόσεις της βάσης δεδομένων, όταν αυξάνονται τα μηχανήματα ενώ το σύστημα είναι ενεργό και τρέχει. Σε αυτήν την περίπτωση φορτώνουμε πάλι τα δεδομένα σε εξυπηρετητές και τρέχουμε το φόρτο εργασίας. Αλλά πλέον, προσθέτουμε εξυπηρετητές χωρίς να σταματήσουμε το φόρτο εργασίας, ενώ παρατηρούμε τις επιδράσεις στις επιδόσεις. [4]

6.1.2 Τα Workloads για το YCSB

Η Yahoo! ανέπτυξε ένα κεντρικό σύνολο από workloads για την αξιολόγηση διαφορετικών πλευρών των επιδόσεων ενός συστήματος, το οποίο αποκαλείται το Κεντρικό Πακέτο του YCSB (YCSB Core Package). Στο πλαίσιο εργασίας του YCSB, ένα πακέτο είναι μια συλλογή από συσχετισμένα workloads. Το κάθε workload αντιπροσωπεύει ένα συγκεκριμένο μίγμα από read και write λειτουργίες, μεγέθη δεδομένων, κατανομές αιτήσεων και άλλων. Μπορεί να χρησιμοποιηθεί για την εκτίμηση των συστημάτων σε ένα συγκεκριμένο σημείο στο χώρο επιδόσεων (performance space). Ένα πακέτο που συμπεριλαμβάνει πολλά workloads εξετάζει ένα μεγαλύτερο εύρος του χώρου αυτού. Παρότι το κεντρικό πακέτο εξετάζει πολλούς ενδιαφέροντες τομείς, ο χρήστης μπορεί να δημιουργήσει το δικό του workload, μεταβάλλοντας ένα σύνολο από παραμέτρους ή ακόμα και γράφοντας κώδικα σε Java.

6.1.2.1 Λειτουργία του κεντρικού πακέτου workloads

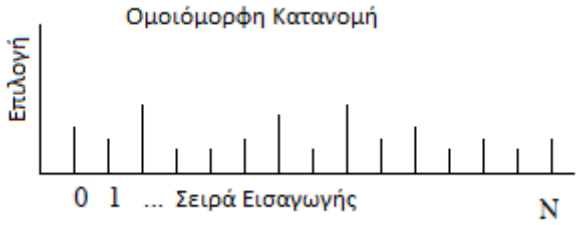
Στην εφαρμογή υπάρχει ένας πίνακας από εγγραφές, η κάθε μία εκ των οποίων έχει F πεδία. Κάθε εγγραφή αναγνωρίζεται από το πρωτεύον κλειδί της, το οποίο θα είναι μία συμβολοσειρά όπως «user515345». Το κάθε πεδίο ονομάζεται field0, field1 κοκ. Οι τιμές του κάθε πεδίου είναι μία τυχαία συμβολοσειρά από χαρακτήρες ASCII, μήκους L. Η κάθε λειτουργία στο χώρο αποθήκευσης επιλέγεται τυχαία να είναι μία εκ των [4]:

- **Insert:** εισαγωγή νέας εγγραφής
- **Update:** ενημέρωση μίας εγγραφής, αντικαθιστώντας την τιμή ενός πεδίου
- **Read:** ανάγνωση μίας εγγραφής, είτε επιλέγοντας τυχαία ένα πεδίο, είτε όλα τα πεδία της.
- **Scan:** Ανίχνευση των εγγραφών στη σειρά, αρχίζοντας από ένα τυχαία επιλεγμένο κλειδί. Ο αριθμός των εγγραφών που θα ανιχνευθούν είναι επίσης τυχαίος. Για την ανίχνευση, η κατανομή του μήκους των ανιχνεύσεων επιλέγεται ως μέρος του workload. Επομένως, η μέθοδος scan() παίρνει ένα αρχικό κλειδί και τον αριθμό των εγγραφών που θα ανιχνεύσει. Η παράμετρος του αριθμού των εγγραφών επιτρέπει τον καθορισμό συγκεκριμένων σημείων περάτωσης της ανίχνευσης.

6.1.2.2 Κατανομές των workloads

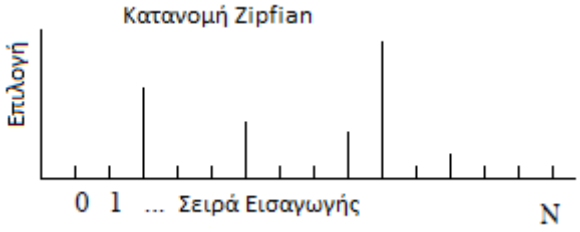
Το πρόγραμμα-πελάτη των workloads πρέπει να κάνει αρκετές τυχαίες επιλογές όταν παράγει φορτίο: ποια λειτουργία να εκτελέσει, ποια εγγραφή να κάνει read ή write, πόσες εγγραφές να ανιχνεύσει. Αυτές οι αποφάσεις κυβερνώνται από τυχαίες κατανομές. Οι χρησιμοποιούμενες στο YCSB είναι [4]:

- **Ομοιόμορφη (Uniform):** Επιλέγει ένα αντικείμενο, ομοιόμορφα, στην τύχη. Για παράδειγμα, όταν επιλέγει μία εγγραφή, όλες οι άλλες εγγραφές στη βάση δεδομένων έχουν την ίδια πιθανότητα να επιλεγούν.



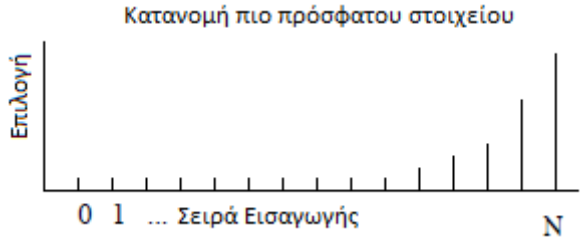
Εικόνα 32: Η ομοιόμορφη κατανομή

- **Zipfian:** Επιλέγει ένα αντικείμενο σύμφωνα με την κατανομή Zipfian. Για παράδειγμα, όταν επιλέγει μία εγγραφή, κάποιες εγγραφές θα είναι πολύ δημοφιλείς (η κεφαλή της κατανομής) και κάποιες θα είναι το αντίθετο (η ουρά της κατανομής).



Εικόνα 33: Η κατανομή Zipfian

- **Πιο πρόσφατου στοιχείου (Latest):** Είναι όπως η κατανομή Zipfian, αλλά οι εγγραφές που εισήχθησαν τελευταίες αποτελούν την κεφαλή της κατανομής.



Εικόνα 34: Η κατανομή του πιο πρόσφατου στοιχείου

- **Πολυωνυμική (Multinomial):** Οι πιθανότητες για το κάθε αντικείμενο μπορεί να καθοριστούν. Για παράδειγμα, μπορούμε να καθορίσουμε την πιθανότητα 0.95 για το read, 0.05 για την ενημέρωση και 0 για τις άλλες λειτουργίες.

6.1.2.3 Τα workloads που χρησιμοποιήθηκαν

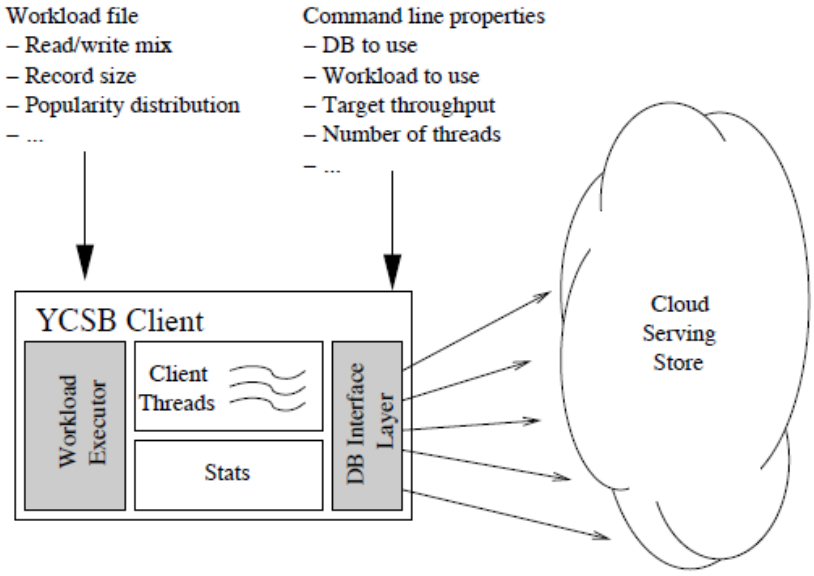
Στην παρούσα εργασία, για το YCSB χρησιμοποιήθηκαν τα workloads που προσέφερε η ίδια η Yahoo! στο κεντρικό πακέτο των workloads της. Η μόνη αλλαγή που έγινε ήταν στον αριθμό των εγγραφών και των λειτουργιών που θα εκτελούνταν, όπου βάλαμε 100,000 εγγραφές (αλλάζοντας την παράμετρο recordcount του κάθε workload) και 1,000,000 λειτουργίες (αλλάζοντας την παράμετρο operationcount του κάθε workload).

Τα workloads που χρησιμοποιήσαμε ήταν τα εξής:

I. **Workload B: Κυρίως read**
   Πρόκειται για ένα workload με 95/5 μίγμα reads/writes και για την ακρίβεια είχαμε παραμέτρους readproportion=0.95 και updateproportion=0.05 με κατανομή Zipfian. Παράδειγμα μίας τέτοιας εφαρμογής θα ήταν μία εφαρμογή για ετικέτες σε φωτογραφίες, όπου η προσθήκη μίας ετικέτας είναι ένα update, αλλά οι περισσότερες είναι ανάγνωση ετικετών , οπότε read.
II. **Workload C: Μόνο read**
   Αυτό το workload περιέχει μόνο read, δηλαδή με παράμετρο readproportion=1, ενώ είχαμε κατανομή Zipfian. Αυτή θα μπορούσε να είναι η αποθήκευση προφίλ χρηστών, όπου τα προφίλ έχουν δημιουργηθεί σε άλλη εφαρμογή.
III. **Workload D: Κυρίως read με δημοφιλές το πιο πρόσφατο**
   Εδώ έχουμε μίγμα 95/5 για read/insert, δηλαδή readproportion=0.95 και insertproportion=0.05 παραμέτρους, με κατανομή Latest. Αυτό σημαίνει ότι εισάγονται νέες εγγραφές και οι πιο πρόσφατες είναι οι πιο δημοφιλείς. Τέτοια εφαρμογή θα ήταν εφαρμογή όπου γίνονται ενημερώσεις κατάστασης των χρηστών, καθώς θέλουν να διαβάζουν τα τελευταία νέα.
IV. **Workload E: Μικρού εύρους scans**
   Σε αυτό το workload έχουμε μίγμα 95/5 scans/inserts, δηλαδή παραμέτρους για το workload scanproportion=0.95 και insertproportion=0.05, με κατανομή Zipfian. Γίνονται ερωτήματα μικρού εύρους , αντί για μεμονωμένες εγγραφές. Παράδειγμα τέτοιας εφαρμογής θα ήταν συνομιλίες που βασίζονται σε threads (όπως σε ένα forum), όπου το κάθε scan είναι για τις δημοσιεύσεις (posts) ενός νήματος (thread).

6.1.3 Η Αρχιτεκτονική του YCSB

Το πρόγραμμα-πελάτη (client) του YCSB είναι ένα πρόγραμμα Java για την παραγωγή δεδομένων που θα φορτωθούν στη βάση και την παραγωγή των λειτουργιών που θα αποτελέσουν το workload. Η αρχιτεκτονική του YCSB φαίνεται στην παρακάτω εικόνα:



Εικόνα 35: Η αρχιτεκτονική του YCSB προγράμματος-πελάτη, όπως συνδέεται με τη βάση δεδομένων νέφους [4]

Η κύρια λειτουργία είναι η οδήγηση πολλαπλών νημάτων πελάτη (client threads) από το πρόγραμμα-εκτελεστή του workload. Το κάθε νήμα (thread) εκτελεί μία ακολουθία από λειτουργίες, κάνοντας κλήσεις στο στρώμα διεπαφής της βάσης δεδομένων, ώστε να τη φορτώσει (load), αλλά και να εκτελέσει το workload (run). Τα νήματα ρυθμίζουν το ρυθμό που παράγουν αιτήσεις, ώστε να μπορούν να ελέγξουν άμεσα το προσφερόμενο φορτίο στη βάση δεδομένων. Ρυθμίζουν επίσης την καθυστέρηση και το throughput, το οποίο επιτυγχάνεται μέσω των λειτουργιών τους, και στη συνέχεια αναφέρουν τις μετρήσεις στη διεργασία των στατιστικών στοιχείων. Στο τέλος του πειράματος, η διεργασία αυτή προσφέρει τις μετρήσεις αθροιστικά και αναφέρει το μέσο όρο, την 95η και την 99η καθυστερήσεις σε εκατοστημόρια και ένα ιστόγραμμα ή σειρά χρόνων των μετρήσεων.

Το πρόγραμμα-πελάτη παίρνει μία σειρά από παραμέτρους που καθορίζουν τις λειτουργίες του. Αυτές χωρίζονται σε δύο ομάδες:

Ιδιότητες του Workload (Workload Properties)

Αυτές οι ιδιότητες καθορίζουν το workload, ανεξαρτήτως από τη βάση δεδομένων και το τρέξιμο του πειράματος. Είναι οι παράμετροι που αναφέρθηκαν και κατά την παρουσίαση των workloads, στο προηγούμενο μέρος του κεφαλαίου, για παράδειγμα το μίγμα read/write και η κατανομή που χρησιμοποιήθηκε.

Ιδιότητες κατά την Εκτέλεση (Runtime Properties)

Είναι οι συγκεκριμένες εκείνες ιδιότητες που απαιτούνται για το κάθε πείραμα. Για παράδειγμα, το στρώμα της διεπαφής της βάσης δεδομένων που χρησιμοποιείται (στην περίπτωσή μας MongoDB), ιδιότητες που αρχικοποιούν το στρώμα αυτό (όπως τα ονόματα των μηχανημάτων που φιλοξενούν τη βάση – hostnames) και ο αριθμός των νημάτων. [4]

6.1.4 Επέκταση του YCSB για τη MongoDB και τη MongoDB MVCC

6.1.4.1 Επεκτασιμότητα του YCSB

Ένας από τους κύριους σκοπούς του YCSB είναι η επεκτασιμότητά του. Για την ακρίβεια, ένας από τους σκοπούς των δημιουργών του ήταν το να καταστήσουν εύκολο να μετρούν οι προγραμματιστές μεγάλη ποικιλία συστημάτων νέφους εξυπηρέτησης.

Όπως φαίνεται και στην εικόνα 35, όπου φαίνεται η αρχιτεκτονική του YCSB, δύο είναι τα μέρη που μπορεί να αντικατασταθούν, ο Workload Executor (πρόγραμμα εκτέλεσης των workloads) και το DB Interface Layer (το στρώμα διεπαφής της βάσης δεδομένων). Ο Workload executor περιέχει τον κώδικα για την εκτέλεση της φάσης φόρτωσης και της φάσης συνδιαλλαγών του workload. Μπορεί να χρησιμοποιήσει τα workloads όπως αυτά παρουσιάστηκαν παραπάνω. [4] Το κομμάτι αυτό έμεινε ανέπαφο στη δική μας υλοποίηση.

Το στρώμα διεπαφής της βάσης δεδομένων μεταφράζει απλά αιτήματα (όπως read()) από νήματα του προγράμματος-πελάτη σε κλήσεις στη βάση δεδομένων. Η επιλογή αυτής της παραμέτρου γίνεται κατά την εντολή για το τρέξιμο του πειράματος, όπως είδαμε προηγουμένως, καθώς αυτές μπορούν να επιλεγούν δυναμικά.

Συγκεκριμένα, για να μπορέσει το πρόγραμμα-πελάτη του YCSB να χρησιμοποιηθεί ως benchmark εργαλείο για μία νέα βάση δεδομένων θα πρέπει να γραφεί μία νέα κλάση που να υλοποιεί τις μεθόδους [4]:

- **read()**
  διαβάζει μία εγγραφή από τη βάση δεδομένων και επιστρέφει ένα συγκεκριμένο πεδίο ή όλα τα πεδία.
- **insert()**
  εισάγει μία εγγραφή στη βάση δεδομένων.

- **update()**
  ενημερώνει μία εγγραφή στη βάση δεδομένων, προσθέτοντας ή αντικαθιστώντας τα καθορισμένα πεδία.
- **delete()**
  διαγράφει μία εγγραφή από τη βάση δεδομένων
- **scan()**
  εκτελεί μία ανίχνευση εύρους (range scan), διαβάζοντας έναν καθορισμένο αριθμό εγγραφών, ξεκινώντας από ένα δοσμένο κλειδί.

Η φυσική έκδοση του YCSB δεν υποστηρίζει τη MongoDB (και προφανώς ούτε τη MVCC). Οπότε, αντικαταστήσαμε το DB Interface Layer για να τις υποστηρίζει. Δηλαδή, υλοποιήσαμε τις μεθόδους αυτές, για την abstract class DB που ορίζεται στον κώδικα του YCSB, στα αρχεία MongoDBClient και MongoMVCCDbClient, κάνοντάς τες override.

6.2 Αναλυτική παρουσίαση ελέγχου

6.2.1 Χρήση του YCSB για τη MongoDB και τη MongoDB MVCC

Για τη χρήση του YCSB για την παρούσα εργασία και τις μετρήσεις για τη MongoDB και τη MongoDB MVCC, αφού η επέκταση του YCSB ήταν έτοιμη και είχε μεταγλωττιστεί (compiled) και ανέβει στα Virtual Machines (Εικονικά Μηχανήματα – VMs) που φιλοξενούσαν το YCSB και τις βάσεις μας (και θα περιγράψουμε σε επόμενο κεφάλαιο), χρησιμοποιήθηκαν οι παρακάτω εντολές.

Φόρτωση του workload στη βάση δεδομένων προς εξέταση

```
./ycsb load mongodb -P ../workloads/workloadc -s -threads 10 -p
mongodb.url=mongodb://83.212.84.242:27017 -p mongodb.database=ycsb > output.data
```

Αυτή είναι μία εντολή για τη φόρτωση του workload στη βάση δεδομένων μας. Για την ακρίβεια χρησιμοποιείται εδώ μέσω της παραμέτρου –P και του αρχείου που ακολουθεί (όπως είναι στο φάκελο workloads του παραπάνω directory και εδώ είναι το workloadc). Προφανώς αντικαθιστώντας το workloadc με workloadb, workloade, workloadd, μπορούμε να φορτώσουμε το αντίστοιχο workload στη βάση. Αντικαθιστώντας το load mongodb με load mongomvccdb, φορτώνουμε τη βάση για χρήση της MVCC έκδοσης.

Το –s σημαίνει ότι ζητάμε από το YCSB να δείχνει αναλυτικά στατιστικά κατά το τρέξιμό του.

Μέσω της σημαίας –threads *αριθμός_threads* ορίζουμε τον αριθμό των threads που θα χρησιμοποιηθούν για αυτήν τη φόρτωση. Το διατηρήσαμε σταθερό στα 10 threads για όλες τις μετρήσεις, αφού δεν επηρέασε τα αποτελέσματά μας. Με το –p mongodb.url= ορίστηκε το url της βάσης, η οποία είναι η mongodb, στην IP διεύθυνση 83.212.84.242 (είναι η διεύθυνση ενός από τα VMs που

αναφέραμε, με τα υπόλοιπα να τελειώνουν σε 241, 243 και 244 αντίστοιχα). Το 27017 που το συνοδεύει δηλώνει τη θύρα (port) που χρησιμοποιείται στο μηχάνημα για να ακούει η mongodb τα διάφορα requests.

Με την παράμετρο –p mongodb.database= ορίζεται η βάση δεδομένων στη mongodb του μηχανήματος που θα χρησιμοποιηθεί. Εδώ χρησιμοποιούμε τη βάση με όνομα ycsb (αν θέλουμε να φορτώσουμε για την έκδοση της MVCC, τότε χρησιμοποιούμε τη βάση με όνομα ycsbmvcc αντίστοιχα). Αν η βάση αυτή δεν υπάρχει, τότε δημιουργείται κατά το τρέξιμο της εντολής load.

Τέλος, με το > output.data, όπως ορίζεται από το linux, γράφονται τα μηνύματα εξόδου του συστήματος κατά τη φόρτωση, σε ένα αρχείο με όνομα output.data.

Εκτέλεση του workload για τη βάση δεδομένων που φορτώθηκε

```
./ycsb run mongodb -P ../workloads/workloadc -s -p
mongodb.url=mongodb://83.212.84.242:27017 -p mongodb.collection=usertable -p
mongodb.database=ycsb -target 1000000000 -p mongodb.maxconnections=10 -threads 1 >
output.data
```

Εδώ αντί για load έχουμε run, καθώς πρόκειται για την εκτέλεση (τρέξιμο) του ycsb και του workload. Όμοια με πριν, για την εκτέλεση της MVCC, θα αντικαθιστούσαμε το mongodb με mongomvccdb.

Το workload που χρησιμοποιείται ορίζεται με την παράμετρο –P ακριβώς όπως στην περίπτωση του load. Ακριβώς όπως στο load ορίζεται και η διεύθυνση της βάσης, με τις ίδιες διευθύνσεις μηχανημάτων (προφανώς θα χρησιμοποιήσουμε την ίδια διεύθυνση με το μηχάνημα στο οποίο φορτώθηκε πριν η βάση).

Στη συνέχεια ακολουθεί η παράμετρος –p mongodb.collection=usertable, η οποία δείχνει ότι χρησιμοποιείται η συλλογή της βάσης με όνομα usertable (αυτή είναι η default συλλογή, στην οποία φορτώθηκαν τα δεδομένα από το workload με την προηγούμενη εντολή).

Η επόμενη παράμετρος –p mongodb.database ορίζει όπως και πριν το όνομα της βάσης που χρησιμοποιείται, από την οποία αντλούνται οι εγγραφές για το πείραμα (προφανώς το όνομα θα πρέπει να είναι ίδιο με αυτό της βάσης που χρησιμοποιήθηκε κατά τη φόρτωση).

Με τη σημαία –target *αριθμός_target* προσδιορίζεται το ανώτατο όριο του συνολικού throughput (όλων των νημάτων μαζί) που θέλουμε να επιτύχουμε. Αν δεν είναι δυνατό να επιτευχθεί αυτό το όριο, τότε η εκτέλεση φτάνει στο ανώτατο δυνατό throughput που μπορεί να επιτύχει.

Με την παράμετρο –p mongodb.maxconnections= ορίζουμε τον αριθμό των μέγιστων ταυτόχρονων συνδέσεων που θα χρησιμοποιήσει το ycsb στη βάση. Αυτή η παράμετρος διατηρείται στο 10 καθ’ όλη τη διάρκεια των μετρήσεών μας.

Τέλος, με τη σημαία –threads *αριθμός_threads*, ορίζεται ο αριθμός των νημάτων που θα χρησιμοποιηθούν από το ycsb για να εκτελέσουν τις μετρήσεις.

6.2.2 Παρουσίαση των πειραμάτων και των αποτελεσμάτων τους

Όταν έχει γίνει το compile του YCSB με τις επεκτάσεις για τη MongoDB και MongoDB MVCC, αφού ανεβάσουμε το πακέτο-φάκελο στα απομακρυσμένα μηχανήματα, γίνεται σύνδεση σε ένα από τα απομακρυσμένα μηχανήματα (συνήθως στο Okeanos1), μέσω του PuTTY και το σύστημά μας για τη διεκπεραίωση των πειραμάτων είναι έτοιμο να δεχθεί τις εντολές μας.

Αλλάζουμε τον αριθμό των εγγραφών και των λειτουργιών που θα γίνουν σε κάθε πείραμα (σε 100,000 και 1,000,000 αντίστοιχα, όπως αναφέρθηκε στην προηγούμενη ενότητα). Οι εντολές που χρησιμοποιήθηκαν για τα πειράματά μας και στις οποίες γίνεται αναφορά σε αυτήν την ενότητα, ακολουθούν τα όσα περιγράφηκαν στην προηγούμενη ενότητα, μεταβάλλοντας τις κατάλληλες παραμέτρους της κάθε μίας ανάλογα με το στόχο του πειράματος.

6.2.2.1 Μέτρηση του Throughput με μεταβολή του Αριθμού των Threads

Σε όλες τις μετρήσεις σε αυτό το κομμάτι, θέλαμε να μετρήσουμε το throughput, που μπορεί να επιτύχει ένα thread όταν τρέχει μόνο του, οπότε θέσαμε το target throughput σε ένα πολύ υψηλό νούμερο, το οποίο είναι ανέφικτο (συγκεκριμένα –target 10000000), ώστε να δούμε το πραγματικό μέγιστο throughput της κάθε περίπτωσης. Μεταβάλλαμε τον αριθμό των threads από 1 ως 17, οπότε παίρνουμε τη μέτρηση του throughput για κάθε μία από τις περιπτώσεις αυτές.

Workload B

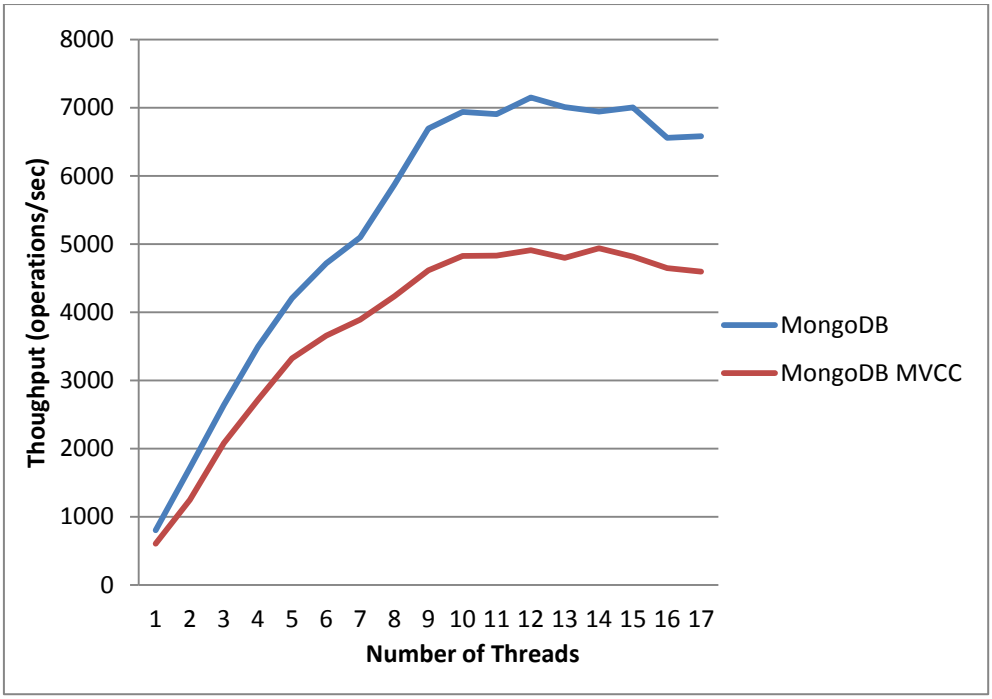
Αρχικά φορτώνουμε το workloadb (με την εντολή load) για τη φυσική έκδοση της MongoDB και δημιουργείται η βάση δεδομένων ycsb στο δεύτερο μηχάνημα που φιλοξενεί τη MongoDB. Στη συνέχεια, εκτελούμε την εντολή run με αριθμό threads από 1 (-threads 1) μέχρι 17. Δε χρειάζεται να γίνεται εκκαθάριση της βάσης δεδομένων, μεταξύ της κάθε μέτρησης, διότι το Workload B αποτελείται από reads και ενημερώσεις, οι οποίες δεν προσθέτουν εγγραφές στην έκδοση της MongoDB και η βάση μένει ίδια.

Για τις μετρήσεις της MongoDB MVCC, φορτώνουμε με το workloadb (με την εντολή load για mongomvccdb) και δημιουργείται η βάση ycsbmvcc. Στη συνέχεια, εκτελούμε την εντολή run για τη MongoMVCCDB, για threads 1 ως 17. Εδώ, η MVCC σε κάθε ενημέρωση προσθέτει μία νέα εγγραφή. Έτσι η βάση δεδομένων μεταβάλλεται μετά από κάθε εκτέλεση. Αυτό σημαίνει, ότι μετά από

κάθε πείραμα, πρέπει να διαγραφεί η βάση ycsbmvcc μέσω του JMongoBrowser και να επαναδημιουργηθεί με την εντολή load.

Τα αποτελέσματα που παίρνουμε φαίνονται στον παρακάτω πίνακα:

Number of Threads	MongoDB Throughput (operations/sec)	MongoDB MVCC Throughput (operations/sec)
1	804	607
2	1717	1248
3	2639	2078
4	3499	2718
5	4205	3323
6	4717	3656
7	5098	3893
8	5874	4234
9	6696	4617
10	6939	4827
11	6910	4834
12	7150	4912
13	7012	4797
14	6944	4939
15	7008	4819
16	6561	4651
17	6584	4596



Εικόνα 36: Το Throughput σε σχέση με τον αριθμό των threads για το workloadb

Παρατηρούμε ότι υπάρχει ένα ταβάνι για την αύξηση του throughput ανάλογα με τον αριθμό των threads που τρέχουν και αυτό συμβαίνει λόγω του ότι μετά τα 10 threads εμφανίζεται ένας συνωστισμός των λειτουργιών των threads, με αποτέλεσμα κάποια threads να αναγκάζονται να αναμένουν την ολοκλήρωση εργασιών άλλων. Παρατηρείται επίσης μία αναμενόμενη διαφορά μεταξύ των επιδόσεων της MongoDB και της έκδοσης με MVCC που οφείλεται στην απομόνωση των λειτουργιών της MVCC αλλά και στο overhead που προστίθεται από την όλη διαδικασία λειτουργίας της MVCC κατά την ενημέρωση. Η διαφορά δεν είναι απαγορευτική, αλλά τίθεται θέμα βελτίωσης των επιδόσεων της MVCC.

Workload C

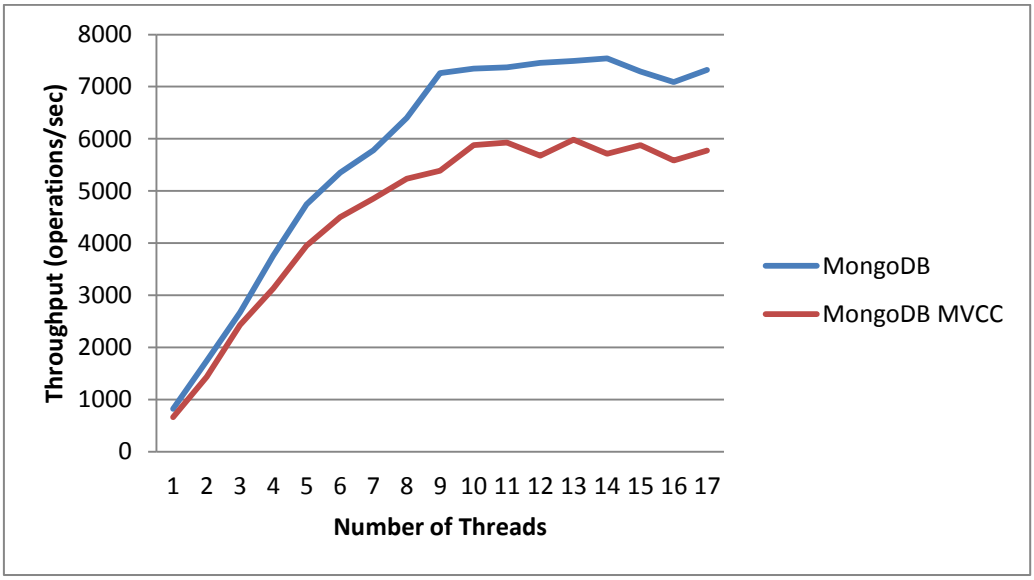
Όμοια με προηγουμένως, φορτώνουμε το workloadc στη βάση ycsb και εκτελούμε το πείραμα για αριθμό threads από 1 ως 17. Το ίδιο κάνουμε και για τη MongoMVCCDB, φορτώνοντας το workloadc στη βάση ycsbmvcc και εκτελώντας για 1 μέχρι 17 threads. Και στις δύο περιπτώσεις τώρα, δεν χρειάζεται η διαγραφή της βάσης και η επαναφόρτωση μετά από κάθε τρέξιμο, αφού το workloadc αποτελείται μόνο από reads, δηλαδή η βάση δε μεταβάλλεται.

Έτσι έχουμε:

Number of Threads	MongoDB Throughput (operations/sec)	MongoDB MVCC Throughput (operations/sec)
1	820	664
2	1744	1434
3	2667	2427
4	3762	3131
5	4742	3952
6	5351	4500
7	5781	4856
8	6404	5232
9	7259	5388
10	7345	5880
11	7372	5930
12	7455	5678
13	7496	5985
14	7542	5711
15	7291	5882
16	7088	5584
17	7323	5777

Εδώ παρατηρούμε πάλι ένα άνω όριο στο κατά πόσο μπορεί να αυξηθεί το throughput ανάλογα με τον αριθμό των threads που χρησιμοποιούνται, λόγω συνωστισμού. Το όριο αυτό είναι πάλι κοντά στα 10 threads.

Εδώ η διαφορά μεταξύ της MongoDB και της έκδοσης με MVCC είναι ακόμη μικρότερη, αφού γίνεται μόνο read κι έτσι η MVCC δεν προσθέτει επιπλέον εγγραφές. Παρόλα αυτά, υπάρχει διαφορά λόγω του overhead που προσθέτει η διαδικασία της MVCC και συγχρονισμού με τον TM.

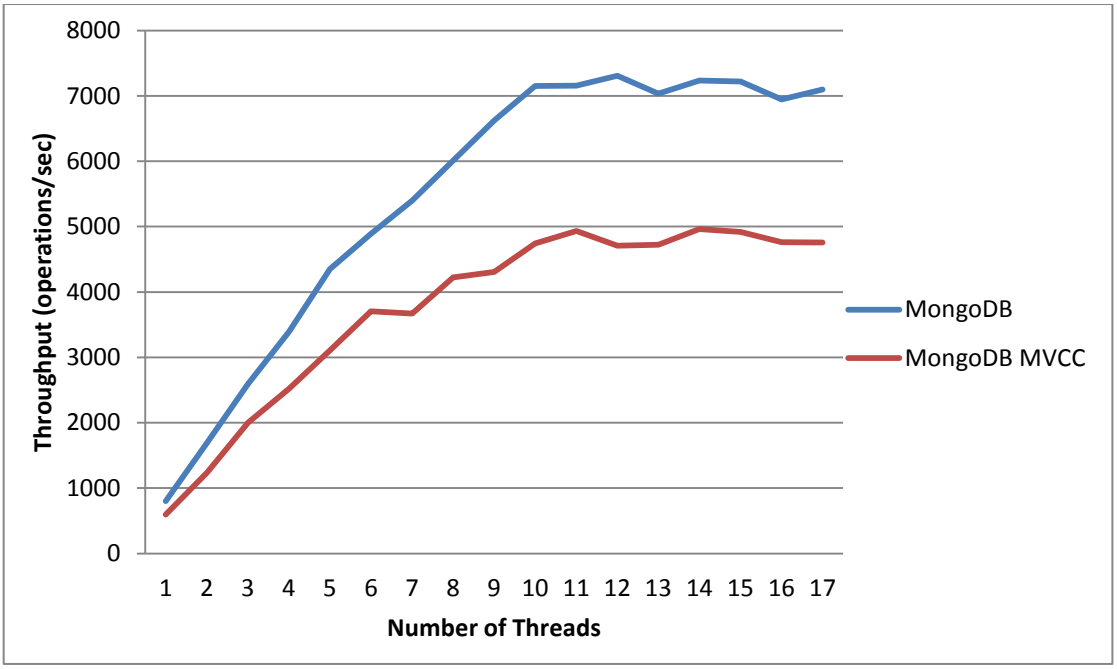


Εικόνα 37: Το Throughput σε σχέση με τον αριθμό των threads για το workloadc

Workload D

Όμοια με προηγουμένως, φορτώνουμε το workloadd στη βάση ycsb και εκτελούμε το πείραμα για αριθμό threads από 1 ως 17. Το ίδιο κάνουμε και για τη MongoMVCCDB, φορτώνοντας το workloadd στη βάση ycsbmvcc και εκτελώντας για 1 μέχρι 17 threads. Και στις δύο περιπτώσεις τώρα, απαιτείται η διαγραφή της βάσης και η επαναφόρτωση με την αντίστοιχη εντολή load μετά από κάθε τρέξιμο, αφού το workloadd έχει και λειτουργίες εισαγωγής στοιχείων, πράγμα που σημαίνει ότι η βάση δεδομένων μεταβάλλεται μετά το τρέξιμο κάθε μέτρησης.

Number of Threads	MongoDB Throughput (operations/sec)	MongoDB MVCC Throughput (operations/sec)
1	803	596
2	1694	1238
3	2592	2003
4	3393	2519
5	4349	3105
6	4892	3703
7	5398	3672
8	6011	4226
9	6622	4309
10	7152	4744
11	7158	4934
12	7309	4710
13	7033	4724
14	7236	4962
15	7219	4918
16	6946	4764
17	7099	4760



Εικόνα 38: Το Throughput σε σχέση με τον αριθμό των threads για το workloadd

Παρατηρείται πάλι ένα ταβάνι για την αύξηση του Throughput, λόγω συνωστισμού, στα 10 threads.

Η διαφορά των δύο εκδόσεων της MongoDB έχει αυξηθεί, όπως φαίνεται, λόγω του αυξημένου overhead που εισάγει η διαδικασία της MVCC, αφού αυτό είναι μεγαλύτερο για το write (και άρα το insert) απ' ότι ήταν για το read. Η διαφορά πάλι, δεν αποτελεί παράγοντα απόρριψης της MVCC, αλλά βελτίωση των επιδόσεων της θα ήταν ένα θετικό βήμα.

Workload E

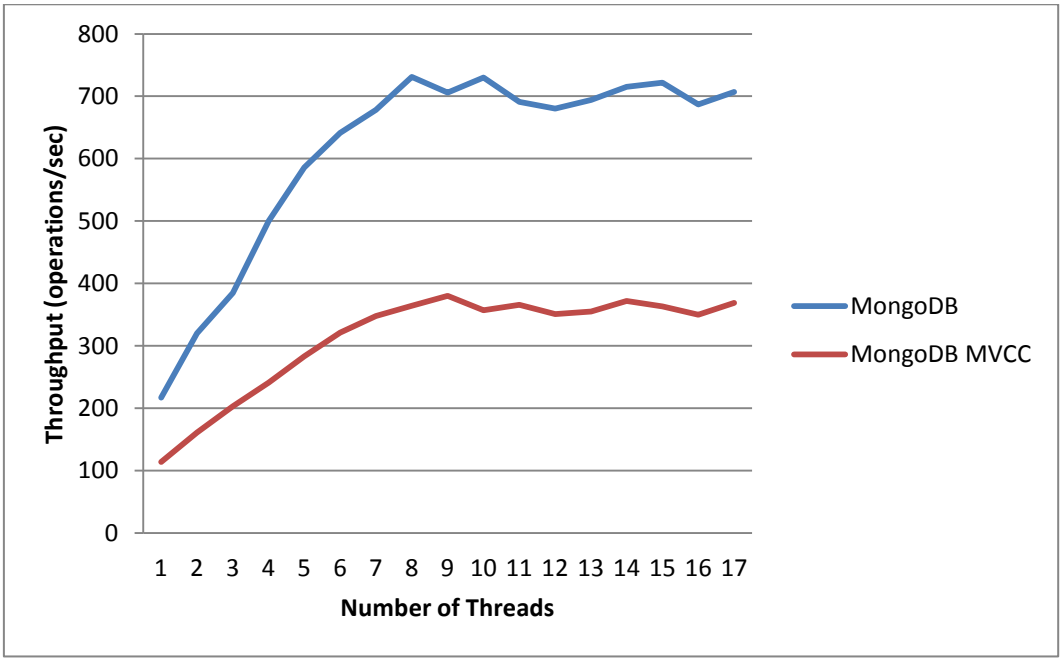
Όμοια με προηγουμένως, φορτώνουμε το workloade στη βάση ycsb και εκτελούμε το πείραμα για αριθμό threads από 1 ως 17. Το ίδιο κάνουμε και για τη MongoMVCCDB, φορτώνοντας το workloade στη βάση ycsbmvcc και εκτελώντας για 1 μέχρι 17 threads. Και στις δύο περιπτώσεις, απαιτείται η διαγραφή της βάσης και η επαναφόρτωση με την αντίστοιχη εντολή load μετά από κάθε τρέξιμο, αφού το workloade έχει και λειτουργίες εισαγωγής στοιχείων, πράγμα που σημαίνει ότι η βάση δεδομένων μεταβάλλεται μετά το τρέξιμο κάθε μέτρησης.

Εδώ παρατηρούμε πολύ χαμηλά νούμερα και για τις δύο περιπτώσεις. Αυτό έχει να κάνει με την υλοποίηση της μεθόδου scan και το χειρισμό της από το YCSB, κάτι το οποίο κάνει το YCSB να «γονατίζει» για την έκδοση της MongoDB (η οποία υπενθυμίζεται ότι δεν υποστηρίζεται από την αρχική του έκδοση). Παρόλα αυτά, τα νούμερα είναι ασφαλή για την εξαγωγή συμπερασμάτων.

Παρατηρούμε για άλλη μία φορά ένα άνω όριο στην αύξηση του Throughput, στα 9 threads αυτή τη φορά. Η διαφορά μεταξύ των δύο εκδόσεων της MongoDB φαίνεται να αυξάνεται (αν και τα νούμερα είναι πολύ μικρά, οπότε η διαφορά αυξάνεται στην πραγματικότητα όσον αφορά στο ποσοστό της). Αυτή

η αύξηση οφείλεται τόσο στο overhead της λειτουργίας του συστήματος της MVCC, όσο και στη διαδικασία για το scan της, εφόσον θα πρέπει να διατρέχεται και το πεδίο _nextCmtTmstmp για την εύρεση του πιο πρόσφατου στοιχείου.

Number of Threads	MongoDB Throughput (operations/sec)	MongoDB MVCC Throughput (operations/sec)
1	217	114
2	320	161
3	385	203
4	500	241
5	586	283
6	641	321
7	678	348
8	731	364
9	706	380
10	730	357
11	691	366
12	680	351
13	694	355
14	715	372
15	722	363
16	687	350
17	707	369



Εικόνα 39: Το Throughput σε σχέση με τον αριθμό των threads για το workloade

6.2.2.2 Μέτρηση του Latency με μεταβολή του Throughput

Σε αυτό το μέρος, γίνεται η μέτρηση της καθυστέρησης (Latency), ανάλογα με το Throughput που θέλουμε να επιτύχουμε. Έτσι, για κάθε τιμή του Throughput ανάμεσα σε 500 και 7500 (με βήμα 500) κάνουμε μέτρηση του latency που προκύπτει για τις λειτουργίες στη βάση δεδομένων.
Ο αριθμός των threads που χρησιμοποιούνται ορίζεται κάθε φορά από τα διαγράμματα του Throughput σε σχέση με τον αριθμό των Threads. Για την ακρίβεια, ανάλογα με το επιθυμητό Throughput, κοιτάμε στο διάγραμμα πόσα threads χρειαζόμαστε για να επιτύχουμε αυτόν τον αριθμό. Δεν υπάρχει λόγος να χρησιμοποιηθούν παραπάνω, εφόσον θα έχουμε μικρότερο target throughput.
Σε περίπτωση που το επιθυμητό Throughput δεν είναι εφικτός αριθμός, τότε απλά χρησιμοποιούμε τον αριθμό των threads που βρήκαμε ότι θα επιφέρουν το μέγιστο δυνατό Throughput.

Workload B

Αρχικά φορτώνουμε το workloadb (με την εντολή load) για τη φυσική έκδοση της MongoDB και δημιουργείται η βάση δεδομένων ycsb στο δεύτερο μηχάνημα που φιλοξενεί τη MongoDB. Στη συνέχεια, εκτελούμε την εντολή run με Throughput (–target) από 500 ως 7500 με βήμα 500 και αριθμό threads όπως περιγράφηκε παραπάνω και όπως φαίνεται στον πίνακα που ακολουθεί. Δε χρειάζεται να γίνεται εκκαθάριση της βάσης δεδομένων, μεταξύ της κάθε μέτρησης, διότι το Workload B αποτελείται από reads και ενημερώσεις, οι οποίες δεν προσθέτουν εγγραφές στην έκδοση της MongoDB και η βάση μένει ίδια.

Για τις μετρήσεις της MongoDB MVCC, φορτώνουμε με το workloadb (με την εντολή load για mongomvccdb) και δημιουργείται η βάση ycsbmvcc. Στη συνέχεια, εκτελούμε την εντολή run για τη MongoMVCCDB, για Throughput από 500 έως 8500. Εδώ, η MVCC σε κάθε ενημέρωση προσθέτει μία νέα εγγραφή. Έτσι η βάση δεδομένων μεταβάλλεται μετά από κάθε εκτέλεση. Αυτό σημαίνει, ότι μετά από κάθε πείραμα, πρέπει να διαγραφεί η βάση ycsbmvcc μέσω του JmongoBrowser (drop database) και να επαναδημιουργηθεί με την εντολή load.

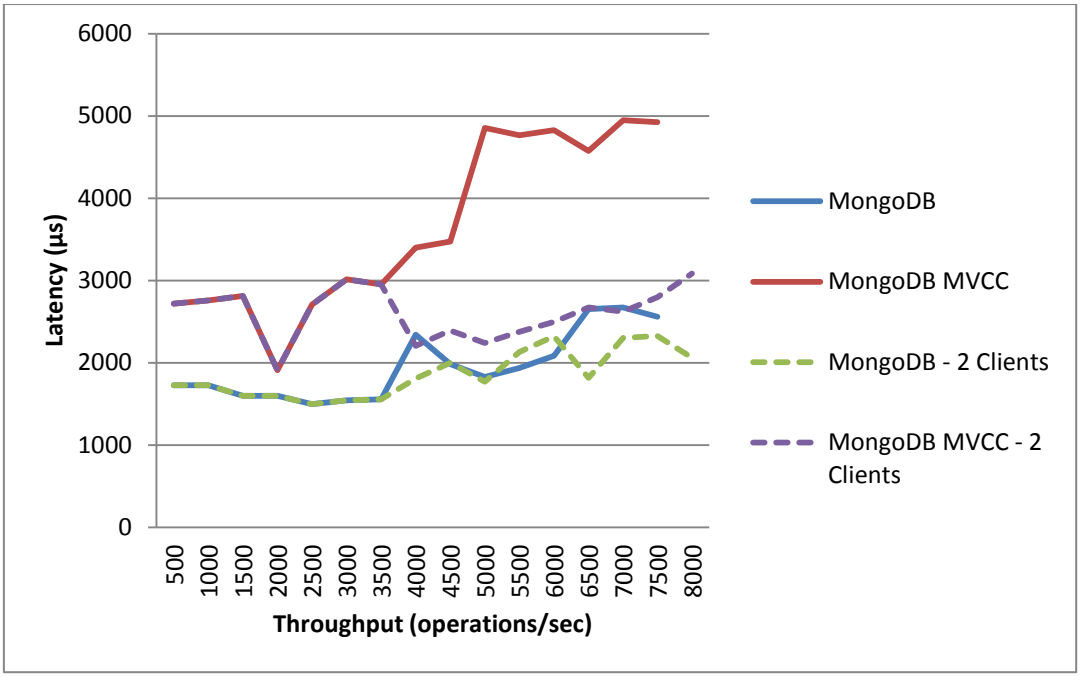
Τα αποτελέσματα φαίνονται στον πίνακα και το γράφημα της επόμενης σελίδας. Εδώ παρατηρούμε τα εξής: Η καθυστέρηση της MVCC είναι αρκετά μεγαλύτερη από την αντίστοιχη της MongoDB. Αυτό είναι λογικό και αναμενόμενο, εφόσον το throughput της έχει μικρότερες τιμές, και λόγω του overhead από τη διαδικασία του συστήματος της MVCC.

Το άλλο φαινόμενο που παρατηρείται είναι η απότομη αύξηση της καθυστέρησης (κυρίως για τη MVCC) από τις 3500 λειτουργίες/δευτερόλεπτο και άνω, όπως μπορεί να φανεί από την κόκκινη καμπύλη στην εικόνα 40. Αυτό συμβαίνει λόγω του ότι το YCSB φτάνει στο δικό του «ταβάνι», λόγω συνωστισμού, και δεν μπορεί να επιτύχει πάνω από 4500 λειτουργίες/δευτερόλεπτο. Για την επίλυση αυτού του προβλήματος, χρησιμοποιούμε δύο clients του YCSB, σε δύο μηχανήματα, τα οποία τρέχουν (run) ταυτόχρονα το πείραμα στην ίδια βάση ενός τρίτου μηχανήματος που φιλοξενεί τις βάσεις (ycsb και ycsbmvcc). Για τη χρήση των δύο clients, οι

εγγραφές ορίζονται σε 500,000 στο καθένα, ώστε να αθροίζονται στις 1,000,000 συνολικά. Με αυτόν τον τρόπο, επιτυγχάνονται Throughputs μεγαλύτερου αριθμού από τις 3500, αθροίζοντας τα αντίστοιχα Throughputs των δύο clients, όπως φαίνεται και στον πίνακα των μετρήσεων.

Wanted Throughput	MongoDB Latency (μs)	Threads	MongoDB MVCC Latency (μs)	Threads
500	1726	1	2720	1
1000	1728	2	2759	2
1500	1597	2	2813	3
2000	1599	3	1909	3
2500	1495	3	2707	4
3000	1543	4	3015	5
3500	1557	4	2952	6
4000	2341	5	3398	8
4500	1982	6	3473	9
5000	1827	7	4853	14
5500	1936	8	4764	14
6000	2084	9	4829	14
6500	2653	9	4576	14
7000	2673	12	4949	14
7500	2560	12	4925	14

Wanted Throughput	Threads	Latency (μs)	Threads	Latency (μs)
500	1	1726	1	2720
1000	2	1728	2	2759
1500	2	1597	3	2813
2000	3	1599	3	1909
2500	3	1495	4	2707
3000	4	1543	5	3015
3500	4	1557	6	2952
4000	3+3	1809	3+3	2204
4500	3+3	2000	3+4	2390
5000	3+3	1769	4+4	2238
5500	3+4	2130	4+5	2376
6000	4+4	2320	5+5	2495
6500	4+4	1814	5+6	2671
7000	4+4	2302	6+6	2620
7500	4+5	2324	6+8	2798
8000	5+5	2059	8+8	3086



Εικόνα 40: Η καθυστέρηση σε σχέση με το Throughput για το workloadb

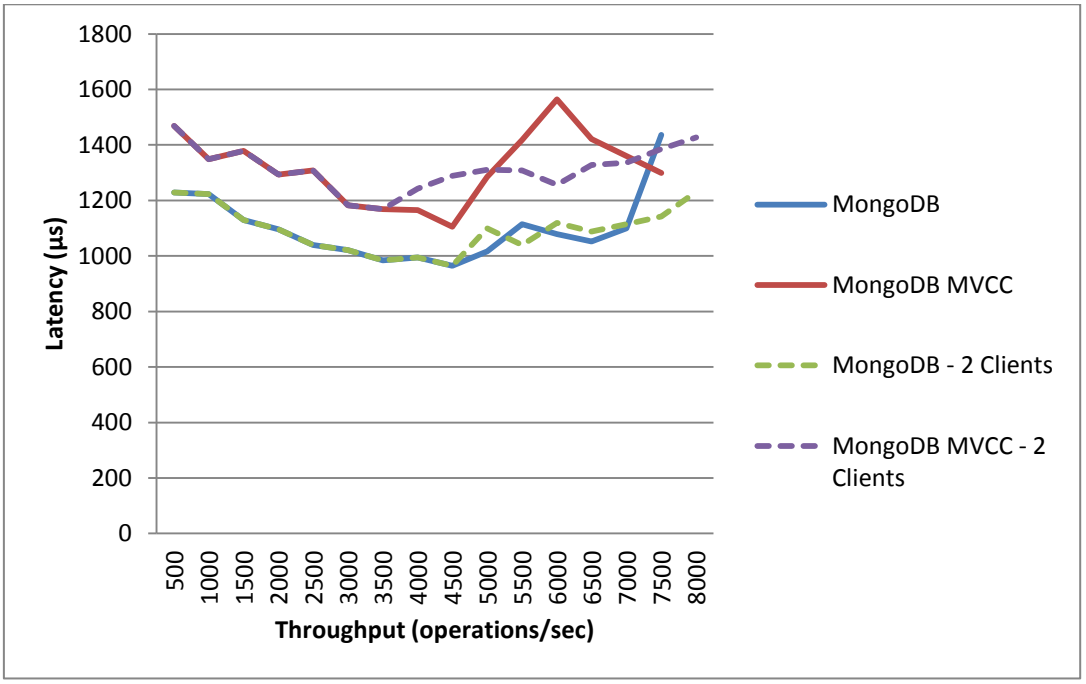
Workload C

Όμοια με το workloadb, φορτώνουμε το workloadc με την αντίστοιχη εντολή load στις ycsb και ycsbmvcc βάσεις δεδομένων και εκτελούμε μετρήσεις για Throughput από 500 μέχρι 7500. Εδώ, δεν είναι απαραίτητη η διαγραφή της βάσης δεδομένων μετά από κάθε εκτέλεση, αφού το workload αυτό αποτελείται μόνο από λειτουργίες read, οι οποίες δεν επηρεάζουν τη βάση δεδομένων.

Επίσης, δεν υπάρχει πραγματικά απότομη αύξηση του Latency όπως στην προηγούμενη περίπτωση, όμως έγιναν οι μετρήσεις για 2 clients, ώστε να μπορεί να χρησιμοποιηθούν ως σημείο αναφοράς.

Wanted Throughput	-	MongoDB Latency (µs)	Threads	-	MongoDB MVCC Latency (µs)	Threads
500		1229	1		1469	1
1000		1223	2		1349	2
1500		1129	2		1378	3
2000		1096	3		1293	3
2500		1039	3		1308	4
3000		1021	4		1182	4
3500		984	4		1169	5
4000		995	5		1165	6
4500		965	5		1105	6
5000		1017	6		1285	8
5500		1115	7		1418	10
6000		1079	8		1564	10
6500		1052	9		1421	10
7000		1099	9		1360	10
7500		1436	14		1299	10

Wanted Throughput	Threads	Latency (μs)	Threads	Latency (μs)
500	1	1229	1	1469
1000	2	1223	2	1349
1500	2	1129	3	1378
2000	3	1096	3	1293
2500	3	1039	4	1308
3000	4	1021	4	1182
3500	4	984	5	1169
4000	5	995	3+3	1242
4500	5	965	3+4	1289
5000	3+3	1099	4+4	1311
5500	3+4	1039	4+4	1308
6000	4+4	1119	4+4	1256
6500	4+4	1088	4+5	1328
7000	4+4	1115	5+5	1336
7500	4+5	1142	5+6	1386
8000	5+5	1231	6+6	1427



Εικόνα 41: Η Καθυστέρηση σε σχέση με το Throughput για το workloadc

Εδώ παρατηρούμε πάλι μία διαφορά στην καθυστέρηση, με τις τιμές της MVCC να είναι μεγαλύτερες, λόγω του συνηθισμένου overhead. Η διαφορά παρ' όλα αυτά είναι μικρότερη, καθώς και οι δύο βάσεις κάνουν μόνο λειτουργίες read.

Η χρήση 2 client του YCSB βελτιώνει την καθυστέρηση (ειδικά για τη MVCC) , όπως ήταν αναμενόμενο και η καθυστέρηση σταθεροποιείται στα επίπεδα των 1300 μs, όπως φαίνεται από τη διακεκομμένη μωβ γραμμή.

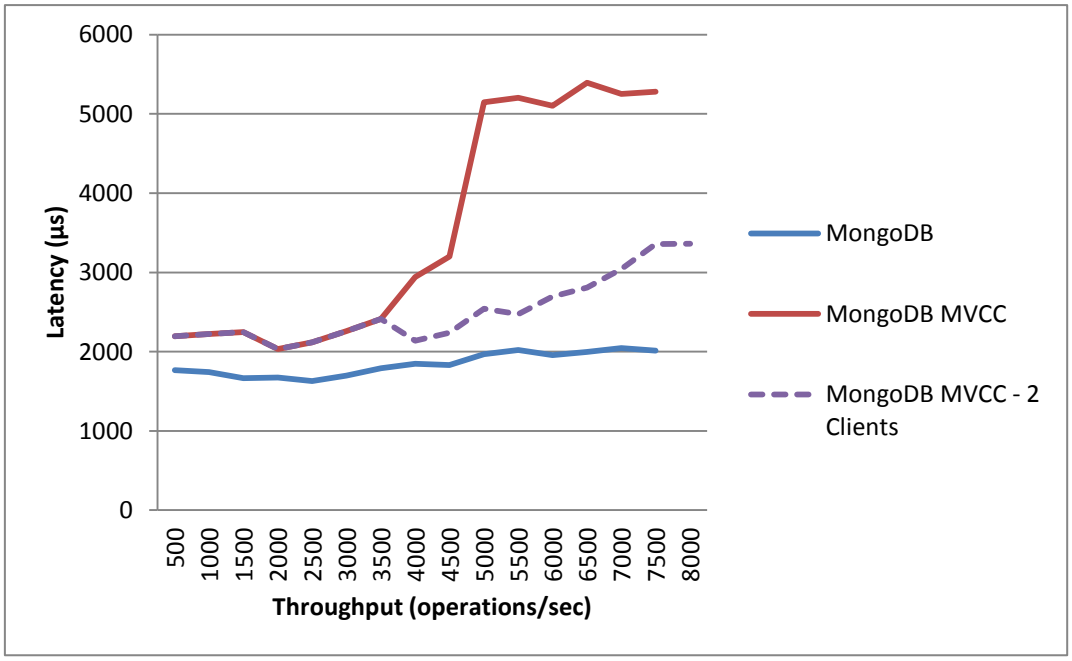
Workload D

Όμοια με την προηγούμενη περίπτωση, φορτώνουμε το workloadd στις βάσεις δεδομένων μας (ycsb και ycsbmvcc) και εκτελούμε τις αντίστοιχες εντολές run για τιμές Throughput από 500 μέχρι 7500, με βήμα 500. Στην περίπτωση αυτή, απαιτείται η διαγραφή της κάθε βάσης δεδομένων μετά την κάθε εκτέλεση, καθώς το workload περιέχει και λειτουργίες εισαγωγής στοιχείων, με αποτέλεσμα την αύξηση του μεγέθους της βάσης δεδομένων. Έτσι διαγράφουμε τη βάση και την επανα-φορτώνουμε με τα workload μετά από κάθε πείραμα.

Wanted Throughput	MongoDB Latency (μs)	Threads	MongoDB MVCC Latency (μs)	Threads
500	1768	1	2193	1
1000	1742	2	2224	2
1500	1666	2	2246	3
2000	1675	3	2031	3
2500	1631	3	2118	4
3000	1697	4	2258	5
3500	1791	5	2412	6
4000	1846	6	2943	8
4500	1831	6	3198	10
5000	1968	7	5148	14
5500	2019	8	5201	14
6000	1956	8	5100	14
6500	1998	9	5393	14
7000	2045	10	5252	14
7500	2012	10	5279	14

Παρατηρείται πάλι μία σαφής διαφορά της καθυστέρησης μεταξύ των δύο εκδόσεων (φυσικής και MVCC) της MongoDB, με τη διαφορά να είναι μεγαλύτερη λόγω της προσθήκης λειτουργιών insert που κάνουν το overhead της MVCC ακόμη μεγαλύτερο (λόγω των writes). Επίσης, παρατηρείται πάλι το φαινόμενο της απότομης αύξησης της καθυστέρησης για την περίπτωση της MVCC, λόγω του φαινομένου συνωστισμού του YCSB. Έτσι, χρησιμοποιείται πάλι η διαδικασία των 2 client του YCSB για να ξεπεραστεί αυτό το πρόβλημα και να φτάσουμε το επιθυμητό σε κάθε περίπτωση Throughput. Κάτι τέτοιο δεν απαιτείται για τη φυσική έκδοση της MongoDB, όπως φαίνεται, καθώς η καθυστέρηση παραμένει σε σταθερά επίπεδα γύρω στα 2000 μs και το δείχνει η μωβ διακεκομμένη γραμμή.

Wanted Throughput	-	Threads	Latency (µs)
500		1	2193
1000		2	2224
1500		3	2246
2000		3	2031
2500		4	2118
3000		5	2258
3500		6	2412
4000		3+3	2139
4500		3+4	2239
5000		4+4	2540
5500		4+5	2472
6000		5+5	2694
6500		5+6	2808
7000		6+6	3044
7500		6+8	3359
8000		8+8	3361



Εικόνα 42 Η Καθυστέρηση σε σχέση με το Throughput για το workloadd

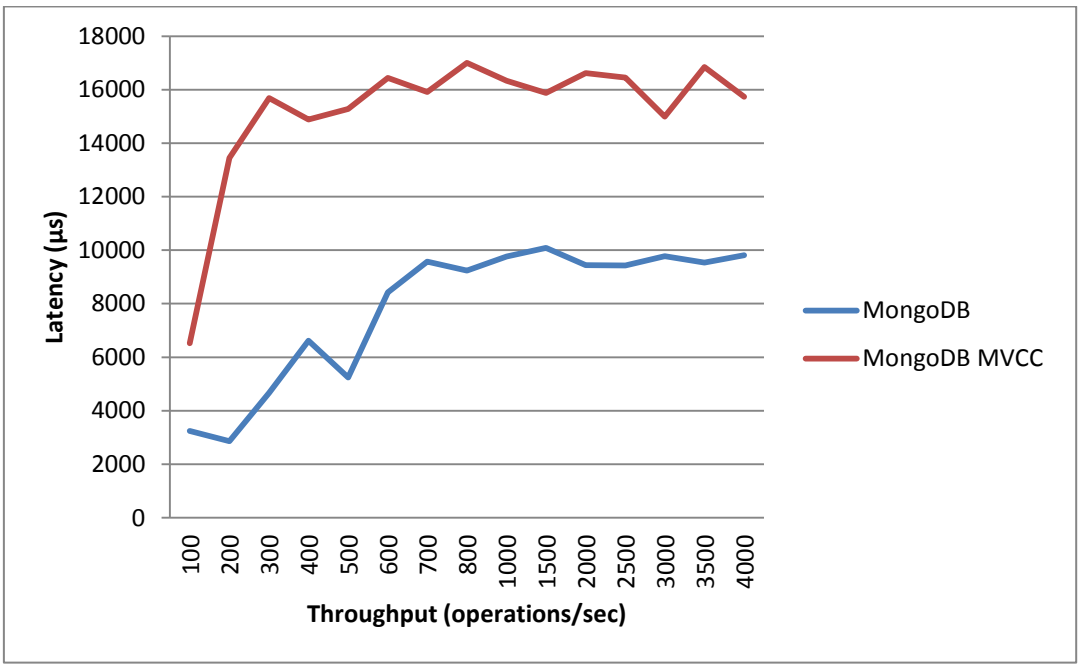
Workload E

Όπως και στις προηγούμενες μετρήσεις, φορτώνουμε τις αντίστοιχες βάσεις δεδομένων. Αυτή τη φορά, λόγω της ιδιομορφίας του συγκεκριμένου workload και των μικρών τιμών Throughput που μας έδωσε κατά τις μετρήσεις του σε σχέση με τον αριθμό των threads, εκτελούμε μετρήσεις για τιμές Throughput μεταξύ 100 και 1000, με βήμα 100 και στη συνέχεια 1000 με 4000, με βήμα 500.

Πάλι, λόγω των λειτουργιών εισαγωγής στοιχείων στις βάσεις, είναι απαραίτητη η διαγραφή και επανα-φόρτωση των στοιχείων των βάσεων, μετά από κάθε

εκτέλεση του YCSB. Τα αποτελέσματα των μετρήσεων φαίνονται στον παρακάτω πίνακα.

Wanted Throughput	-	MongoDB Latency (μs)	Threads	-	MongoDB MVCC Latency (μs)	Threads
100		3248	1		6520	1
200		2867	1		13453	3
300		4671	2		15687	6
400		6621	4		14879	6
500		5240	4		15283	6
600		8424	6		16438	6
700		9578	8		15909	6
800		9235	8		17002	6
1000		9762	8		16329	6
1500		10083	8		15871	6
2000		9437	8		16624	6
2500		9425	8		16453	6
3000		9780	8		14987	6
3500		9542	8		16840	6
4000		9808	8		15734	6



Εικόνα 43: Η Καθυστέρηση σε σχέση με το Throughput για το workloade

Παρατηρούμε πολύ υψηλές τιμές καθυστέρησης και για τις δύο εκδόσεις της MongoDB, πράγμα που οφείλεται, όπως και στις μετρήσεις του Throughput για το ίδιο workload, στη διαχείριση του YCSB για τη scan στη MongoDB. Οι υψηλές αυτές τιμές δικαιολογούν και τις πολύ χαμηλές τιμές του Throughput που βρέθηκαν προηγουμένως, αλλά μας επιτρέπουν πάλι να βγάλουμε το συμπέρασμα της μεγάλης διαφοράς μεταξύ της καθυστέρησης της φυσικής έκδοσης της MongoDB και της αντίστοιχης έκδοσης του συστήματος με MVCC.

Αυτή η διαφορά οφείλεται στο συνηθισμένο overhead της διαδικασίας της MVCC, στο επίπεδο απομόνωσης των συνδιαλλαγών της και στη λειτουργία scan, η οποία στην περίπτωση της MVCC πρέπει να ελέγξει και το πεδίο _nextCmtTmstmp για να βρει την τελευταία έκδοση των στοιχείων σε κάθε περίπτωση.

7

Επίλογος

Στην ενότητα αυτή γίνεται μία σύνοψη της εργασίας και παρουσιάζονται τα συνολικά συμπεράσματα που μπορούμε να βγάλουμε από τα αποτελέσματα. Γίνεται αναφορά στα θετικά και τα αρνητικά στοιχεία του συστήματός μας, όπως αυτά προκύπτουν από τις μετρήσεις μας, ενώ παρουσιάζονται και ιδέες για επέκταση του συστήματος, βελτιώσεις και αντιμετώπιση των αδύναμων σημείων του.

7.1 Σύνοψη και συμπεράσματα

Στόχος της εργασίας ήταν η δημιουργία του συστήματος παροχής υπηρεσιών , με την υποστήριξη ταυτόχρονων συνδιαλλαγών, για τη μη-σχεσιακή βάση MongoDB. Αυτό επιτεύχθηκε μέσω της χρήσης ενός ολιστικού Διαχειριστή Συνδιαλλαγών (TM), ο οποίος προσφέρει την υποστήριξη συνδιαλλαγών και των ιδιοτήτων ACID, και η ενσωμάτωση του οποίου στο σύστημά μας απαιτούσε την επέκταση της βάσης δεδομένων σε μία έκδοση ταυτοχρονισμού πολλών-εκδόσεων.

Σημαντικό κομμάτι μετά το συνολικό στήσιμο του συστήματος ήταν η επαλήθευση της απόδοσής του και η αξιολόγηση των αποτελεσμάτων του. Αυτό έγινε μέσω του εργαλείου αξιολόγησης συστημάτων νέφους εξυπηρέτησης, YCSB, το οποίο επεκτάθηκε κατάλληλα για την υποστήριξη της MongoDB και της έκδοσης της MongoDB του συστήματός μας.

Παρατηρώντας τα αποτελέσματα των μετρήσεων, οι οποίες παρουσιάστηκαν στο προηγούμενο κεφάλαιο, μπορούμε να βγάλουμε αρκετά συμπεράσματα για τις επιδόσεις του συστήματός μας, σε σχέση με τη φυσική έκδοση της MongoDB, και ειδικά για την κάθε περίπτωση workload.

Αρχικά, φαίνεται πως σε κάθε περίπτωση υπάρχει τουλάχιστον ένα μικρό overhead (επιπλέον φορτίο) στο σύστημα, λόγω της όλης διαδικασίας που περιγράφηκε στο κεφάλαιο 4, για κάθε λειτουργία. Το overhead είναι μεγαλύτερο ή μικρότερο, ανάλογα με το αν πρόκειται για read ή write. Το write προσθέτει μία επιπλέον καθυστέρηση, καθώς απαιτεί και έλεγχο συγκρούσεων write από τον ολιστικό TM. Έτσι βλέπουμε ότι στο Workload C, το οποίο αποτελείται από λειτουργίες read μόνο, η διαφορά μεταξύ των δύο εκδόσεων της Mongo είναι μικρή. Αντίθετα, αυξάνεται στις άλλες περιπτώσεις, στις οποίες γίνεται και write, δηλαδή στην ενημέρωση και την εισαγωγή στοιχείων. Το scan έχει μία αναμενόμενη καθυστέρηση, καθώς απαιτείται η ανάγνωση όλων των εγγραφών, αλλά στην MVCC απαιτείται και η εύρεση της τελευταίας έκδοσης, μέσω του timestamp του commit της επόμενης έκδοσης. Επίσης, η ενημέρωση αυξάνει το μέγεθος της βάσης δεδομένων, πράγμα που δε συμβαίνει στη φυσική έκδοση της MongoDB. Αυτό μπορεί να επηρεάσει την επίδοση σε σχέση με τη φυσική έκδοση, σε περιπτώσεις που έχουμε ενημέρωση των εγγραφών (όπως στο Workload B).

Όλα αυτά, αλλά και το υψηλότερο επίπεδο απομόνωσης που υπάρχει μεταξύ των συνδιαλλαγών στο σύστημά μας, οδηγούν στην εμφάνιση αυτής της διαφοράς για το εφικτό Throughput και την Καθυστέρηση, μεταξύ των δύο εκδόσεων της MongoDB. Ακόμη και στη χειρότερη περίπτωση, η διαφορά δεν είναι απαγορευτική, αλλά αυτό εξαρτάται και από το μέγεθος και τη λειτουργία της εφαρμογής που θέλουμε να ενσωματώνει το σύστημα της MongoDB με MVCC.

Από την άλλη, η χρήση των συνδιαλλαγών και η προσθήκη των ιδιοτήτων που αυτές προσφέρουν σε μία βάση δεδομένων γίνεται δυνατή μέσω του συστήματός μας. Η MongoDB με MVCC, σε συνδυασμό με τον ολιστικό TM, υποστηρίζει τις συνδιαλλαγές, οι οποίες έλειπαν από τη MongoDB και εξασφαλίζουν τις ACID ιδιότητες που είναι απαραίτητες για ένα σύγχρονο σύστημα βάσεων δεδομένων και εξυπηρέτησης νέφους.

Έτσι, το πόσο αξίζει να χρησιμοποιηθεί η MVCC έκδοσή μας της MongoDB εξαρτάται από το τι ζητείται από την εφαρμογή, από το μέγεθος και από τη χρήση της. Όπως στις περισσότερες τεχνολογίες, υπάρχει πάντα ένα αντίτιμο (trade-off) για τη χρήση τους, σε σχέση με άλλες τεχνολογίες. Εδώ το trade-off αυτό είναι μεταξύ των επιδόσεων και των συνδιαλλαγών και ACID ιδιοτήτων. Αν ένα σύστημα θέλουμε να έχει ιδιότητες όπως η απομόνωση και η ανθεκτικότητα, κάτι που σημαίνει και απουσία των περισσότερων συγκρούσεων, όπως στην περίπτωση ενός σύγχρονου συστήματος νέφους εξυπηρέτησης, τότε η χρήση του TM και συνεπώς της MVCC είναι αναπόφευκτη και όπως φάνηκε από τις μετρήσεις η διαφορά δεν είναι τόσο μεγάλη ώστε να είναι αποτρεπτική..

Από την άλλη, ένα σύστημα μπορεί να μην μας ενδιαφέρει να είναι το ίδιο αποδοτικό ως προς την αντιμετώπιση των σφαλμάτων και την αποφυγή συγκρούσεών του και ταυτόχρονα να απαιτείται η πλέον ταχύτερη διεκπεραίωση των λειτουργιών. Σε αυτήν την περίπτωση η φυσική έκδοση της MongoDB θα ήταν προτιμότερη.

7.2 Μελλοντικές επεκτάσεις

Παρότι ο στόχος της εργασίας επετεύχθη και δημιουργήθηκε ένα σύστημα που υποστηρίζει τις συνδιαλλαγές, τη χρήση του TM και συνεπώς της MVCC, παρατηρείται αυτή η διαφορά ταχύτητας από την αντίστοιχη φυσική έκδοση της MongoDB. Μπορεί να μην είναι απαγορευτική, όμως μπορεί να αυξήσει αρκετά έως πολύ το χρόνο διεκπεραίωσης των λειτουργιών της βάσης δεδομένων σε μία μεγάλη εφαρμογή. Οι σύγχρονες εφαρμογές διαδικτύου απαιτούν την ταχύτερη δυνατή χρήση των βάσεων δεδομένων, σε συνδυασμό με την ασφάλεια που προσφέρουν οι ACID δυνατότητες που ενσωματώθηκαν με τις συνδιαλλαγές.

Αυτό σημαίνει, ότι η ιδανική λύση είναι η βελτίωση του συστήματος και η επέκτασή του. Η κεντρική ιδέα για επέκταση είναι γύρω από τη χρήση πολλαπλών βάσεων δεδομένων, με την αντίστοιχη επέκταση Mongo-Ext για την κάθε μία, τις οποίες θα συγχρονίζει συνολικά ένας ολιστικός TM. Αυτό θα σήμαινε ότι θα μπορούσαν να γίνονται περισσότερες λειτουργίες ταυτόχρονα και να φθάσουμε σε σαφώς μεγαλύτερα επίπεδα Throughput και μείωση της καθυστέρησης. Σε αυτήν την περίπτωση όμως, θα πρέπει να επεκταθεί ο TM για το συγχρονισμό της ταυτόχρονης χρήσης όλων των βάσεων, αλλά και η μετατροπή των ίδιων των επεκτάσεων των βάσεων και του οδηγού MongoMVCC-Driver, καθώς δημιουργούνται θέματα με τις πολλαπλές εκδόσεις, το συγχρονισμό τους και την εύρεση της τελευταίας ενημέρωσης.

8

Βιβλιογραφία

[1] A. Bilas, G. Saloustros, G. Papadakis, P. Kranas, S. Stamokostas και D. Dominguez-Sal, «Coherent and Rich PaaS with a Common Programming Model,» 2014.

[2] D. Majumdar, «A Quick Survey of MultiVersion Concurrency Algorithms,» p. 8, 4 Νοέμβριος 2007.

[3] R. Jimenez-Peris, I. Brondino, M. Patiño-Martínez και P. Kranas, «Holistic Transaction Management Architecture».

[4] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan και R. Sears, «Benchmarking Cloud Serving Systems with YCSB».

[5] D. C. Tsitchizris και F. H. Lochovsky, Data Models, Prentice–Hall, 1982.

[6] P. Beynon-Davies, Database Systems 3rd Edition, Palgrave, Basingstoke, UK, 2004.

[7] C. W. Bachmann, «The Programmer as Navigator,» 1973.

[8] S. Fortune, «A Brief History of Databases,» 27 Φεβρουάριος 2014. [Ηλεκτρονικό]. Available: http://avant.org/media/history-of-databases.

[9] «DB-Engines Ranking,» Ιανουάριος 2013. [Ηλεκτρονικό]. Available: http://db-engines.com/en/ranking. [Πρόσβαση 2013 Ιανουάριος 22].

[10] S. Proctor, «Exploring the Architecture of the NuoDB Database,» 12 Ιούλιος 2013.

[11] «Why NoSQL?,» Couchbase, 2014. [Ηλεκτρονικό]. Available: http://www.couchbase.com/nosql-resources/what-is-no-sql. [Πρόσβαση

10 Φεβρουάριος 2015].

[12] T. G. Peter Melle, «The NIST Definition of Cloud,» *NIST Special Publication 800-145,* 2011.

[13] CloudAve, «Recession Is Good For Cloud Computing – Microsoft Agrees,» 22 Αύγουστος 2010. [Ηλεκτρονικό]. Available: http://www.cloudave.com/link/recession-is-good-for-cloud-computing-microsoft-agrees.

[14] F. Gens, «Defining "Cloud Services" and "Cloud Computing",» 23 Σεπτέμβριος 2008. [Ηλεκτρονικό]. Available: http://blogs.idc.com/ie/?p=190.

[15] H. Qiang και e. al, «Formulating Cost-Effective Monitoring Strategies for Service-based Systems.,» 2013.

[16] «A Self-adaptive hierarchical monitoring mechanism for Clouds,» [Ηλεκτρονικό]. Available: Elsevier.com.

[17] H. Smith, Xero For Dummies, John Wiley & Sons, 2013.

[18] R. King, «Cloud Computing: Small Companies Take Flight,» *Bloomberg BusinessWeek,* 2008.

[19] «Encrypted Storage and Key Management for the cloud,» 30 Ιούλιος 2009. [Ηλεκτρονικό]. Available: Cryptoclarity.com. [Πρόσβαση 22 Αυγούστου 2010].

[20] «Is The Private Cloud Really More Secure?,» 2014 Οκτώριος 12. [Ηλεκτρονικό]. Available: CloudAndCompute.com.

[21] B. Prasad Rimal, E. Choi και I. Lumb, «A Taxonomy and Survey of Cloud Computing Systems,» σε *Fifth International Joint Conference on INC, IMS and IDC*, 2009.

[22] Google Support, « What's included in my edition of Gmail?,» 2012. [Ηλεκτρονικό]. Available: http://support.google.com/a/bin/answer.py?hl=en&answer=175121.

[23] Facebook, «Company Info - Facebook Newsroom,» 2014. [Ηλεκτρονικό]. Available: https://newsroom.fb.com/company-info/.

[24] M. Armburst, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica και M. Zaharia, «Above the Clouds: A Berkeley View of Cloud Computing,» 10 Φεβρουάριους 2009.

[25] MongoDB, «Introduction to MongoDB,» MongoDB, [Ηλεκτρονικό]. Available: http://www.mongodb.org/about/introduction/. [Πρόσβαση 12 Φεβρουάριος 2015].

[26] MongoDB, «MongoDB Replica Introduction,» [Ηλεκτρονικό]. Available: http://docs.mongodb.org/manual/core/replication-introduction/. [Πρόσβαση 2015].

[27] MongoDB, «MongoDB Sharding,» MongoDB, [Ηλεκτρονικό]. Available: http://docs.mongodb.org/manual/sharding/.

[28] MongoLab, «Why is MongoDB wildly popular? It's a data structure thing.,» 2 Αύγουστος 2012. [Ηλεκτρονικό]. Available: http://blog.mongolab.com/2012/08/why-is-mongodb-wildly-popular/.

[29] E. Kindler και I. Krivy, «Object-Oriented Simulation of systems with sophisticated control,» *International Journal of General Systems,* p. 313–343, 2011.

[30] J. Lewis και W. Loftus, Java Software Solutions Foundations of Programming Design 6th ed, Pearson Education Inc., 2008.

[31] B. Pierce, Types and Programming Languages, MIT Press, 2002.