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

Αυτοματοποιημένη, ελαστική διαχείριση NoSQL συστημάτων σε περιβάλλοντα υπολογιστικών νεφών για διαφορετικά είδη φόρτου εργασίας

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ευδοκία Σ. Κασσέλα

Επιβλέπων : Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.

Αθήνα, Φεβρουάριος 2014

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

Αυτοματοποιημένη, ελαστική διαχείριση NoSQL συστημάτων σε περιβάλλοντα υπολογιστικών νεφών για διαφορετικά είδη φόρτου εργασίας

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ευδοκία Σ. Κασσέλα

Επιβλέπων : Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την ….η Φεβρουαρίου 2014.

.....	.....	.....
Νεκτάριος Κοζύρης Καθηγητής Ε.Μ.Π.	Νικόλαος Παπασπύρου Επίκ. Καθηγητής Ε.Μ.Π.	Δημήτριος Τσουμάκος Επίκ. Καθηγητής Ι.Π.

Αθήνα, Φεβρουάριος 2014

.....

Ευδοκία Σ. Κασσέλα

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Τα τελευταία χρόνια, έχει σημειωθεί μεγάλη αύξηση της δημοτικότητας του cloud computing. Μέσα από τις πλατφόρμες που προσφέρουν υπηρεσίες IaaS οι χρήστες έχουν τη δυνατότητα να δεσμεύσουν ελαστικά πόρους για την εκτέλεση των εφαρμογών τους. Καθώς όμως αλλάζουν δυναμικά οι απαιτήσεις της εφαρμογής απαιτείται αυξομείωση των χρησιμοποιούμενων πόρων, η οποία γίνεται είτε χειροκίνητα από το χρήστη είτε με τη χρήση μιας συγκεκριμένης απλής τακτικής που προσφέρουν διάφορα συστήματα αυτόματης διαχείρισης του νέφους. Οι τακτικές που χρησιμοποιούν αυτά τα συστήματα είναι προκαθορισμένες και μπορεί να μην απευθύνονται στις ανάγκες του χρήστη για αυτόματη κλιμάκωση ανάλογα με τις ανάγκες της εφαρμογής του.

Στην παρούσα εργασία αξιολογούμε την επίδοση του TIRAMOLA, ενός συστήματος που επιτρέπει την αυτόματη αυξομείωση του μεγέθους μιας NoSQL συστοιχίας με βάση μια τακτική ορισμένη από το χρήστη, σε διαφορετικές συνθήκες λειτουργίας της συστοιχίας. Συγκεκριμένα, μελετάμε αν αυξομειώνει επιτυχώς το μέγεθος της συστοιχίας όταν εφαρμόσουμε φορτίο διαφορετικού μεγέθους και είδους. Με κατάλληλη επιλογή της τακτικής αλλαγής μεγέθους ώστε να περιέχει πληροφορίες για το είδος του φορτίου εκτός από το μέγεθός του, τα αποτελέσματά μας επιβεβαιώνουν ότι ο TIRAMOLA μπορεί να λειτουργήσει σωστά υπό οποιεσδήποτε συνθήκες καθιστώντας τον ένα αποδοτικό και αξιόπιστο εργαλείο που μπορεί να καλύψει οποιαδήποτε ανάγκη του χρήστη.

Λέξεις Κλειδιά: δέσμευση πόρων, υπολογιστικό νέφος, ελαστικότητα, διαχείριση νέφους, NoSQL, TIRAMOLA

Abstract

Lately, the use of cloud computing has gained extreme popularity. Through cloud platforms that provide infrastructure as a service (IaaS) users can elastically provision resources allowing their applications to throttle them. As the needs of an application change dynamically, it is necessary to expand or contract dedicated resources. Usually resources' scaling is manually performed by the user or with a manager that dynamically consolidates remote cloud resources based on a simple predefined policy. However, the policies used by such systems are specific and may not address to the needs of the user's application.

In this work we evaluate the performance of TIRAMOLA, a cloud-enabled framework that allows automated resizing of NoSQL clusters according to any user-defined policy, in different cluster operation conditions. In particular, we study if it can correctly decide the cluster size with an incoming load of different size and query types. For this purpose, we choose the scaling policy so as to contain performance metrics divided by query type and also modify the decision making process. An extensive experimental evaluation on an HBase cluster confirms that workload-aware TIRAMOLA can operate successfully in any environment, behaving accordingly to any input load, and so being determined as an efficient tool that can cover any user's need.

Keywords: cloud resource provisioning; elasticity; policy-based optimization; NoSQL; TIRAMOLA; workload aware

Ευχαριστίες

Με την παρούσα διπλωματική εργασία ολοκληρώνεται ένας εξαετής κύκλος σπουδών στη Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου. Με αυτή την ευκαιρία θα ήθελα να εκφράσω την ευγνωμοσύνη μου σε ορισμένους ανθρώπους, των οποίων η συμβολή ήταν πολύτιμη για την περάτωση αυτής της εργασίας και των σπουδών μου.

Αρχικά θα ήθελα να ευχαριστήσω τον καθηγητή Νεκτάριο Κοζύρη, που μου έδωσε την ευκαιρία να εργαστώ πάνω σε ένα πολύ ενδιαφέρον και σύγχρονο αντικείμενο, το cloud computing.

Επίσης, θα ήθελα να ευχαριστήσω τους ερευνητές του Εργαστηρίου Υπολογιστικών Συστημάτων του Ε.Μ.Π. Ιωάννη Κωνσταντίνου και Χριστίνα Μπούμπουκα, για την αμέριστη βοήθεια και τις χρήσιμες συμβουλές τους, οι οποίες διευκόλυναν σημαντικά την περάτωση της εργασίας αυτής.

Τέλος, θα ήθελα να ευχαριστήσω την οικογένειά μου για την υποστήριξή τους σε κάθε μου επιλογή όλα αυτά τα χρόνια.

Πίνακας Περιεχομένων

Πίνακας σχημάτων

Κεφάλαιο 1
Εισαγωγή

1.1 Elasticity σε cloud computing με εφαρμογή σε NoSQL συστήματα

Η σημαντική αύξηση των υπολογιστικών και αποθηκευτικών απαιτήσεων σε εφαρμογές παγκόσμιου ιστού, όπως ανάλυσης στοιχείων ιστού (web analytics), επιχειρησιακής νοημοσύνης και κοινωνικής δικτύωσης, είχε ως αποτέλεσμα τα χρησιμοποιούμενα κεντρικά συστήματα με SQL βάσεις δεδομένων να μην μπορούν να ανταπεξέλθουν σε αυτές τις απαιτήσεις. Κατανοώντας τους περιορισμούς από τη χρήση κεντρικών συστημάτων, η έρευνα στράφηκε στη χρήση NoSQL βάσεων δεδομένων σε κατανεμημένα συστήματα πολλών κόμβων και στην ανάπτυξη εργαλείων για την αυτόματη διαχείρισή τους.

Οι NoSQL βάσεις δεδομένων είναι οριζόντια κλιμακώσιμοι, κατανεμημένοι, μη-σχεσιακοί χώροι αποθήκευσης δεδομένων. Μπορούν να αποθηκεύσουν και να ταξινομήσουν πολύ μεγάλα σύνολα δεδομένων, συνήθως χωρίς περιορισμό ως προς το μοντέλο, εξυπηρετώντας ταυτόχρονα ένα μεγάλο αριθμό ερωτημάτων. Τα βασικά χαρακτηριστικά τους που σχετίζονται με τη κλιμακωσιμότητα και την απόδοση είναι:

- Το auto-sharding: Η κατανομή των δεδομένων σε εξυπηρετητές γίνεται αυτόματα χωρίς να απαιτείται η συμμετοχή άλλων εφαρμογών. Εξυπηρετητές μπορούν να προστίθενται και να αφαιρούνται χωρίς να σταματά η λειτουργία του συστήματος και τα δεδομένα μοιράζονται αυτόματα στους εξυπηρετητές. Οι περισσότερες NoSQL βάσεις δεδομένων διατηρούν επίσης αντίγραφα των δεδομένων, αποθηκεύοντας τα αντίγραφα σε διάφορους εξυπηρετητές ακόμα και σε διαφορετικά κέντρα δεδομένων, για να διασφαλίσουν τη διαθεσιμότητα των δεδομένων και τη δυνατότητα ανάκτησής τους σε περίπτωση αστοχίας υλικού.
- Το caching: Για τη μείωση του χρόνου αναμονής (latency) και την αύξηση του ρυθμού εξυπηρέτησης ερωτημάτων (throughput) στα δεδομένα που διατηρεί η βάση, γίνεται caching των δεδομένων στη μνήμη του συστήματος με ευθύ τρόπο χρησιμοποιώντας προηγμένες τεχνολογίες. Αυτή η μέθοδος είναι πιο απλοϊκή και

κατανοητή συγκριτικά με τη σχεσιακή τεχνολογία που το κομμάτι του caching είναι συνήθως ένα ξεχωριστό κομμάτι υποδομής που πρέπει να αναπτυχθεί και ενσωματωθεί σε κάθε εξυπηρετητή και το οποίο θα διαχειρίζεται ρητά η ομάδα εργασιών.

Η κλιμακωσιμότητα είναι κυρίως εφικτή μέσω του sharding. Χρησιμοποιώντας αυτό το μηχανισμό, πολλές NoSQL υλοποιήσεις μπορούν να προσαρμόζουν την απόδοσή τους: ο ρυθμός εξυπηρέτησης, ο χρόνος αναμονής, κα. διαφοροποιούνται ανάλογα με το μέγεθος των δεσμευμένων πόρων (όπως ο αριθμός των κόμβων-εξυπηρετητών).

Η ελαστική απόδοση καθιστά τις NoSQL βάσεις δεδομένων ως τις πιο κατάλληλες για χρήση σε IaaS (Infrastructure as a Service) πλατφόρμες υπολογιστικών νεφών (cloud) που παρέχουν τη δυνατότητα ελαστικής δέσμευσης και απελευθέρωσης πόρων ανάλογα με τις ανάγκες του χρήστη. Η ελαστικότητα στο νέφος ορίζεται ως η ικανότητα επέκτασης ή περιορισμού των χρησιμοποιούμενων πόρων σύμφωνα με μια συγκεκριμένη πολιτική. Η αυτόματη όμως κλιμάκωση του μεγέθους των πόρων με βάση μια προκαθορισμένη πολιτική (auto-scaling) έχει προσδιοριστεί ως μια από τις κορυφαίες προκλήσεις στο χώρο του cloud computing [24], κυρίως λόγω της δυσκολίας για τον χρήστη να ορίσει κατάλληλα την επιθυμητή πολιτική όταν η εφαρμογή του εκτελείται σε απομακρυσμένη υποδομή παρόχου και οι ανάγκες της αλλάζουν δυναμικά. Υπάρχουν αρκετά συστήματα που προσφέρουν τη δυνατότητα αυτόματης διαχείρισης του νέφους με χρήση κάποιας υφιστάμενης απλής πολιτικής. Στην εργασία αυτή θα μελετήσουμε το πιο γνωστό εργαλείο ανοικτού κώδικα που μπορεί να χρησιμοποιηθεί για αυτό το σκοπό, που ονομάζεται TIRAMOLA.

Ο TIRAMOLA είναι ένα σύστημα που επιτρέπει την αυτοματοποιημένη επέκταση ή περιορισμό των δεσμευμένων πόρων μιας οποιασδήποτε NoSQL συστοιχίας που είναι υλοποιημένη σε νέφος, σύμφωνα με τακτικές ορισμένες από το χρήστη. Επιτρέπει αδιάκοπη αλληλεπίδραση με τις περισσότερες IaaS πλατφόρμες, ζητώντας/απελευθερώνοντας πόρους εικονικών μηχανών (VMs) και τις οργανώνει σε μια NoSQL συστοιχία. Επίσης περιέχει μια μονάδα παρακολούθησης που καταγράφει μετρικά από τη μεριά του χρήστη αλλά και των εξυπηρετητών της συστοιχίας. Οι αποφάσεις αλλαγής μεγέθους της συστοιχίας μοντελοποιούνται ως μια Μαρκοβιανή διαδικασία λήψης αποφάσεων (MDP) που αυτόματα αποφασίζει την πιο επικερδή κατάσταση λειτουργίας (δηλαδή αριθμό VMs) για τη συστοιχία σε πραγματικό χρόνο σύμφωνα με την τακτική που έχει ορίσει ο χρήστης και τις αλλαγές στο περιβάλλον

λειτουργίας. Το σύστημα είναι πολύ ευέλικτο, επιτρέποντας τη χρήση διαφορετικών NoSQL συστημάτων, τακτικών αλλαγής μεγέθους και φορτίου εισόδου (που εφαρμόζεται στη συστοιχία).

Στις υπάρχουσες μελέτες αξιολόγησης του συστήματος ([10], [12]) έχει επιβεβαιωθεί η ορθή λειτουργία του ακόμα και όταν το φορτίο εισόδου αλλάξει συχνότητα (ως 10 φορές γρηγορότερα) και πλάτος (αύξηση μέχρι 130%) σε σχέση με το φορτίο που ο TIRAMOLA είχε "μάθει" προηγουμένως να χειρίζεται με μια συγκεκριμένη τακτική. Το φορτίο που χρησιμοποιήθηκε σε όλες τις περιπτώσεις είχε όμως μόνο ερωτήματα ανάγνωσης, δηλαδή δεν άλλαζε είδος.

1.2 Αντικείμενο διπλωματικής

Στην παρούσα διπλωματική εργασία θα αξιολογήσουμε την επίδοση του TIRAMOLA όταν το φορτίο που εφαρμόζεται στη συστοιχία δεν έχει μόνο ερωτήματα ανάγνωσης αλλά και εγγραφής. Το NoSQL σύστημα που χρησιμοποιήθηκε ως υπόβαθρο για την κατασκευή της συστοιχίας είναι η υλοποίηση HBase. Για την επιλογή της τακτικής αλλαγής μεγέθους είναι απαραίτητη η μελέτη του τρόπου λειτουργίας και της απόδοσης του συστήματος της HBase με διαφορετικά είδη φορτίου, τα οποία παράγουμε με το εργαλείο YCSB. Αφού επιλέξουμε την τακτική θα ελέγξουμε την ορθή λειτουργία του συστήματος αποφάσεων του TIRAMOLA με φορτία εισόδου που έχουν διαφορετικό ποσοστό ερωτημάτων ανάγνωσης και εγγραφής.

Παράλληλα με την πειραματική αποτίμηση της λειτουργίας του, αντικείμενο της εργασίας είναι και η βελτιστοποίηση της επίδοσης του TIRAMOLA με διαφορετικά είδη φορτίου, τροποποιώντας κατάλληλα το σύστημα αποφάσεων και την τακτική που χρησιμοποιείται.

1.3 Οργάνωση κειμένου

Στο Κεφάλαιο 2 παρουσιάζονται εργασίες σχετικές με το αντικείμενο της διπλωματικής. Αρχικά γίνεται αναλυτική περιγραφή του συστήματος του TIRAMOLA στο Κεφάλαιο 3, με ιδιαίτερη έμφαση στη διαδικασία λήψης αποφάσεων που θα χρειαστεί να τροποποιηθεί στη συνέχεια. Στο Κεφάλαιο 4 περιγράφουμε τον τρόπο λειτουργίας του συστήματος της HBase και του YCSB, τα οποία είναι απαραίτητο να κατανοήσουμε για

να εξηγήσουμε την απόδοση που εμφανίζει η συστοιχία. Στο Κεφάλαιο 5 παραθέτουμε τις ρυθμίσεις που χρησιμοποιήσαμε στην HBase και το YCSB για να πετύχουμε τη μέγιστη δυνατή απόδοση με οποιοδήποτε είδος φορτίου, μαζί με διαγράμματα που δείχνουν την απόδοση της συστοιχίας. Στη συνέχεια παρουσιάζουμε τις προτεινόμενες αλλαγές στον TIRAMOLA για να χειρίζεται τα διαφορετικά είδη φορτίων και τα συγκριτικά αποτελέσματα της αξιολόγησής του. Στο Κεφάλαιο 6 περιγράφεται συνοπτικά η διαδικασία και τα συμπεράσματα της εργασίας και προτείνονται .

Κεφάλαιο 2

Σχετικές εργασίες

Το MET [23] παρέχει αυτοματοποιημένη ελαστικότητα για HBase συστοιχίες με ετερογενή τρόπο ανάλογα με το μοτίβο πρόσβασης. Το MET προσθέτει ή αφαιρεί κόμβους από τη συστοιχία, τους οποίους ρυθμίζει ειδικά ανάλογα με το φορτίο που αναμένει να εξυπηρετήσουν. Ομαδοποιεί τα δεδομένα με παρόμοιο μοτίβο πρόσβασης (δηλαδή είδος φορτίου) και τα τοποθετεί σε κάποιους από τους διαθέσιμους κόμβους της συστοιχίας τους οποίους ρυθμίζει ετερογενώς. Συγκεκριμένα, ανάλογα με το είδος του φορτίου ρυθμίζει το μέγεθος του block δεδομένων και τη μνήμη σε κάθε κόμβο με στόχο τη βελτιστοποίηση της απόδοσης. Τα μετρικά που λαμβάνει υπόψη για την αλλαγή μεγέθους της συστοιχίας είναι η χρήση της CPU, η αναμονή για I/O στο δίσκο και η χρήση της μνήμης, ενώ για τη ρύθμιση των κόμβων χρησιμοποιεί επιπλέον μετρικά όπως ο αριθμός αιτημάτων ανάγνωσης/εγγραφής και ο βαθμός τοπικότητας των δεδομένων σε κάθε κόμβο. Αν και το MET μεγιστοποιεί την απόδοση της συστοιχίας λαμβάνοντας υπόψη και το είδος του φορτίου, όπως στοχεύουμε στη συγκεκριμένη εργασία, βασίζεται σε μια συγκεκριμένη τακτική για την αλλαγή του μεγέθους της συστοιχίας και επιχειρεί να ομαδοποιήσει τα δεδομένα και να τους εφαρμόσει διαφορετικές ρυθμίσεις για την επίτευξη της μέγιστης απόδοσης. Ο TIRAMOLA δεν αλλάζει τις ρυθμίσεις του συστήματος ούτε μετακινεί τα δεδομένα αλλά βελτιστοποιεί την απόδοση βασιζόμενος στην αυξομείωση των πόρων της συστοιχίας.

Τα υπόλοιπα υπάρχοντα συστήματα διαχειρίζονται τους πόρους της συστοιχίας με οποιοδήποτε είδος φορτίου, στοχεύοντας όμως στη βελτιστοποίηση παραγόντων όπως το κόστος, η χρήση των πόρων, κα. χωρίς να λαμβάνουν ρητά υπόψη το είδος του φορτίου στη λήψη της απόφασης. Ενδεικτικά, περιγράφουμε παρακάτω συνοπτικά κάποια από αυτά τα συστήματα.

Η προσέγγιση στο [19] έχει παρόμοια μονάδα λήψης αποφάσεων με τον TIRAMOLA, προσπαθώντας να προσδιορίσει ένα κατάλληλο αριθμό εικονικών μηχανημάτων με σκοπό να μεγιστοποιήσει το κέρδος. Το σύστημα λαμβάνει υπόψη μετρικά υψηλού επιπέδου (π.χ., χρόνος απόκρισης της εφαρμογής) και χρησιμοποιεί μικροοικονομική για να συγκλίνει σε ένα ισοζύγιο για αναλογικό μοίρασμα. Η προσέγγιση που χρησιμοποιείται στον TIRAMOLA είναι πολύ περισσότερο γενικευμένη, καθώς μπορεί

να εφαρμόσει οποιοδήποτε μοντέλο κόστους ή βελτιστοποίησης στη λειτουργικότητά του.

Η υλοποίηση στο [16] παρουσιάζει τακτικές για την ελαστική κλιμάκωση βαθμίδων αποθήκευσης βασισμένων στο Hadoop File System (HDFS) με χρήση αυτομάτου ελέγχου. Η χρήση της CPU καταδεικνύει τον αριθμό των VMs που χρειάζονται για να ικανοποιείται ένας προκαθορισμένος σκοπός (π.χ., ο χρόνος απόκρισης να είναι μικρότερος από 3 δευτερόλεπτα). Αυτή η προσέγγιση χρησιμοποιεί ολοκληρωμένα συστήματα παρακολούθησης· ο TIRAMOLA μπορεί να βελτιστοποιήσει την διάθεση των πόρων βασιζόμενος σε μια τακτική που είναι ευέλικτα ορισμένη από τον χρήστη και ένα πλούσιο σύνολο συλλεγμένων μετρικών.

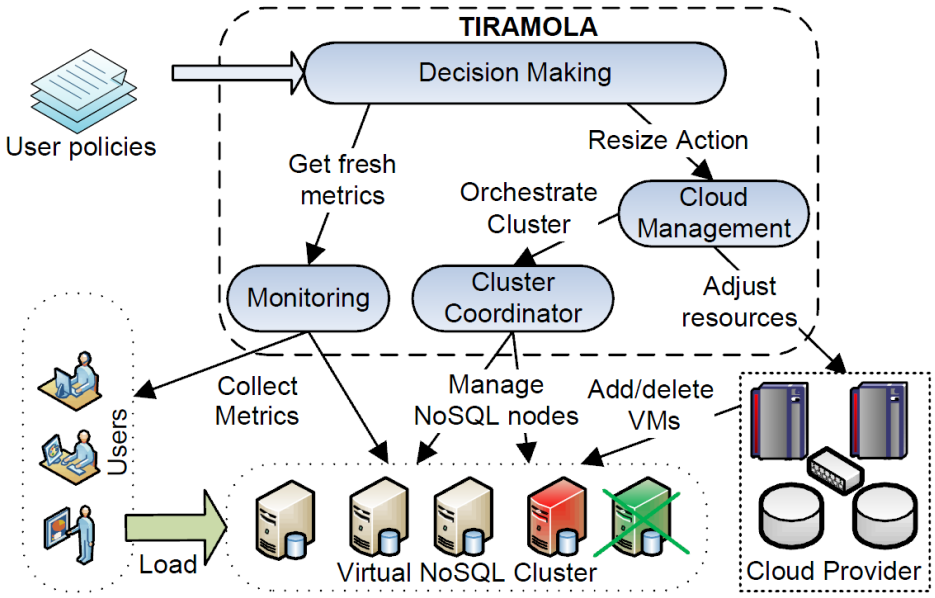
Το Starfish [17] βασίζεται επίσης πάνω στο HDFS και προσδιορίζει το μέγεθος των απαιτούμενων πόρων για την εκτέλεση MapReduce εργασιών με στόχο την ελαχιστοποίηση του χρόνου εκτέλεσης και του κόστους. Χρησιμοποιεί ένα σύστημα που ονομάζεται Elastisizer στο οποίο οι χρήστες μπορούν να εκφράσουν σε μορφή ερωτήματος το πρόβλημα του προσδιορισμού του μεγέθους της συστοιχίας. Ο Elastisizer λύνει το πρόβλημα για κάθε συγκεκριμένη εργασία χρησιμοποιώντας μια μηχανή προβλέψεων που συνδυάζει job-profiling και προσομοίωση για την εκτίμηση του χρόνου εκτέλεσης.

Στο Nefeli [18], οι χρήστες παρέχουν στοιχεία για το είδος της εφαρμογής, επιτρέποντας στο πρόγραμμα να τροποποιήσει τις τακτικές προγραμματισμού για να βελτιώσει την απόδοση της εφαρμογής. Όμως είναι αναγκαίο να είναι εγκατεστημένο ένα ενδιάμεσο λογισμικό στο στρώμα διαχείρισης του νέφους, ενώ στον TIRAMOLA χρησιμοποιούνται μόνο εργαλεία χρήστη για το νέφος χωρίς να γνωρίζουμε απαραίτητα τα εσωτερικά στοιχεία της πλατφόρμας διαχείρισης νέφους.

Κεφάλαιο 3

Αρχιτεκτονική του TIRAMOLA

Ο TIRAMOLA [5], [10], [12] είναι ένα project ανοιχτού κώδικα που δημιουργήθηκε για να διευκολύνει την αυτόματη εκχώρηση πόρων για NoSQL συστοιχίες υπολογιστών (clusters). Υποστηρίζονται οι πιο γνωστές NoSQL υλοποιήσεις, όπως οι Cassandra, HBase, Riak και Voldemort. Στο παρακάτω σχήμα παρουσιάζεται γραφικά η αρχιτεκτονική του TIRAMOLA.



Σχήμα 1: Η αρχιτεκτονική του TIRAMOLA

Η μονάδα Λήψης Αποφάσεων (*Decision Making* module) ενσωματώνει την πολιτική του χρήστη (user policy) που καθορίζεται από μια συνάρτηση ανταμοιβής, καθώς και διάφορα μετρικά (metrics) που καταγράφονται από την πλευρά της συστοιχίας και των χρηστών και αποφασίζει περιοδικά την εκτέλεση κάποιας ενέργειας για την αλλαγή του μεγέθους της συστοιχίας. Στέλνει αυτές τις ενέργειες αλλαγής μεγέθους (resize actions) στη μονάδα Διαχείρισης του Νέφους (*Cloud Management* module) η οποία επικοινωνεί με τον πάροχο υπηρεσιών νέφους (cloud provider) ώστε να απελευθερώσει ή να δεσμεύσει περισσότερες εικονικές μηχανές (VMs). Ο Συντονιστής της Συστοιχίας (*Cluster Coordinator*) είναι μετά υπεύθυνος για την κατάστρωση και εκτέλεση των

σχετικών εντολών προσθήκης και αφαίρεσης που αφορούν ειδικά τη NoSQL συστοιχία που χρησιμοποιείται. Η μονάδα Παρακολούθησης (*Monitoring* module) διατηρεί ενημερωμένα μετρικά επίδοσης που συλλέγονται από τους κόμβους της συστοιχίας και των χρηστών.

Στα πλαίσια αυτής της εργασίας μελετήσαμε τη λειτουργία της μονάδας Λήψης Αποφάσεων καθώς και των απαραίτητων στοιχείων για τη λειτουργία της, δηλαδή τη μονάδα Παρακολούθησης και την ορισμένη από το χρήστη πολιτική, τα οποία περιγράφουμε λεπτομερώς στις επόμενες 2 ενότητες.

3.1 Μονάδα Λήψης Αποφάσεων

Η μονάδα αυτή αποφασίζει την κατάλληλη ενέργεια αλλαγής μεγέθους της συστοιχίας ανάλογα με την επίδοσή της, όπως την αντιλαμβάνεται ο χρήστης, με βάση την πολιτική που ορίζει ο χρήστης. Ο TIRAMOLA συνθέτει αυτή τη διαδικασία ως μια *Μαρκοβιανή διαδικασία λήψης αποφάσεων (MDP)* που συνεχώς προσδιορίζει την πιο επικερδή ενέργεια σχετικά με την τωρινή κατάσταση του συστήματος. Τα κέρδη (βραχυπρόθεσμα και μακροπρόθεσμα) ορίζονται μέσω μιας συνάρτησης ανταμοιβής που εκφράζει την πολιτική αλλαγής μεγέθους που κάθε χρήστης επιθυμεί να τηρεί. Τα μετρικά που χρησιμοποιούνται για τον ορισμό αυτής της συνάρτησης προέρχονται από την κατάσταση της συστοιχίας (αριθμός VMs, χρήση CPU, διαθέσιμη μνήμη, κ.α.) και το εφαρμοσμένο φορτίο (μέσος χρόνος αναμονής και ρυθμός εξυπηρέτησης των ερωτημάτων). Τα μετρικά που αφορούν το φορτίο δεν περιλαμβάνουν πληροφορία για το είδος των ερωτημάτων (ποσοστό αιτημάτων ανάγνωσης, εισαγωγής, ενημέρωσης, κ.α.) παρά μόνο για τον ρυθμό εξυπηρέτησής τους και τον χρόνο αναμονής για να ολοκληρωθούν, επομένως για τη λήψη της απόφασης δεν λαμβάνεται υπόψη το είδος του φορτίου. Όταν ληφθεί η απόφαση αλλαγής μεγέθους η μονάδα προωθεί την εντολή στην μονάδα Διαχείρισης του Νέφους.

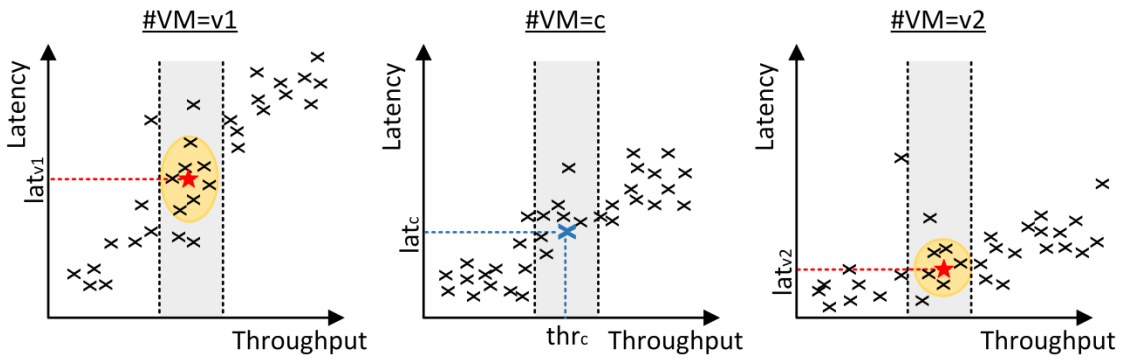
3.1.1 Διαδικασία λήψης αποφάσεων

Έστω ότι οι διαθέσιμες καταστάσεις είναι $s = (s_1, s_2, \ldots, s_M)$, όπου s_i αναπαριστά μια συστοιχία με $i \in [1, M]$ NoSQL κόμβους (VMs) σε λειτουργία. Η συνάρτηση ανταμοιβής $r(s)$ αναπαριστά σε αριθμητικό επίπεδο το πόσο "καλό" είναι να είμαστε στην

κατάσταση s. Γενικά $r(\mathrm{s}) = f(\text{κέρδη}, \text{κόστη})$, δηλαδή η πολιτική του χρήστη ορίζεται ενσωματώνοντας κατάλληλα διάφορα μετρικά στη συνάρτηση ανταμοιβής. Για παράδειγμα, ο υψηλός ρυθμός εξυπηρέτησης ερωτημάτων μπορεί να είναι επικερδής ενώ ο χρόνος αναμονής που παραβιάζει τη Συμφωνία Επιπέδου Εξυπηρέτησης (SLA), το κόστος ανά VM, κ.α. μπορούν να θεωρούνται επιζήμια. Ένα παράδειγμα συνάρτησης ανταμοιβής $r(\mathrm{s})$ που λαμβάνει υπόψη τον ρυθμό εξυπηρέτησης (*thr*) και τον χρόνο αναμονής (*lat*) εκτός από το κόστος (*C*) ανά VM που λειτουργεί, είναι η εξής: $r(\mathrm{s}) = B \cdot thr - C \cdot |\mathrm{VMs}| - D \cdot lat$, όπου B,C,D κατάλληλα επιλεγμένες σταθερές. Η συνάρτηση ανταμοιβής είναι ένα σημαντικό κομμάτι της σύνθεσης, καθώς επιτρέπει διαφορετικές πολιτικές αλλαγής μεγέθους ανά πάροχο χρησιμοποιώντας διαφορετικές συναρτήσεις ανταμοιβής.

Στον TIRAMOLA υλοποιούμε μια αυξητική, γραμμική μέθοδο που εκτελείται στη μονάδα Λήψης Αποφάσεων για να υπολογιστούν οι τιμές $r(\mathrm{s})$, την οποία περιγράφουμε ακολούθως: Τα δεδομένα που συλλέγονται από τη μονάδα Παρακολούθησης συγκροτούν ένα d-διάστατο dataset, όπου $\mathrm{d} \geq 2$. Οι διαστάσεις που συμπεριλαμβάνονται εξ' ορισμού είναι ο ρυθμός εξυπηρέτησης (*thr*) και το μέγεθος της συστοιχίας (αριθμός των VMs). Κάθε μετρικό που συμπεριλαμβάνεται στη συνάρτηση ανταμοιβής προστίθεται επιπλέον στις διαστάσεις του μοντέλου μας. Καθώς λειτουργεί η συστοιχία, ένας d-διάστατος πίνακας συμπληρώνεται, προσθέτοντας ένα σημείο σε κάθε χρονικό σημείο καταγραφής. Όταν έρθει η ώρα να ληφθεί μια απόφαση, το σύστημα πρέπει να υπολογίσει το $r(\mathrm{s}')$ για κάθε κατάσταση s' που επιτρέπεται να πάμε από την τωρινή κατάσταση s. Η μέθοδός μας βασίζεται στην αρχή ότι το σύστημα δρα με ένα προβλέψιμο τρόπο, δηλαδή συμπεριφέρεται με τον ίδιο τρόπο σε παρόμοιες συνθήκες. Έτσι, πραγματοποιούμε ομαδοποίηση σε συστάδες (clustering) των (d-1)-διάστατων datasets (αποκλείοντας τη διάσταση για τον αριθμό των VMs) που αντιστοιχούν στις καταστάσεις $\{\mathrm{s}'\}$. Όμως, δεν συμπεριλαμβάνονται όλα τα data points: Ακολουθώντας τη αρχή μας, μας ενδιαφέρουν μετρήσεις που είναι σχετικές με το τωρινό φορτίο στη συστοιχία. Έτσι, υπολογίζοντας και ένα βαθμό μεταβλητότητας, επιτρέπουμε μόνο τα σημεία μέσα σε μία συγκεκριμένη "λωρίδα" τιμών γύρω από την τωρινή μέτρηση του φορτίου να τροφοδοτηθούν στο μηχανισμό ομαδοποίησης.

Το κεντροειδές της μεγαλύτερης συστάδας (cluster) (αν ο αριθμός των συστάδων είναι $k>1$) μπορεί να χρησιμοποιηθεί σαν ένα αντιπροσωπευτικό σημείο του οποίου οι συντεταγμένες θα χρησιμοποιηθούν για τον υπολογισμό του $r(s')$.



Σχήμα 2: Επιλογή αντιπροσωπευτικών σημείων για τον υπολογισμό των *r*(v1) και *r*(v2)

Στο Σχήμα 2 φαίνεται ένα παράδειγμα αυτής της διαδικασίας. Υποθέτουμε ότι $r(s) = f(throughput, latency, \text{VMs})$. Τα τρία διαγράμματα αναπαριστούν τρία 2-διάστατα datasets, ένα για κάθε μέγεθος της συστοιχίας: v1, v2, και c (το τωρινό μέγεθος), με $v1 < c < v2$. Όλα τα data-points που έχουν συλλεχθεί μέχρι στιγμής σημειώνονται με ένα 'x'. Όταν το σύστημα αποφασίζει για την πιθανή αλλαγή μεγέθους, η πιο πρόσφατη μέτρηση σημειώνεται με ένα μπλε 'x', σηματοδοτώντας ότι η συστοιχία επιδεικνύει χρόνο αναμονής lat_c όταν ο ρυθμός εξυπηρέτησης είναι ίσος με thr_c. Για τον υπολογισμό των *r*(v1) και *r*(v2) ομαδοποιούμε τα data points μέσα σε γκρι περιοχές (τις αντίστοιχες thr-λωρίδες) και επιλέγουμε τις y-συντεταγμένες των κεντροειδών (lat_{v1}, lat_{v2}). Γενικά, διαλέγουμε αντιπροσωπευτικά σημεία σε p datasets (d-1) διαστάσεων, όπου p είναι ο αριθμός των επιτρεπτών μεταβάσεων από την τωρινή κατάσταση. Για κάθε dataset, μόνο τα data points που βρίσκονται μέσα σε ένα (d-1)-υπερκύβο τροφοδοτούνται στο μηχανισμό ομαδοποίησης. Χρησιμοποιώντας τα επιλεγμένα αντιπροσωπευτικά σημεία υπολογίζονται για κάθε πιθανή μελλοντική κατάσταση τα προσδοκώμενα κέρδη (δηλαδή η τιμή της συνάρτησης ανταμοιβής) και επιλέγεται τελικά η πιο επικερδής κατάσταση σε σχέση με την τωρινή.

Η χρησιμοποιούμενη προσέγγιση με χρήση MDP έχει το πλεονέκτημα ότι δεν απαιτεί μοντέλο του περιβάλλοντος ενώ είναι υλοποιημένη με μια μέθοδο με ελάχιστες υπολογιστικές απαιτήσεις. Επομένως, το σύστημα μπορεί να μάθει άμεσα από εμπειρία

χωρίς μοντέλο της δυναμικής του περιβάλλοντος. Η εκμάθηση είναι πραγματικού χρόνου και συνεχίζει καθ' όλη τη διάρκεια ζωής του συστήματος.

Ο TIRAMOLA χρησιμοποιεί περιόδους εκπαίδευσης (training sessions) για να επιταχύνει τη σύγκλιση και να αυξήσει την ακρίβεια της εκτίμησης του $r(s)$. Η εκπαίδευση είναι σημαντική ειδικά στα πρώτα στάδια της διαδικασίας εκμάθησης (στην αρχική φάση λειτουργίας του) για να διασφαλιστεί η σωστή συμπεριφορά με ελάχιστη ή καθόλου εμπειρία.

3.2 Μονάδα Παρακολούθησης

Ο TIRAMOLA χρησιμοποιεί το Ganglia [13] ένα κατανεμημένο κλιμακώσιμο εργαλείο παρακολούθησης που επιτρέπει την απομακρυσμένη συλλογή των τρεχόντων ή ιστορικών στατιστικών της συστοιχίας (όπως μέση χρήση CPU, δικτύου, μνήμης ή χώρου στο δίσκο, αριθμό ανοιχτών νημάτων στους χρήστες, κ.α.) μέσω ενός XML API. Εκτός από τα μετρικά των εξυπηρετητών της συστοιχίας, έγιναν ορισμένες τροποποιήσεις που επιτρέπουν στο Ganglia να συλλέγει μετρικά σε σχέση με το χρήστη. Αυτό είναι απαραίτητο, καθώς η κατάσταση του συστήματος μπορεί να εξαρτάται και από πληροφορίες για το χρήστη όπως ο μέσος χρόνος αναμονής και ο ρυθμός εξυπηρέτησης των ερωτημάτων. Για να το καταφέρουμε τροποποιήσαμε τους χρήστες μας ώστε ο καθένας να αναφέρει τα μετρικά του χρησιμοποιώντας μια γνωστή λειτουργία του Ganglia που ονομάζεται `gmetric spoofing`. Με αυτό τον τρόπο η μονάδα παρακολούθησης τροφοδοτεί τη μονάδα λήψης αποφάσεων με την τρέχουσα, ενημερωμένη κατάσταση του συστήματος, λαμβάνοντας υπόψη μετρικά και από τη μεριά του χρήστη και από τη μεριά των εξυπηρετητών.

Κεφάλαιο 4

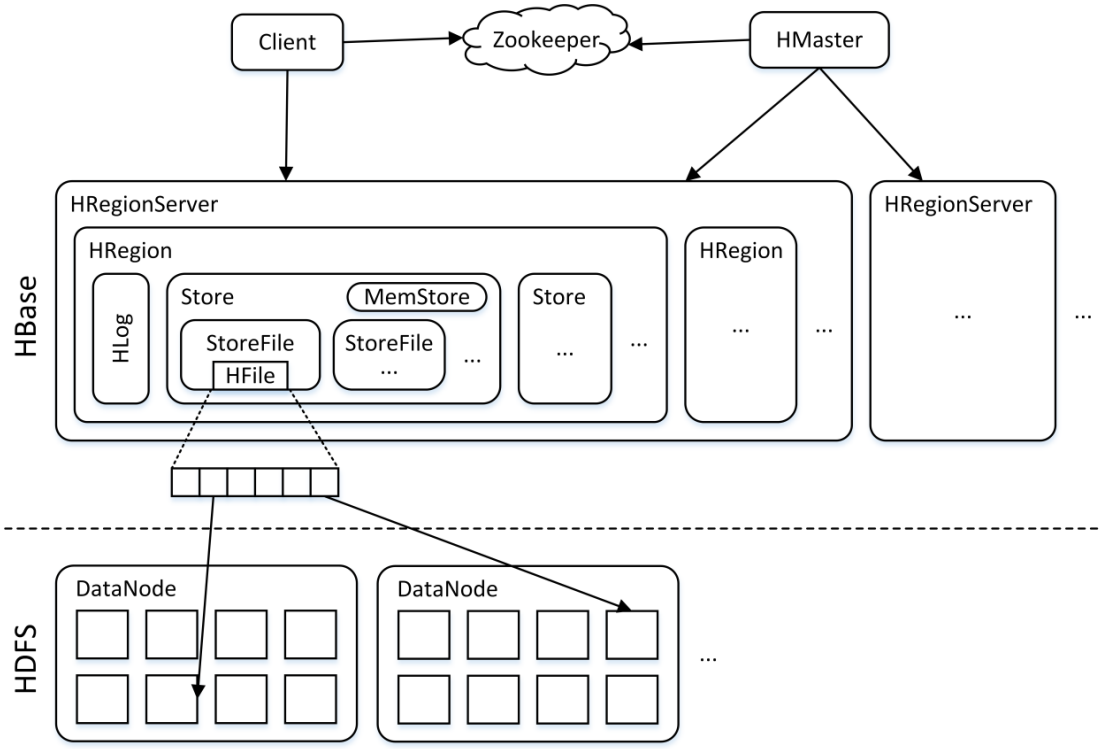
Περιγραφή υποσυστημάτων

4.1 Apache HBase

Η HBase [3],[8],[9] είναι ένα αυστηρά συνεκτικό NoSQL σύστημα αποθήκευσης δεδομένων με σχεδόν βέλτιστη επίδοση κατά τις εγγραφές και τελειοποιημένη επίδοση στις αναγνώσεις. Αποθηκεύει τα δεδομένα αραιά, με τρόπο προσανατολισμένο στις στήλες και όχι στις γραμμές όπως στις πιο συνήθεις βάσεις δεδομένων, ενώ διατηρεί πολλαπλές εκδόσεις για κάθε τιμή καταγράφοντας πως αυτή αλλάζει με το χρόνο, με τη χρήση χρονοσφραγίδων (timestamps).

Για την αποθήκευση των δεδομένων η HBase βασίζεται στο Hadoop Distributed File System (HDFS) [2], [22], ένα σταθερό και αξιόπιστο κατανεμημένο σύστημα αρχείων που παρέχει ένα κλιμακώσιμο στρώμα αποθήκευσης με αντίγραφα (replicas) και είναι ανεκτικό σε σφάλματα. Έτσι παρέχει υψηλού ρυθμού πρόσβαση σε δεδομένα μεγάλου όγκου ενώ εξασφαλίζει την διατήρηση των δεδομένων σε περιπτώσεις αστοχίας υλικού εφαρμόζοντας αντιγραφή (replication) των αλλαγών σε ένα ρυθμιζόμενο αριθμό φυσικών κόμβων.

Στο παρακάτω σχήμα παρουσιάζεται η αρχιτεκτονική της HBase και ο τρόπος με τον οποίο αυτή συνδέεται με το HDFS.



Σχήμα 3: Η αρχιτεκτονική της HBase

Το σύστημα αποτελείται από έναν κύριο εξυπηρετητή (HMaster) και έναν αριθμό από εξυπηρετητές περιοχών (HRegionServer) οι οποίοι εκτελούνται στους διαφορετικούς κόμβους της συστοιχίας. Ο HMaster είναι υπεύθυνος για τον έλεγχο των HRegionServers και εκτελεί τις απαραίτητες διαδικασίες προκειμένου να επιτευχθεί ομοιόμορφη κατανομή του φορτίου (load-balancing) στους διάφορους κόμβους του συστήματος. Οι HRegionServers είναι υπεύθυνοι για την αποθήκευση, διαχείριση και ανάκτηση των δεδομένων από το HDFS μετά από κάποιο αίτημα του χρήστη ή κατά τη διαδικασία της εξισορρόπησης φορτίου (load-balancing). Για το συγχρονισμό των διαφόρων ενεργειών πάνω στα δεδομένα κατά την ταυτόχρονη πρόσβαση για εγγραφή ή ανάγνωση, η HBase απαιτεί την ύπαρξη του Zookeeper [1]. Πρόκειται για ένα αξιόπιστο κατανεμημένο σύστημα επίτευξης ομοφωνίας βασισμένο στο πρωτόκολλο Paxos [20]. Ο Zookeeper είναι απαραίτητος για τον έλεγχο ύπαρξης ενός μόνο HMaster, τον εντοπισμό των HRegionServers, σε ποιον φιλοξενείται η περιοχή -ROOT-, κ.α. οπότε ουσιαστικά εξασφαλίζει τη συνάφεια των δεδομένων την HBase.

Το HDFS διατηρεί τα δεδομένα σε έναν αριθμό DataNodes που εκτελούνται σε ορισμένους κόμβους της συστοιχίας ανεξάρτητα από τους HRegionServers. Δηλαδή ο αριθμός των DataNodes μπορεί να είναι διαφορετικός από τον αριθμό των

HRegionServers και οι DataNodes μπορούν να υλοποιηθούν σε οποιουσδήποτε κόμβους αρκεί όλοι οι HRegionServers να έχουν πρόσβαση σε αυτούς. Επίσης, αντίγραφα των δεδομένων διατηρούνται σε διαφορετικούς DataNodes για την ασφάλεια των δεδομένων.

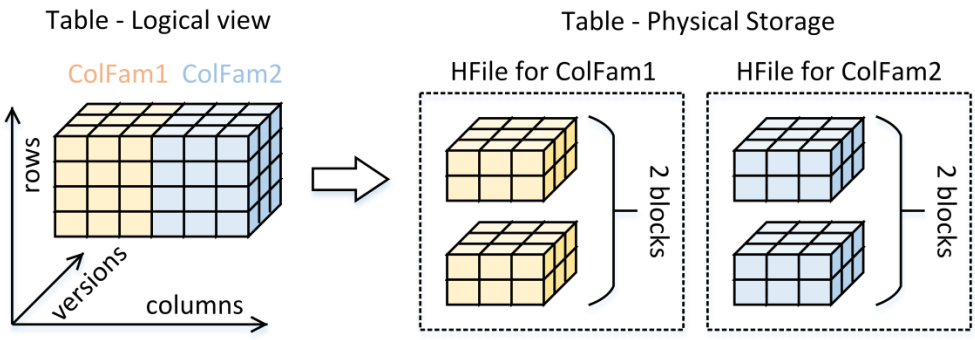
Ακολουθεί πιο αναλυτική περιγραφή της λειτουργίας του συστήματος που απεικονίζεται παραπάνω.

4.1.1 Διαμόρφωση των blocks

Η πρωταρχική μονάδα του συστήματος είναι η στήλη (column), όπως τονίσαμε στην προηγούμενη ενότητα. Μια ή περισσότερες στήλες σχηματίζουν μια γραμμή (row) που δεικτοδοτείται μοναδικά από ένα κλειδί γραμμής (row key). Οι στήλες μιας γραμμής ομαδοποιούνται επιπλέον σε οικογένειες στηλών (column families). Οι γραμμές ταξινομούνται λεξικογραφικά βάσει του row key και ένας αριθμός γραμμών σχηματίζει ένα πίνακα.

Για κάθε κελί (cell) του πίνακα μπορούν να αποθηκευτούν πολλαπλές εκδόσεις του με τη χρήση χρονοσφραγίδων (timestamps), δίνοντας έτσι μια επιπλέον διάσταση στον πίνακα. Κάθε κελί δεικτοδοτείται από ένα κλειδί το οποίο αποτελεί συνδυασμό των row, column και timestamp. Για πιο γρήγορη πρόσβαση στην τελευταία έκδοση κάθε κελιού διατηρείται φθίνουσα σειρά των timestamps.

Οι ταξινομημένες γραμμές ομαδοποιούνται σε blocks μεγέθους 64KB και όταν ένας HRegionServer ζητήσει πρόσβαση ακόμα και σε ένα μόνο κελί στο HDFS, ανακτάται ολόκληρο το block που το περιέχει για αποδοτικότερη λειτουργία I/O. Δηλαδή, η ανάγνωση έστω και ενός Byte δεδομένων έχει σαν αποτέλεσμα την ανάκτηση 64KB δεδομένων από το HDFS.

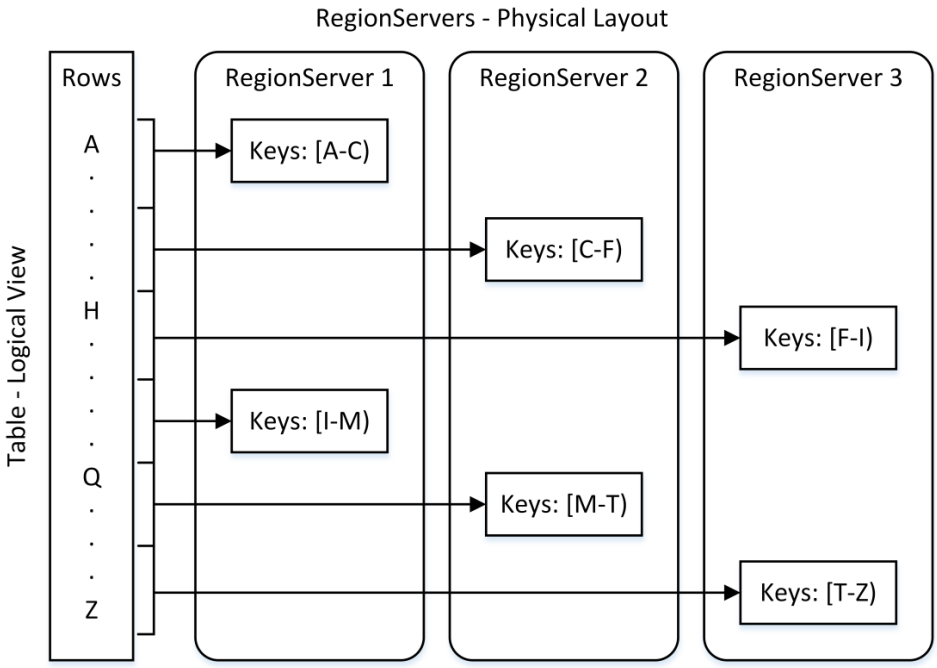


Σχήμα 4: Σχηματισμός των blocks

Για κάθε διαφορετική οικογένεια στηλών δημιουργούνται και διαφορετικά block, ακόμα και εάν αυτά αφορούν τα ίδια κλειδιά γραμμών. Με άλλα λόγια, μια γραμμή στην HBase που περιέχει δεδομένα σε δυο διαφορετικές οικογένειες στηλών θα έχει τα δεδομένα της αποθηκευμένα σε τουλάχιστον δυο διαφορετικά block (Σχήμα 4).

4.1.2 HRegions και HFiles

Ένας αριθμός συνεχόμενων blocks ομαδοποιούνται και σχηματίζουν μια περιοχή, γνωστή και ως HRegion, άρα οι HRegions είναι στην ουσία ομάδες συνεχόμενων γραμμών. Το μέγεθος μιας HRegion είναι εκατοντάδες φορές μεγαλύτερο του μεγέθους των blocks και μια τυπική τιμή είναι 64-128 MB. Κάθε HRegion εξυπηρετείται από έναν HRegionServer ενώ ένας HRegionServer μπορεί να εξυπηρετεί πολλά HRegions.



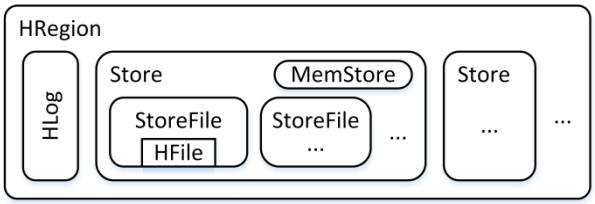
Σχήμα 5: Γραμμές ομαδοποιημένες σε περιοχές που εξυπηρετούνται από διαφορετικούς εξυπηρετητές

Τα δεδομένα των HRegions αποθηκεύονται στο δίσκο σε χαμηλού-επιπέδου αρχεία αποθήκευσης (StoreFiles), που ονομάζονται HFiles. Τα HFiles εσωτερικά περιέχουν μια σειρά από blocks καθώς και ένα ευρετήριο για αυτά τα blocks με σκοπό την αποδοτική αναζήτηση στα δεδομένα των block αυτών. Η HBase αποθηκεύει τα HFiles στο HDFS, μοιράζοντας τα blocks τους σε HDFS blocks, μεγέθους μερικών εκατοντάδων MB.

4.1.3 Store, MemStore και flush

Κάθε HRegion αποθηκεύει, όπως αναφέραμε προηγουμένως, ένα σύνολο γραμμών. Τα δεδομένα αυτών των γραμμών μπορεί να ανήκουν σε διαφορετικές column families (Σχήμα 4) και οργανώνονται σε ξεχωριστές δομές που ονομάζονται Stores.

Τα δεδομένα μιας συγκεκριμένης column family ανήκουν σε ένα Store, δηλαδή μια HRegion έχει ένα Store για κάθε column family. Κάθε Store περιέχει τα σχετικά HFiles καθώς και ένα κομμάτι χώρου στη μνήμη που ονομάζεται MemStore.



Σχήμα 6: Stores σε μια HRegion

Κατά την εγγραφή δεδομένων (εισαγωγή ή ενημέρωση με εγγραφή νεότερης έκδοσης ενός ή περισσοτέρων κελιών) σε ένα Store ο HRegionServer αποθηκεύει τα νέα κελιά με τις τιμές τους στο MemStore του συγκεκριμένου Store. Μόνο όταν σε ένα Store γεμίσει το MemStore γράφονται τα περιεχόμενά του σε ένα νέο HFile στο συγκεκριμένο Store, αποφεύγοντας με αυτό τον τρόπο τη συνεχόμενη δημιουργία πολλών μικρών αρχείων στο δίσκο. Η διαδικασία εγγραφής των δεδομένων του MemStore στο δίσκο ονομάζεται flush και το μέγιστο μέγεθος ενός MemStore πριν γίνει flush στο δίσκο είναι 128MB.

Τα δεδομένα αποθηκεύονται στο MemStore ταξινομημένα σε δομές παρόμοιες με αυτές των HFiles, χωρίς να υπάρχει περιορισμός στον αριθμό των εκδόσεων που αποθηκεύονται. Έτσι όταν χρειαστεί να γίνει flush του MemStore στο δίσκο διατηρείται ο επιθυμητός αριθμός των τελευταίων εκδόσεων, απορρίπτοντας τις πιο παλιές, και γίνεται απευθείας μετατροπή σε HFile.

4.1.4 BlockCache

Καθώς η HBase ανακτά δεδομένα κατά την ανάγνωση κελιών, διαβάζει όπως είπαμε προηγουμένως ολόκληρα blocks δεδομένων από τα HFiles και διατηρεί αυτά τα blocks σε ένα κομμάτι μνήμης που χρησιμοποιείται ως LRU cache, τη λεγόμενη BlockCache, έτσι ώστε οι επόμενες αναγνώσεις να μην χρειάζονται συνέχεια λειτουργίες I/O στο δίσκο. Κάθε HRegionServer διατηρεί τη δική του BlockCache για να αποθηκεύει τα αναγνωσμένα blocks από τις HRegions που εξυπηρετεί.

Ένα block στη BlockCache θεωρείται μη-έγκυρο (invalid) όταν διαφοροποιηθεί το αρχικό block στο δίσκο, όταν δηλαδή αποθηκευτεί στο δίσκο μια νέα έκδοση κάποιου

από τα κελιά που περιέχει ή δημιουργηθεί ένα νέο κελί στο συγκεκριμένο block. Τα περιεχόμενα της BlockCache δεν επηρεάζονται επομένως όταν αποθηκεύονται ενημερωμένες εκδόσεις ή νέα κελιά στο MemStore παρά μόνο όταν γίνει flush αυτού του MemStore στο δίσκο. Σε αυτή την περίπτωση, πρέπει στην επόμενη ανάγνωση να μεταφερθεί πάλι το block από το δίσκο.

4.1.5 Διαφορές και ομοιότητες των BlockCache και MemStore

Σε κάθε κόμβο που λειτουργεί ένας HRegionServer δεσμεύεται στη μνήμη χώρος για τη BlockCache και για τα MemStores των HRegions που εξυπηρετεί. Η ύπαρξη της BlockCache και των MemStores στη μνήμη του συστήματος είναι πολύ σημαντική για την αποδοτική λειτουργία της HBase. Η BlockCache χρησιμοποιείται στην ουσία ως μια cache για την ανάγνωση δεδομένων, ενώ τα MemStores χρησιμοποιούνται ως buffers για την εγγραφή δεδομένων. Με τη χρήση τους επιτυγχάνεται η εκτέλεση λιγότερων λειτουργιών I/O στο δίσκο και η αύξηση της απόδοσης της HBase.

4.1.6 Splitting και compaction

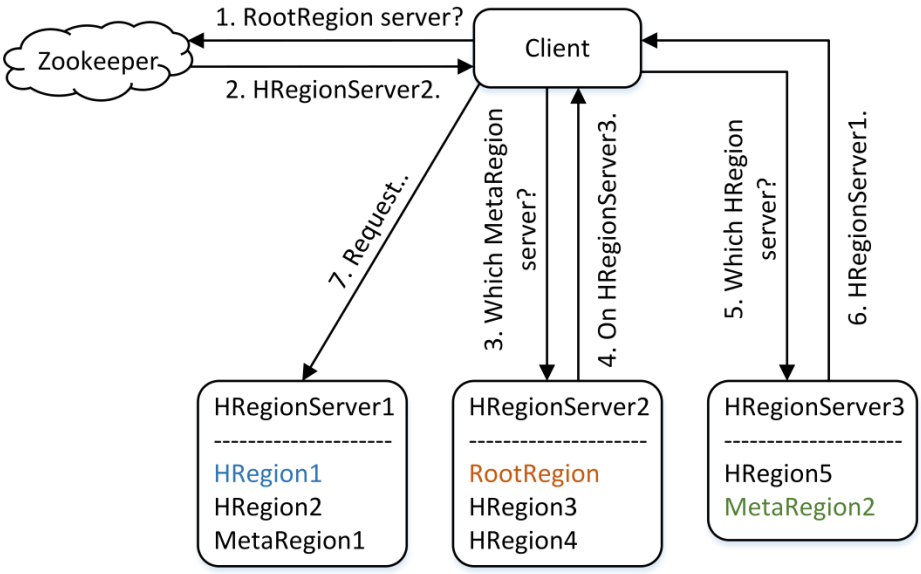
Φυσικά όσο περισσότερα δεδομένα είναι αποθηκευμένα στη βάση τόσα περισσότερα HRegions δημιουργούνται. Κατά την εισαγωγή δεδομένων, ο αριθμός των HRegions αυξάνεται σταδιακά καθώς αυξάνεται και ο όγκος των αποθηκευμένων δεδομένων, με τον τρόπο που περιγράφουμε ακολούθως. Όπως αναφέραμε στην ενότητα 4.1.3 τα νέα δεδομένα αποθηκεύονται αρχικά σε MemStores, τα οποία όταν κατά περιόδους γεμίζουν, γράφουν τα περιεχόμενά τους σε νέα HFiles στα αντίστοιχα Stores. Σε ένα Store ο αριθμός των HFiles αυξάνεται σταδιακά, καθώς δημιουργούνται νέα από το επαναλαμβανόμενο flush του MemStore, και όταν γίνουν αρκετά ενώνονται σε λιγότερα HFiles μεγαλύτερου μεγέθους. Αυτό συνεχίζεται μέχρι κάποιο HFile να ξεπεράσει το προκαθορισμένο μέγιστο μέγεθος για τα HFiles, οπότε γίνεται το λεγόμενο splitting και η HRegion που το περιέχει χωρίζεται σε δύο HRegions.

Η διαδικασία συνένωσης μερικών μικρότερων HFiles σε ένα μεγαλύτερο ονομάζεται compaction. Υπάρχουν δύο είδη compaction: το major compaction και το minor compaction. Με το minor compaction ενώνονται 3 τουλάχιστον (ως 10 το πολύ) HFiles που έχουν μέγεθος κάτω από 128MB (προτεραιότητα έχουν αυτά με την παλαιότερη ημερομηνία δημιουργίας) σε ένα νέο HFile. Κάθε φορά που εκτελείται flush ελέγχεται αν είναι απαραίτητη η εκτέλεση minor compaction στο εν λόγω Store. Το major compaction

γίνεται κάθε 24 ώρες σε όλα τα Stores, εφόσον είναι απαραίτητο, και συμπεριλαμβάνει όλα τα HFiles ενός Store, δημιουργώντας τελικά ένα HFile σε κάθε Store.

4.1.7 Βασική ροή επικοινωνίας

Για να μπορούν οι πελάτες να βρίσκουν τον HRegionServer που εξυπηρετεί ένα σύνολο κλειδιών η HBase παρέχει δύο ειδικούς πίνακες καταλόγου (catalog tables), που ονομάζονται -ROOT- και .META.. Ο σχεδιασμός της απαιτεί να υπάρχει πάντα μόνο μία περιοχή για τον πίνακα -ROOT-.



Σχήμα 7: Διαδικασία εύρεσης του ζητούμενου από τον πελάτη row key

Όταν ένας πελάτης επιθυμεί πρόσβαση σε ένα συγκεκριμένο κλειδί ή μια ομάδα κλειδιών είτε για εγγραφή είτε για ανάγνωση, αρχίζει την αναζήτηση με βάση το row key εφόσον η ταξινόμηση στη βάση γίνεται με βάση αυτά. Για να αποκτήσει λοιπόν πρόσβαση σε μια συγκεκριμένη γραμμή επικοινωνεί πρώτα με τον Zookeeper και ανακτά το όνομα (hostname) του εξυπηρετητή της -ROOT- περιοχής. Κατόπιν, επικοινωνεί με τον συγκεκριμένο HRegionServer που διαθέτει το ROOT table και από εκεί βρίσκει τον HRegionServer που εξυπηρετεί την περιοχή του πίνακα .META. που έχει πληροφορίες για το ζητούμενο row key. Τέλος ανακτά από τον HRegionServer της επιθυμητής .META. περιοχής το όνομα του εξυπηρετητή της περιοχής που περιέχει το ζητούμενο row key. Στη συνέχεια ο πελάτης επικοινωνεί απευθείας με τον ζητούμενο HRegionServer και ο

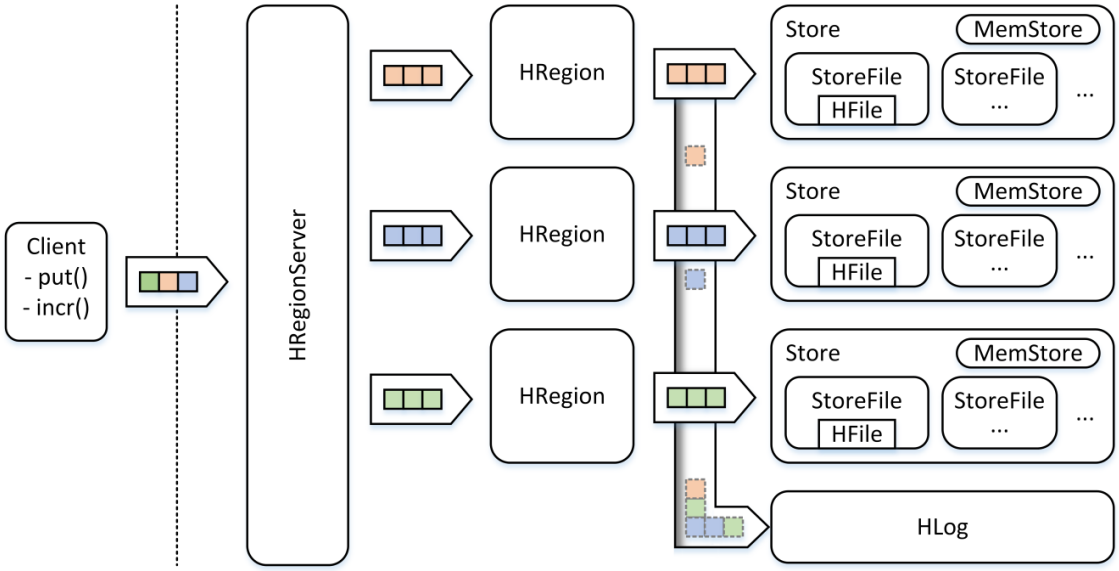
τελευταίος ανοίγει τη περιοχή και δημιουργεί το αντίστοιχο HRegion και ένα Store για κάθε column family, όπως φαίνονται στο Σχήμα 3. Το Store δεσμεύει επίσης ένα MemStore, ως χώρο στη μνήμη.

4.1.8 Διαδικασία ενημέρωσης δεδομένων

Όταν ένας πελάτης στείλει σε ένα HRegionServer ένα αίτημα ενημέρωσης, αυτός προωθεί το αίτημα στο ανάλογο HRegion. Αρχικά, αποθηκεύονται οι αλλαγές στα δεδομένα σε ένα αρχείο καταγραφής που ονομάζεται write-ahead log (WAL), το οποίο βρίσκεται στο HDFS και είναι κοινόχρηστο σε όλα τα HRegions του HRegionServer (αναπαριστάται με το HLog στα σχήματα 3 και 6). Στη συνέχεια γράφονται τα δεδομένα στο MemStore και τότε ενημερώνεται ο πελάτης ότι το αίτημά του ολοκληρώθηκε επιτυχώς.

Βέβαια όσο τα δεδομένα μένουν στη μνήμη δεν είναι ασφαλή καθώς μπορεί να χαθούν αν ο εξυπηρετητής τεθεί εκτός λειτουργίας. Αυτό το πρόβλημα λύνεται με τη χρήση του WAL, που κάνει πρώτα από όλα ένα ασφαλές αντίγραφο του αιτήματος στο HDFS, για αυτό και είναι προϋπόθεση η επιτυχημένη εγγραφή στο WAL ώστε να θεωρηθεί ότι ολοκληρώθηκε επιτυχώς το αίτημα. Να υπενθυμίσουμε επίσης ότι το HDFS είναι replicated σύστημα αρχείων άρα έχουμε ένα ακόμα επίπεδο ασφάλειας ώστε ακόμα και αν ο συγκεκριμένος HRegionServer τεθεί εκτός λειτουργίας για μεγάλο χρονικό διάστημα, να μπορεί κάποιος άλλος που έχει αντίγραφο του WAL να “ξανατρέξει” τις αλλαγές που ζητήθηκαν.

Στο παρακάτω σχήμα παρουσιάζεται γραφικά το σύστημα που περιγράψαμε.



Σχήμα 8: Το WAL στην αρχιτεκτονική της HBase

Όταν το MemStore γεμίσει γίνεται flush, δηλαδή τα περιεχόμενά του γράφονται σε νέο HFile στο συγκεκριμένο Store, όπως περιγράψαμε στην ενότητα 4.1.3. Κατά τη διάρκεια του flush το σύστημα μπορεί να συνεχίσει να εξυπηρετεί τους πελάτες χωρίς να χρειάζεται να τους μπλοκάρει, καθώς ένα νέο/άδειο κομμάτι μνήμης δεσμεύεται από το MemStore ενώ το παλιό/γεμάτο κομμάτι του MemStore μετατρέπεται σε αρχείο. Μετά το flush, οι σχετικές καταχωρήσεις στο WAL φυσικά διαγράφονται.

Η εγγραφή αιτημάτων στο WAL γίνεται σειριακά, όπως αυτά καταφτάνουν, και το replication σε άλλους κόμβους γίνεται πάντα ασύγχρονα. Επομένως, το WAL δεν περιέχει ταξινομημένες εγγραφές, και μπορούμε να το θεωρήσουμε παρόμοιο με το journal ενός journaling file system (π.χ. το ext3) [21]. Ο χρόνος αναμονής για την ολοκλήρωση ενός αιτήματος ενημέρωσης διαμορφώνεται έτσι μόνο από τις καθυστερήσεις εγγραφής στο τοπικό WAL και το MemStore.

Να σημειώσουμε επίσης ότι μέχρι να ολοκληρωθεί το αίτημα εγγραφής και σταλεί η σχετική επιβεβαίωση στον πελάτη αυτός διατηρεί το row lock, δηλαδή κλειδώνει όλα τα κελιά της συγκεκριμένης οικογένειας στηλών στη σχετική γραμμή για να εξασφαλίσει την ατομικότητα της λειτουργίας. Όσο διαρκεί αυτό το κλείδωμα, τα αιτήματα ανάγνωσης που αφορούν αυτά τα δεδομένα μπλοκάρονται και περιμένουν σε μια ουρά για να εξυπηρετηθούν.

4.1.9 Διαδικασία ανάγνωσης δεδομένων

Για να διαβάσει η HBase ένα block από το HDFS απαιτείται να βρει τη θέση του ανατρέχοντας απλά στα ευρετήρια των HFiles. Στη συνέχεια αποθηκεύει στη BlockCache εκτός από το ζητούμενο block και το ευρετήριο του HFile που άνοιξε. Σε αυτό το block πρέπει να διατρέξει τη γραμμή με το σχετικό row key για να βρει το ζητούμενο κλειδί. Φυσικά αν υπάρχουν αποθηκευμένες πολλές εκδόσεις ενός κελιού σε κάποιο block, επιστρέφεται πάντα η τελευταία έκδοση στον πελάτη.

Όταν ένας πελάτης στείλει σε ένα HRegionServer ένα αίτημα ανάγνωσης, αυτός προωθεί κατά τα γνωστά το αίτημα στο ανάλογο HRegion. Αρχικά γίνεται μια δυαδική αναζήτηση στα ευρετήρια που διατηρούνται στην μνήμη (BlockCache) για την εύρεση του block που πιθανώς περιέχει το ζητούμενο κλειδί. Αν δεν υπάρχει το block στη BlockCache, ή δεν περιέχει έγκυρα δεδομένα, γίνεται ανάγνωση του block από τα HFiles απευθείας από το HDFS. Για την εύρεση της επιθυμητής τελευταίας έκδοσης γίνεται αναζήτηση στη σχετική γραμμή του block αλλά και στα δεδομένα του MemStore, όπου μπορεί να υπάρχει νεότερη έκδοση, και επιστρέφεται το συνδυασμένο αποτέλεσμα. Συνοπτικά, η HBase ψάχνει επομένως στη BlockCache, στα HFiles εφόσον είναι απαραίτητο, και στο MemStore (εάν αυτό έχει δεδομένα).

Ο χρόνος αναμονής σε ένα αίτημα ανάγνωσης είναι σημαντικά πιο μεγάλος από αυτόν ενός αιτήματος ενημέρωσης. Αυτό συμβαίνει γιατί χρειάζεται στην καλύτερη περίπτωση αναζήτηση μόνο στη BlockCache, ενώ στη χειρότερη περίπτωση χρειάζεται μεταφορά δεδομένων από το δίσκο και αναζήτηση στο MemStore και στη BlockCache. Σε αντίθεση με τα αιτήματα ενημέρωσης όμως, ο πελάτης δε χρειάζεται να διατηρεί row locks για την ανάγνωση (πολλοί πελάτες μπορούν να διαβάζουν ταυτόχρονα την ίδια γραμμή).

4.1.10 Μηχανισμός εξισορρόπησης φορτίου (load-balancing)

Από την ανάλυση της ενότητας 4.1.6 είναι φανερό ότι η βασική μονάδα που σχετίζεται με την κλιμακωσιμότητα και το load-balancing στην HBase είναι η HRegion. Οι HRegions μοιράζονται στους διαθέσιμους φυσικούς κόμβους, κατανέμοντας έτσι το φορτίο ερωτημάτων και μάλιστα επιτρέπουν τη βέλτιστη εξισορρόπηση του φορτίου, εφόσον μπορούν εύκολα να μετακινηθούν σε άλλους κόμβους αν το φορτίο στον κόμβο που τις εξυπηρετεί είναι πολύ μεγάλο ή σταματήσει να λειτουργεί ο κόμβος. Το splitting και η ανάθεση των HRegions σε κόμβους γίνονται αυτόματα (auto-sharding) και ολοκληρώνονται σε σύντομο χρονικό διάστημα.

Στην περίπτωση προσθήκης κόμβου στη συστοιχία, την εκκίνηση δηλαδή ενός νέου HRegionServer, γίνεται μετακίνηση κάποιων HRegions σε αυτόν έτσι ώστε όλοι οι HRegionServers να έχουν περίπου των ίδιο αριθμό HRegions να εξυπηρετήσουν. Τα δεδομένα φυσικά δεν χρειάζεται να μετακινηθούν αφού τους DataNodes δεν τους χειρίζεται η HBase αλλά το HDFS. Έτσι επιτυγχάνεται πολύ γρήγορα εξισορρόπηση του φορτίου στη συστοιχία.

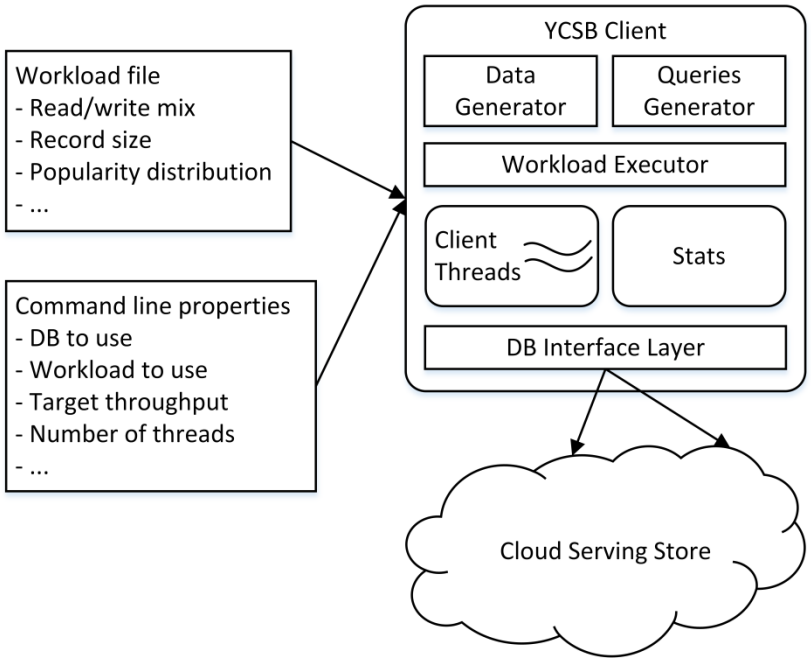
Στην περίπτωση αφαίρεσης κόμβου από τη συστοιχία ή αν ένας κόμβος σταματήσει να λειτουργεί εξαιτίας αστοχίας υλικού, επιτυγχάνεται επίσης γρήγορη ανάκαμψη του συστήματος καθώς οι HRegions που εξυπηρετούσε ο HRegionServer αυτού του κόμβου μοιράζονται στους υπόλοιπους λειτουργικούς HRegionServers. Επίσης, αν λειτουργούσε και κάποιος DataNode σε αυτόν τον κόμβο το σύστημα μπορεί να συνεχίσει να λειτουργεί καθώς αντίγραφα όλων των δεδομένων που περιείχε ο DataNode βρίσκονται σε άλλους DataNodes και το HDFS ξεκινά αυτόματα να αναπληρώνει τα χαμένα αντίγραφα σε άλλους υπάρχοντες DataNodes. Φυσικά η εγγραφή των χαμένων δεδομένων έχει κάποιο κόστος για τους κόμβους που λειτουργούν ως DataNodes και για αυτό συνήθως οι DataNodes τοποθετούνται σε κόμβους που δεν θα αφαιρεθούν από τη συστοιχία για να αποφευχθεί κατά το δυνατόν αυτή η περίπτωση ή σε κάποιους που δεν χρησιμοποιούνται ως HRegionServers. Το μοναδικό που μπορεί να επιβραδύνει την ανάκαμψη του συστήματος όταν αφαιρεθεί κόμβος είναι η ύπαρξη δεδομένων σε MemStores στη μνήμη του συγκεκριμένου κόμβου. Τα δεδομένα αυτά είναι προσωρινά αποθηκευμένα στην μνήμη με σκοπό να εγγραφούν στο HDFS και όπως είναι φυσικό θα χαθούν, σε αντίθεση με το WAL που όντας αποθηκευμένο στο HDFS έχει αντιγραφεί με ασφάλεια σε άλλους DataNodes (ενότητα 4.1.8). Για να ανακτήσει αυτά τα δεδομένα η HBase πρέπει, αφού αναθέσει τις HRegions σε άλλους κόμβους, να ξανατρέξει τις αλλαγές χρησιμοποιώντας το WAL από όποιον DataNode το έχει και η διαδικασία αυτή μπορεί να επιβραδύνει αρκετά το σύστημα μέχρι να ολοκληρωθεί.

4.2 Yahoo Cloud Serving Benchmark

Το Yahoo Cloud Serving Benchmark (YCSB) [7] είναι ένα project ανοικτού λογισμικού που κατασκευάστηκε με σκοπό να διευκολύνει την αξιολόγηση και σύγκριση της απόδοσης διαφορετικών συστημάτων αποθήκευσης και διαχείρισης δεδομένων (με έμφαση σε συστήματα NoSQL) σε υπολογιστικά νέφη. Πρόκειται για ένα πρόγραμμα σε

Java που ενσωματώνεται μαζί με ένα πακέτο πρότυπων workloads σε έναν ή περισσότερους πελάτες. Οι πελάτες αυτοί είναι υπεύθυνοι τόσο για την παραγωγή και αποθήκευση του αρχικού dataset πάνω στο οποίο θα εκτελεστούν τα πειραματικά workloads, όσο και για την εκτέλεση των workloads αυτών.

Η αρχιτεκτονική ενός YCSB πελάτη παρουσιάζεται στο επόμενο σχήμα.



Σχήμα 9: Αρχιτεκτονική YCSB πελάτη

Η βασική λειτουργία ενός YCSB πελάτη είναι η εξής: η διεργασία εκτέλεσης του φόρτου εργασίας (workload executor) χειρίζεται πολλαπλά νήματα. Κάθε νήμα εκτελεί μια ακολουθιακή σειρά λειτουργιών επικοινωνώντας με το στρώμα διεπαφής της βάσης δεδομένων, είτε για να φορτώσει δεδομένα στη βάση είτε για να εκτελέσει το φορτίο (workload). Τα νήματα οριοθετούν το ρυθμό με τον οποίο παράγονται τα ερωτήματα, έτσι ώστε να μπορούμε να ελέγχουμε το φορτίο που εφαρμόζουμε στη συστοιχία. Επίσης τα νήματα μετρούν το χρόνο αναμονής και το ρυθμό εξυπηρέτησης που επιτυγχάνεται στις λειτουργίες, και στέλνουν αυτές τις μετρήσεις στη μονάδα στατιστικών (statistics module) όπου όταν ολοκληρωθεί το φορτίο εξάγονται και στατιστικά αποτελέσματα.

Ο πελάτης δέχεται μια σειρά ιδιοτήτων που καθορίζουν τις λειτουργίες, οι οποίες χωρίζονται σε δύο κατηγορίες:

- Ιδιότητες workload: Ιδιότητες που καθορίζουν το workload, ανεξαρτήτως της βάσης που χρησιμοποιείται. Για παράδειγμα, ο συνολικός αριθμός λειτουργιών που θα εκτελεστούν, η αναλογία των λειτουργιών ανάγνωσης/εγγραφής (read/write), ο όγκος των δεδομένων στον οποίο θα εφαρμοστούν, η κατανομή των ερωτημάτων σε αυτά τα δεδομένα (zipfian, ομοιόμορφη, κα.), το μέγεθος και τα πεδία μιας εγγραφής (record), κα.
- Ιδιότητες χρόνου εκτέλεσης: Ιδιότητες που σχετίζονται με το συγκεκριμένο πείραμα. Για παράδειγμα, το στρώμα διεπαφής της βάσης δεδομένων προς χρήση (π.χ. Cassandra, HBase, κα.), ιδιότητες που χρησιμοποιούνται για την αρχικοποίηση αυτού του στρώματος (όπως το hostname του κεντρικού εξυπηρετητή της βάσης), ο αριθμός των νημάτων στον πελάτη, ο ρυθμός εξυπηρέτησης που στοχεύουμε (target throughput), κα.

Αξίζει να σημειωθεί το γεγονός ότι κάθε YCSB νήμα εκτελεί, όπως τονίσαμε, ακολουθιακά λειτουργίες, δηλαδή στέλνει το επόμενο ερώτημα μόνο αφού έχει απαντηθεί το προηγούμενο. Αυτό σημαίνει ότι ανεξαρτήτως του ρυθμού παραγωγής ερωτημάτων που στοχεύει ένα νήμα, όσο πιο ψηλός είναι ο χρόνος αναμονής για την ολοκλήρωση κάθε ερωτήματος τόσο πιο πολύ πέφτει εκ των πραγμάτων αυτός ο ρυθμός οπότε ουσιαστικά το YCSB αδυνατεί να στείλει το επιθυμητό φορτίο στη βάση. Για παράδειγμα, έστω ότι για να εφαρμόσουμε φορτίο στη συστοιχία χρησιμοποιούμε ένα νήμα με το οποίο θέλουμε να στέλνουμε ερωτήματα με ρυθμό 1 ερώτημα/δευτερόλεπτο. Αν ο χρόνος αναμονής για ένα ερώτημα είναι 2 δευτερόλεπτα τότε το νήμα αναπόφευκτα μπορεί να στείλει 1 ερώτημα ανά 2 δευτερόλεπτα δηλαδή ο ρυθμός πέφτει στα 0.5 ερωτήματα/δευτερόλεπτο. Για αυτό σε συστήματα που υπάρχει υψηλός χρόνος αναμονής χρησιμοποιείται μεγάλος αριθμός νημάτων για την εκτέλεση του φόρτου εργασίας, καθώς το καθένα μπορεί να στέλνει ερωτήματα ταυτόχρονα και ανεξάρτητα από τα υπόλοιπα χρησιμοποιώντας μικρότερο ρυθμό αποστολής ερωτημάτων ανά νήμα, και διατηρώντας τον συνολικό ρυθμό αρκετά μεγάλο. Με αυτό τον τρόπο μπορεί να επιτευχθεί τελικά ο επιθυμητός ρυθμός αποστολής ερωτημάτων.

Κεφάλαιο 5

Πειραματική αξιολόγηση

5.1 Οργάνωση πειραμάτων

Για τη διεξαγωγή των πειραμάτων χρησιμοποιήθηκε το ιδιωτικό υπολογιστικό νέφος του εργαστηρίου το οποίο χρησιμοποιεί το πρόγραμμα διαχείρισης OpenStack. Χρησιμοποιώντας το νέφος αυτό δημιουργούμε εικονικές μηχανές (VMs) η κάθε μια εκ των οποίων έχει επεξεργαστή με 2 εικονικούς πυρήνες, 4GB μνήμης RAM και 200GB αποθηκευτικού χώρου, για να τις χρησιμοποιήσουμε ως πελάτες ή εξυπηρετητές. Αρχικά δημιουργήσαμε 10 VMs - πελάτες για την παραγωγή του φορτίου, οι οποίοι εκτελούν το πρόγραμμα YCSB [7]. Οι πελάτες στέλνουν το φορτίο τους σε μια συστοιχία από 2 έως 10 VMs – εξυπηρετητές. Οι εξυπηρετητές χρησιμοποιούν την βάση NoSQL HBase [3],[8],[9] η οποία τρέχει πάνω από το κατανεμημένο αποθηκευτικό σύστημα Hadoop (HDFS).

Οι εκδόσεις των Hadoop [2], [22] και Ganglia [13] που χρησιμοποιήθηκαν είναι η 1.0.1 και 3.1.2 αντίστοιχα, με τις προεπιλεγμένες ρυθμίσεις τους. Χρησιμοποιούμε μια συστοιχία HBase (v. 0.92.0) αρχικού μεγέθους 4 VMs (μπορεί να αυξηθεί μέχρι 10 VMs και να μειωθεί σε μόνο 2 VMs – έναν HRegionServer και τον HMaster) στην οποία χρησιμοποιήθηκαν οι ρυθμίσεις που περιγράφουμε αναλυτικά στις ενότητες 5.1.1-3. Στους πελάτες ενσωματώθηκε η έκδοση 0.1.4 του YCSB.

Αρχικά κάναμε εισαγωγή 20M YCSB εγγραφών (records) στην συστοιχία χρησιμοποιώντας έναν YCSB πελάτη. Μια YCSB εγγραφή περιέχει δεδομένα μεγέθους 1KB και καταλαμβάνει συνολικά 1,5KB στο δίσκο μαζί με κάποια επιπλέον δεδομένα που αποθηκεύει η HBase (όπως το σχετικό κλειδί). Επομένως δημιουργείται ένα dataset συνολικού μεγέθους 30GB, το οποίο όταν αποθηκευτεί στο HDFS με συντελεστή αντιγραφής (replication factor) 3 φτάνει τα 90GB. Με το YCSB δημιουργήθηκε επίσης το φορτίο που εφαρμόστηκε στη συστοιχία από τους 10 πελάτες, το οποίο περιλαμβάνει απλά get (ανάγνωση) και put (ενημέρωση) ερωτήματα.

Για να παρατηρήσουμε τον τρόπο που αλλάζει η συμπεριφορά και η απόδοση της συστοιχίας καθώς αλλάζει το μέγεθός του φροντίσαμε να παρέχουμε μέσω των πελατών

έναν αρκετά υψηλό ρυθμό αποστολής αιτημάτων με σκοπό να εξαντλήσουμε τους παρεχόμενους πόρους εξυπηρέτησης. Μόνο όταν η συστοιχία έχει φτάσει στα όριά της κατά την εξυπηρέτηση ερωτημάτων μπορούμε να παρατηρήσουμε αλλαγές στην επίδοση με την προσθαφαίρεση πόρων (δηλαδή κόμβων). Στην περίπτωση μιας συστοιχίας που υπολειτουργεί, η προσθήκη επιπλέον κόμβων δεν επιφέρει αλλαγές στην επίδοση που παρατηρούμε.

Η εξάντληση των παρεχόμενων πόρων έχει σαν αποτέλεσμα το σύστημα να φτάνει στα όριά του, και μερικές φορές να παρουσιάζει μη αναμενόμενη συμπεριφορά, λόγω του υπερβολικού φόρτου εργασίας. Για την αντιμετώπιση αυτών των φαινομένων και τη μεγιστοποίηση της απόδοσης του συστήματος όταν εκτελείται μεγάλος αριθμός λειτουργιών ανάγνωσης ή ενημέρωσης τροποποιήθηκαν ορισμένες παράμετροι που ρυθμίζουν τον τρόπο λειτουργίας της HBase. Επίσης για την αποδοτικότερη λειτουργία των πελατών όταν απαιτείται υψηλός ρυθμός αποστολής αιτημάτων επιλέχθηκαν κατάλληλες τιμές για τις απαιτούμενες παραμέτρους λειτουργίας του YCSB. Ακολουθεί αναλυτική περιγραφή της διαδικασίας επιλογής των παραμέτρων με σκοπό τη μεγιστοποίηση της απόδοσης.

5.1.1 Βασικές ρυθμίσεις της HBase

Επιλέξαμε την αποθήκευση μόνο της τελευταίας έκδοσης κάθε κελιού κατά την εκτέλεση ενημερώσεων (η HBase κρατάει εξ' ορισμού τις τελευταίες τρεις εκδόσεις). Η επιλογή αυτή έγινε για να διατηρούμε σταθερό εξαρχής το μέγεθος και τον αριθμό των HFiles και τον συνολικό αριθμό των HRegions. Με αυτό τον τρόπο, όταν εφαρμόζουμε ενημερώσεις εγγραφών στην HBase το dataset παραμένει το ίδιο και έχει τις ίδιες απαιτήσεις όσον αφορά τόσο τον αποθηκευτικό χώρο που χρειάζεται στον δίσκο, όσο και τον χώρο που δεσμεύει στην μνήμη RAM για caching. Αυτό επιτρέπει να εξάγουμε πιο ξεκάθαρα συμπεράσματα κατά την προσθαφαίρεση κόμβων. Επίσης, αντί να καταναλωθεί χώρος της BlockCache για την αποθήκευση των επιπλέον εκδόσεων που μεταφέρει κάθε block, μπορεί να χρησιμοποιηθεί αυτός ο χώρος για αποθήκευση μεγαλύτερου αριθμού εγγραφών.

Έτσι επιτυγχάνεται η επιθυμητή κλιμάκωση κατά την πρόσθεση επιπλέον κόμβων με την κατανομή του αρχικού αριθμού των HRegions στους νέους εξυπηρετητές που προσθέτονται, χωρίς να αλλάζει ο αρχικός αριθμός HRegions κατά τις ενημερώσεις αντικειμένων.

Ο πίνακας που περιέχει τα δεδομένα αποτελείται από μια οικογένεια στηλών, επομένως δημιουργείται 1 Store σε κάθε HRegion. Το μέγιστο μέγεθος ενός HFile, που καθορίζει πότε θα γίνει splitting μια HRegion σε 2 μικρότερες, ορίστηκε στα 32MB. Επιλέξαμε αυτό το μέγεθος για να δημιουργηθεί εξαρχής μεγάλος αριθμός HRegions, που στη συνέχεια θα κατανεμηθούν ομοιόμορφα σε έναν αυξανόμενο αριθμό HRegionServers με σκοπό την καλύτερη εξισορρόπηση φόρτου εργασίας. Επίσης το μέγεθος κάθε HDFS block ορίστηκε να είναι 256MB. Δεδομένου ότι η συνολική μνήμη σε κάθε εξυπηρετητή είναι 4GB, επιλέχθηκαν 3072MB να είναι διαθέσιμα για χρήση από τον αντίστοιχο HRegionServer (για λειτουργίες caching, κλπ) αφήνοντας 1 GB για τις ανάγκες του λειτουργικού συστήματος και των άλλων εφαρμογών που τρέχουν παράλληλα.

5.1.2 Ρυθμίσεις για βελτιστοποίηση λειτουργιών ανάγνωσης

Με βάση το διαθέσιμο χώρο μνήμης για κάθε HRegionServer και το ποσοστό που αφιερώνει η HBase για την BlockCache, το μέγεθος της BlockCache σε κάθε κόμβο είναι 25%·3072MB ≈ 653MB. Όταν η BlockCache γεμίσει περισσότερο από 85%, δηλαδή γεμίσουν περισσότερα από 609MB, αρχίζει να αφαιρεί blocks (eviction) με χρήση της πολιτικής LRU μέχρι να μειωθεί ο χρησιμοποιούμενος χώρος κάτω από τα 609MB. Στη BlockCache αποθηκεύονται εκτός από τα blocks των HFiles, οι πίνακες καταλόγου και τα ευρετήρια των HFiles οπότε λαμβάνουμε υπόψη ότι θα χρησιμοποιηθεί ένα μέρος της και για αυτά (λιγότερο από 50MB).

Συνέπεια της διαδικασίας του eviction είναι η αύξηση του χρόνου αναμονής στην ανάγνωση καθώς αν χρειαστεί να γίνει ανάγνωση κάποιου block που δεν υπάρχει πλέον στην BlockCache, θα πρέπει αυτό να μεταφερθεί πάλι από το δίσκο. Για να μην αφαιρούνται δεδομένα από τη BlockCache και μειωθεί η απόδοση λόγω του αυξημένου χρόνου αναμονής της ανάγνωσης, είναι επιθυμητό να χρησιμοποιούνται λιγότερα από 609MB στη BlockCache κάθε κόμβου, δηλαδή λιγότερα από 559MB καθαρά για τα δεδομένα που ζητάμε. Για το σκοπό αυτό χρειάστηκε να τροποποιηθούν κατάλληλα οι παράμετροι του YCSB που ρυθμίζουν το φορτίο, έτσι ώστε οι λειτουργίες ανάγνωσης να γίνονται στοχευμένα σε περιορισμένο όγκο δεδομένων που να μην προκαλούν υπέρβαση των επιθυμητών ορίων στη BlockCache η οποία με την σειρά της μειώνει το ρυθμό εξυπηρέτησης και αυξάνει το μέσο χρόνο εξυπηρέτησης των ερωτημάτων. Η ρύθμιση του YCSB περιγράφεται αναλυτικά στην ενότητα 5.1.4.

Ο λόγος για τον οποίο κάναμε τους παραπάνω υπολογισμούς, είναι για να έχουμε ένα φορτίο που δεν παρουσιάζει "ανεξήγητες" διακυμάνσεις κατά την προσθαφαίρεση κόμβων οι οποίες οφείλονται σε λειτουργίες caching. Με την επιλογή του συγκεκριμένου workload και μεγέθους BlockCache φροντίζουμε οι αιτήσεις για ανάγνωση να εξυπηρετούνται κατά κύριο λόγο από την BlockCache και όχι από τον δίσκο για όλα τα πιθανά μεγέθη της συστοιχίας. Σε διαφορετική περίπτωση, όπου π.χ. μια μικρή συστοιχία δεν έχει αρκετή BlockCache να αποθηκεύσει όλα τα ζητούμενα αντικείμενα, η προσθήκη επιπλέον κόμβων θα αύξανε δυσανάλογα την επίδοση του συστήματος. Η μεγάλη αύξηση του ρυθμού εξυπηρέτησης θα οφειλόταν περισσότερο στην αύξηση της συνολικής BlockCache (επομένως όλα τα ζητούμενα αντικείμενα θα εξυπηρετούνταν κατευθείαν από την κύρια μνήμη), και λιγότερο στην υπολογιστική ισχύ που θα έδιναν οι παραπάνω κόμβοι. Αυτή η συμπεριφορά, την οποία παρατηρήσαμε σε περιπτώσεις που δεν λαμβάναμε υπόψη την BlockCache, περιπλέκει την ανάλυση που θέλουμε να κάνουμε. Μάλιστα σε όλα τα πειράματα που εκτελέσαμε φροντίσαμε να είναι πάντα προ-φορτωμένη η BlockCache με τα δεδομένα που διαβάζουμε για να έχουμε την μέγιστη απόδοση εξαρχής, χωρίς να περιμένουμε να γίνει πρώτα η μεταφορά όλων των δεδομένων στη μνήμη. Μελλοντικά έχουμε σκοπό να μελετήσουμε την επίδραση τη BlockCache κατά την προσθαφαίρεση κόμβων σε διαφορετικού είδους ερωτήματα.

5.1.3 Ρυθμίσεις για βελτιστοποίηση λειτουργιών ενημέρωσης

Ο συνολικός διαθέσιμος χώρος σε ένα κόμβο για τα MemStores των HRegions που εξυπηρετεί ο συγκεκριμένος HRegionServer είναι 40%·3072MB ≈ 1228MB. Το αποδεκτό όριο χρήσης αυτού του χώρου είναι 35%·3072MB ≈ 1075MB. Στο εξής θα αναφερόμαστε στην τιμή των 1228MB ως το ανώτερο όριο χρήσης του χώρου και στα 1075MB ως το κατώτερο όριο. Υπάρχει ένα νήμα που είναι υπεύθυνο για το flush των MemStores (την αποθήκευση δηλαδή των MemStores στο HDFS και ελευθέρωση του χώρου που καταναλώνουν στην κύρια μνήμη) που ελέγχει τακτικά αυτά τα δύο όρια. Αν εντοπίσει ότι το συνολικό μέγεθος των MemStores έχει υπερβεί το κατώτερο όριο εκτελεί flush μέχρι να μειωθεί ο χώρος που χρησιμοποιείται κάτω από τα 1075MB, ενώ αν έχει ξεπεράσει και το ανώτερο όριο αναστέλλει τις εντολές ενημερώσεων και τις ξεμπλοκάρει όταν ολοκληρωθεί το flush. Επομένως, στο πρώτο όριο κάνει flush και δέχεται και εντολές ενημερώσεων, ενώ στο δεύτερο όριο (που είναι πιο κρίσιμο) κάνει flush και μπλοκάρει τις εντολές. Στα πειράματα που εκτελέσαμε δεν παρατηρήθηκε μπλοκάρισμα

των updates εξαιτίας υπέρβασης του ανώτερου ορίου, αλλά υπήρξε μπλοκάρισμα εξαιτίας συγκεκριμένων ρυθμίσεων στην HBase, τις οποίες αναλύουμε στις επόμενες δύο παραγράφους.

Σε περιβάλλον με υψηλό ρυθμό εκτέλεσης λειτουργιών ενημέρωσης είναι αναμενόμενο ότι τα MemStores θα γεμίζουν σχετικά γρήγορα και θα δημιουργούνται αρκετά HFiles μετά από επαναλαμβανόμενο flushing. Σε αυτή την περίπτωση αν τα HFiles (ή StoreFiles) που υπάρχουν σε ένα Store ξεπεράσουν τον αριθμό της παραμέτρου `hbase.hstore.blockingStoreFiles` (7 από προεπιλογή) τα updates θα μπλοκαριστούν για τη συγκεκριμένη HRegion για να εκτελεστεί compaction και θα ξεμπλοκαριστούν το πολύ μετά από 90 δευτερόλεπτα (προεπιλεγμένη τιμή της παραμέτρου `hbase.hstore.blockingWaitTime`) ακόμα και αν δεν έχει ολοκληρωθεί το compaction. Για να αποκλειστεί αυτή τη περίπτωση μπλοκαρίσματος του update για να εκτελεστεί compaction, θέσαμε στην τελευταία παράμετρο στα 0 δευτερόλεπτα.

Επίσης υπάρχει η περίπτωση να μπλοκαριστούν τα updates για μια HRegion εάν το MemStore της φτάσει το μέγεθος των 2·128MB = 256 MB (`hbase.hregion.block.multiplier` · `hbase.hregion.flush.size`). Γενικά, όπως αναφέραμε προηγουμένως, όταν ένα MemStore ξεπεράσει σε μέγεθος τα 128MB γίνεται flush στο δίσκο. Υπό ορισμένες προϋποθέσεις όμως υπάρχει περίπτωση να μην είναι δυνατή η εκτέλεση του flush και να συνεχίσει να αυξάνεται το μέγεθος του MemStore. Μια τέτοια περίπτωση είναι η εξής: έστω ότι μετά το flush ενός MemStore ο αριθμός των StoreFiles στο Store φτάσει στο όριο (7 StoreFiles) οπότε και θα ξεκινήσει η διαδικασία του compaction με βάση όσα περιγράψαμε στην προηγούμενη παράγραφο. Παράλληλα συνεχίζεται η εξυπηρέτηση αιτημάτων καθώς πριν αρχίσει το flush δεσμεύτηκε άδειος χώρος στη μνήμη για να χρησιμοποιηθεί ως νέο MemStore. Εάν πριν προλάβει να ολοκληρωθεί το compaction, το μέγεθος του νέου MemStore αυξηθεί και ξεπεράσει τα 128MB επειδή δέχθηκε πολλά αιτήματα ενημέρωσης, δε θα μπορεί να γίνει flush επειδή ο αριθμός των StoreFiles είναι ήδη στο όριο. Σε περιπτώσεις όπως αυτή, για να μην συνεχίσει να γεμίζει απεριόριστα ένα MemStore εις βάρος του διαθέσιμου χώρου μνήμης για τα υπόλοιπα MemStore, είναι απαραίτητο να τεθεί το όριο των 256MB στο οποίο βέβαια μπλοκάρονται τα αιτήματα ενημέρωσης. Για να αποφευχθεί το μπλοκάρισμα στη συγκεκριμένη περίπτωση μια λύση είναι η αύξηση της παραμέτρου `hbase.hregion.block.multiplier`, αλλά για λόγους περιορισμένης μνήμης το αποφύγαμε και αυξήσαμε εναλλακτικά την παράμετρο

`hbase.hstore.blockingStoreFiles` στην τιμή 100. Με αυτό τον τρόπο υπάρχει περιθώριο για να αποθηκεύονται περισσότερα νέα HFiles χωρίς να φτάνει τόσο συχνά στο όριο `blockingStoreFiles` ο αριθμός τους. Έτσι το flush του MemStore μπορεί να εκτελείται κανονικά τις περισσότερες φορές (αφού ο αριθμός των HFiles δεν έχει φτάσει στο όριο) και το μέγεθός του δε θα συνεχίζει να αυξάνεται με κίνδυνο να φτάσει το όριο των 256MB όπου μπλοκάρονται τα updates. Επομένως, αφού το μέγεθος του MemStore φτάνει δύσκολα τα 256MB, μπλοκάρονται λιγότερο συχνά τα updates αλλά θα γίνεται ένα πιο χρονοβόρο compaction όταν έχουν δημιουργηθεί 100 HFiles.

Με τις παραπάνω ρυθμίσεις επιτεύχθηκε βελτιστοποίηση της απόδοσης κυρίως σε περιβάλλον με υψηλό ρυθμό αποστολής αιτημάτων ενημέρωσης, μειώνοντας τις πιθανότητες μπλοκαρίσματος λόγω προκαθορισμένων ορίων στην υλοποίηση της HBase.

5.1.4 Επιλογή βέλτιστων παραμέτρων για το YCSB

Για την διαμόρφωση του φορτίου λήφθηκε υπόψη το εποχικό φαινόμενο που παρουσιάζεται σε εφαρμογές παγκόσμιου ιστού, στις οποίες το φορτίο αυξάνεται κατά τις ώρες αιχμής και πέφτει κατά τις ώρες μη αιχμής. Αυτό το φαινόμενο φαίνεται παραδείγματος χάριν στο *Akamai's 24-day load* (Σχ. 14 στο [18]) και το *Microsoft Messenger's weekly load* (Σχ. 5 στο [19]) που ακολουθούν ένα παρόμοιο μοτίβο, όπου το φορτίο συνεχώς αυξάνεται και μειώνεται. Στη συστοιχία εφαρμόστηκε κατά παρόμοιο τρόπο φορτίο ημιτονοειδούς μορφής, το οποίο υλοποιήθηκε ελέγχοντας περιοδικά το ξεκίνημα και σταμάτημα του φορτίου σε έναν αριθμό πελατών.

Για τη δημιουργία του φορτίου σε κάθε πελάτη με χρήση του YCSB απαιτείται ο προσδιορισμός των εξής βασικών παραμέτρων:

- Η αναλογία των αιτημάτων ανάγνωσης/εγγραφής.
- Ο όγκος των δεδομένων στον οποίο θα εφαρμοστούν.
- Η κατανομή των ερωτημάτων σε αυτά τα δεδομένα.
- Ο αριθμός των νημάτων στον πελάτη.
- Η απόδοση που στοχεύουμε (δηλαδή ο ρυθμός εξυπηρέτησης των αιτημάτων).

Αρχικά, τα ερωτήματα που χρησιμοποιήθηκαν, όπως αναφέρθηκε και προηγουμένως, είναι δύο τύπων: ανάγνωσης (read) και ενημέρωσης (update). Επιλέξαμε τη χρήση ερωτημάτων ενημέρωσης αντί εισαγωγής (insert) για να αποφύγουμε την αύξηση του μεγέθους των αποθηκευμένων δεδομένων στη βάση και να μελετήσουμε καθαρά τη

λειτουργία του συστήματος της HBase χωρίς την εμπλοκή του HDFS. Η κατανομή των ερωτημάτων στα δεδομένα ορίστηκε να είναι ομοιόμορφη ενώ πειραματιστήκαμε με διάφορες αναλογίες ερωτημάτων ανάγνωσης/ενημέρωσης.

Ο όγκος των δεδομένων στον οποίο θα εφαρμόζεται το φορτίο επιλέχθηκε βάσει της ανάλυσης που έγινε στην παράγραφο 5.1.2. Το μέγεθος της BlockCache προς χρήση σε κάθε κόμβο υπολογίστηκε προηγουμένως ότι είναι 559MB. Δεδομένου του αρχικού μεγέθους της συστοιχίας (3 εξυπηρετητές χωρίς τον HMaster) έχουμε συνολικά 1.677MB διαθέσιμης BlockCache, που χωράει συνολικά 1.677MB / 64KB/block ≈ 26.200 blocks μεγέθους 64KB. Για να εξαλειφθεί η πιθανότητα να γεμίσει η BlockCache κυρίως όταν το φορτίο έχει αποκλειστικά αιτήματα ανάγνωσης (οπότε κατά συνέπεια θα αυξηθεί ο χρόνος αναμονής στην ανάγνωση όπως περιγράψαμε στην παράγραφο 5.1.2), το φορτίο επιλέχθηκε να εφαρμόζεται σε ένα σύνολο 20.000 εγγραφών στη βάση (σε ποσοστό δηλαδή 1‰). Με αυτό τον τρόπο, ακόμα και αν κάθε εγγραφή που ζητήσουμε για ανάγνωση ανήκει σε διαφορετικό block, δεν πρόκειται να μεταφερθούν πάνω από 20.000 blocks στην BlockCache. Το YCSB επιλέγει τα κλειδιά αυτών των εγγραφών από όλο το σύνολο των δεδομένων με βάση μια συνάρτηση κατακερματισμού των τιμών 1-20.000. Επειδή η συνάρτηση κατακερματισμού είναι τυχαία, η πιθανότητα το κάθε κλειδί να βρίσκεται σε διαφορετικό block είναι μεγάλη, και κάθε αίτημα ανάγνωσης ενός κλειδιού φέρνει ένα ολόκληρο block στην μνήμη. *Οι πελάτες εφαρμόζουν επομένως το φορτίο τους, με ομοιόμορφη κατανομή των ερωτημάτων, στις ίδιες 20.000 εγγραφές.*

Για την επιλογή του αριθμού των νημάτων και του ρυθμού αποστολής αιτημάτων από κάθε πελάτη χρειάστηκε να μελετήσουμε όλες τις περιπτώσεις φορτίων που ορίζονται από τις διαφορετικές αναλογίες ερωτημάτων ανάγνωσης/ενημέρωσης.

Θεωρήσαμε αρχικά σε κάθε πελάτη ένα νήμα και στοχεύσαμε σε *ρυθμό αποστολής 10.000αιτήματα/δευτ.* ανά πελάτη, δηλαδή το συνολικό φορτίο από όλους τους πελάτες παρείχε 100.000αιτήματα/δευτ. στο αρχικό μέγεθος της συστοιχίας. Διαπιστώσαμε ότι οι πελάτες δεν έστελναν ερωτήματα με τον επιθυμητό ρυθμό ούτε σε φορτίο με αιτήματα ανάγνωσης ούτε με αιτήματα ενημέρωσης ενώ η συστοιχία είχε χαμηλή χρήση CPU. Το τελευταίο υποδείκνυε αδυναμία των πελατών να στείλουν παραπάνω ερωτήματα και όχι της συστοιχίας να απαντήσει σε αυτά. Η αδυναμία αυτή σχετίζεται με την ακολουθιακή αποστολή αιτημάτων από ένα νήμα, που μπορεί να καθυστερηθεί από την ύπαρξη υψηλού χρόνου αναμονής όπως εξηγήσαμε στο τέλος της παραγράφου 4.2. Για αυτό αυξήσαμε τον αριθμό των νημάτων στα 100, όπου διαπιστώσαμε ότι οι αυξήθηκε

σημαντικά ο ρυθμός αποστολής αιτημάτων ανάγνωσης αλλά όχι των αιτημάτων ενημέρωσης.

Πιο αναλυτικά, ενώ σε φορτίο με λειτουργίες ανάγνωσης μόνο (read-only workload) η συστοιχία είχε υψηλή χρήση CPU και ανταποκρινόταν σταθερά στον μέγιστο αριθμό αιτημάτων που μπορούσε, σε φορτίο με λειτουργίες ενημέρωσης (update-only workload) δεχόταν αρχικά για ένα μικρό χρονικό διάστημα τον μέγιστο αριθμό αιτημάτων που μπορούσαν να στείλουν οι πελάτες και στη συνέχεια μπλόκαρε η αποστολή αιτημάτων ενημέρωσης για ένα μεγάλο χρονικό διάστημα. Δεδομένου ότι το μπλοκάρισμα συνέβαινε κατά την επεξεργασία των αρχικών ακόμα αιτημάτων και πριν καν προλάβουν να γραφτούν αρκετά δεδομένα στο MemStore διαπιστώσαμε ότι δεν πρόκειται για μπλοκάρισμα εξαιτίας κάποιου ορίου της HBase, όπως ο αριθμός των StoreFiles που εξηγήσαμε προηγουμένως, αλλά για αδυναμία των πελατών να αποκτήσουν νέα row locks καθώς η HBase κλειδώνει την γραμμή κατά τη διαδικασία ενημέρωσης, υποστηρίζοντας exclusive write access. Όταν αποσταλούν δηλαδή αρκετά αιτήματα θα κλειδωθούν όλες οι γραμμές που περιέχουν τις συγκεκριμένες 20.000 εγγραφές που επεξεργαζόμαστε, και θα έχουμε αναμονή από τη μεριά των πελατών να απελευθερωθούν για να μπορέσουν να στείλουν τα επόμενα αιτήματα.

Επομένως σε φορτίο με αιτήματα ενημέρωσης τα παραπάνω νήματα δεν μπορούσαν να βοηθήσουν ουσιαστικά στο να επιτευχθεί ο επιθυμητός ρυθμός αποστολής, σε αντίθεση με την ανάγνωση. Σε ένα βαθμό αυτό ήταν αναμενόμενο, αφού όπως έχουμε προαναφέρει ο χρόνος αναμονής στην ανάγνωση είναι αρκετά μεγαλύτερος σε σχέση με την ενημέρωση και όταν έχουμε μεγάλο χρόνο αναμονής χρειάζονται περισσότερα νήματα όπως αναλύσαμε στο τέλος της ενότητας 4.2. Μάλιστα ο μεγάλος αριθμός νημάτων απλά έδινε τη δυνατότητα να σταλούν αρχικά ακόμα περισσότερα αιτήματα ενημέρωσης ταυτόχρονα και έτσι τα rows κλείδωναν πιο γρήγορα και μαζικά. Για αυτό το λόγο επιχειρήσαμε να μειώσουμε τον αριθμό των νημάτων στο φορτίο με αιτήματα ενημέρωσης. Αφού πειραματιστήκαμε με διάφορες τιμές διαπιστώσαμε ότι με 4 μόλις νήματα επιτυγχάναμε τη μέγιστη δυνατή απόδοση. *Σε αυτό το σημείο έγινε απαραίτητο να δημιουργήσουμε δύο διαφορετικά φορτία σε κάθε πελάτη, ένα με λειτουργίες ανάγνωσης με 100 νήματα και ένα με λειτουργίες ενημέρωσης με 4 νήματα, τα οποία μπορούν να εκτελούνται ταυτόχρονα επάνω στις ίδιες 20.000 εγγραφές. Φυσικά η συνολική επιθυμητή αναλογία σε λειτουργίες ανάγνωσης/ενημέρωσης ρυθμίστηκε ορίζοντας κατάλληλα το ρυθμό αποστολής ερωτημάτων από κάθε φορτίο ξεχωριστά.*

Για να αντιμετωπιστεί το κλείδωμα των γραμμών στο update και να επιτευχθεί ο επιθυμητός ρυθμός αποστολής αιτημάτων ενημέρωσης υπάρχουν δύο επιλογές: είτε να αυξήσουμε τον αριθμό των εγγραφών που επεξεργαζόμαστε είτε αναγκαστικά να μειώσουμε τον ρυθμό αποστολής των αιτημάτων μέχρι να δούμε ότι αυτά προλαβαίνουν να ικανοποιηθούν χωρίς να μπλοκάρονται οι πελάτες. Αρχικά δοκιμάσαμε το δεύτερο στοχεύοντας σε όλο και μικρότερο ρυθμό μέχρι να δούμε ότι δεν υπήρχε πρόβλημα με κλείδωμα rows. Το πρόβλημα εξαλείφθηκε μόνο με πολύ μικρό ρυθμό αποστολής ερωτημάτων και η λύση αυτή απορρίφθηκε καθώς ήταν πολύ μικρό το φορτίο που εφαρμοζόταν στη βάση. Έτσι εφόσον το σύνολο δεδομένων στο οποίο εφαρμόζεται το update φορτίο ήταν προφανώς πολύ μικρό, χρειάστηκε να αυξήσουμε τον αριθμό των εγγραφών, αλλά μόνο στο φορτίο του update καθώς αν αυξήσουμε τα δεδομένα στα οποία γίνεται ανάγνωση θα έχουμε υπερχείλιση της BlockCache. *Τροποποιήσαμε λοιπόν το φορτίο με λειτουργίες ενημέρωσης ώστε να εφαρμόζεται σε όλα τα δεδομένα της βάσης, δηλαδή σε 20.000.000 εγγραφές*, έτσι ώστε να εξαλείψουμε την πιθανότητα να μπλοκαριστεί το update λόγω κλειδωμένων row keys ακόμα και σε περιβάλλον με πολύ υψηλό ρυθμό αποστολής αιτημάτων ενημέρωσης. Με την τελευταία ρύθμιση επιτεύχθηκε η επιθυμητή αύξηση του ρυθμού αποστολής αιτημάτων ενημέρωσης και η μεγιστοποίηση της χρησιμοποιούμενης CPU στη συστοιχία.

Μετά από την επιλογή αυτών των βέλτιστων παραμέτρων, κάθε πελάτης μπορεί πλέον να στέλνει αιτήματα με πολύ μεγάλο ρυθμό στη συστοιχία, το οποίο φυσικά στο αρχικό του μέγεθος είχε τη μέγιστη χρήση CPU και δεν είχε τη δυνατότητα να απαντήσει σε όλα αυτά τα ερωτήματα. Παρόλο που η συστοιχία απαντούσε μόνο σε ένα μέρος των ερωτημάτων, οπότε δεν είχαμε την επιθυμητή απόδοση, οι πελάτες είχαν πολύ αυξημένη χρήση CPU (80%). Για να έχουν τη δυνατότητα οι πελάτες να διαχειριστούν τις όλο και περισσότερες απαντήσεις που θα λαμβάνουν όσο αυξάνεται το μέγεθος της συστοιχίας και η απόδοσή της, επιλέξαμε να μειώσουμε το ρυθμό αποστολής σε κάθε πελάτη στα *6.000 αιτήματα/δευτ*.

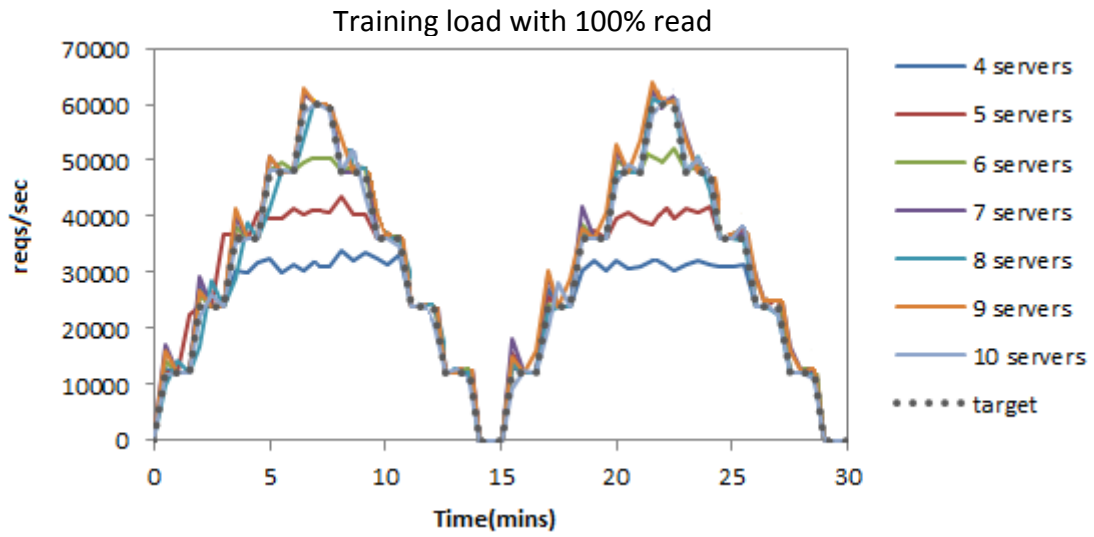
Τέλος, για την τροφοδότηση του TIRAMOLA με τα απαραίτητα μετρικά του χρήστη χρειάστηκε να τροποποιήσουμε το YCSB στους χρήστες ώστε ο καθένας να αναφέρει τις μετρήσεις του χρησιμοποιώντας μια γνωστή λειτουργία του Ganglia που ονομάζεται `gmetric spoofing`. Μάλιστα φροντίσαμε να γίνεται ξεχωριστά η αναφορά του ρυθμού εξυπηρέτησης και του χρόνου αναμονής για κάθε είδος λειτουργίας ώστε να είναι δυνατός ο προσδιορισμός του τύπου του φορτίου.

Σε όλα τα πειράματα διατηρήσαμε σταθερό το συνολικό ρυθμό αποστολής αιτημάτων από κάθε πελάτη στα 6.000αιτήματα/δευτ. και μοιράσαμε με διάφορες αναλογίες αυτό το ρυθμό στα ξεχωριστά φορτία για ερωτήματα ανάγνωσης και ενημέρωσης. Το φορτίο για ανάγνωση εφαρμόστηκε ομοιόμορφα σε 20.000 εγγραφές, με χρήση 100 νημάτων στον πελάτη ενώ το φορτίο για ενημέρωση εφαρμόστηκε ομοιόμορφα σε 20.000.000 εγγραφές, με χρήση 4 νημάτων.

5.2 Πειραματικά αποτελέσματα

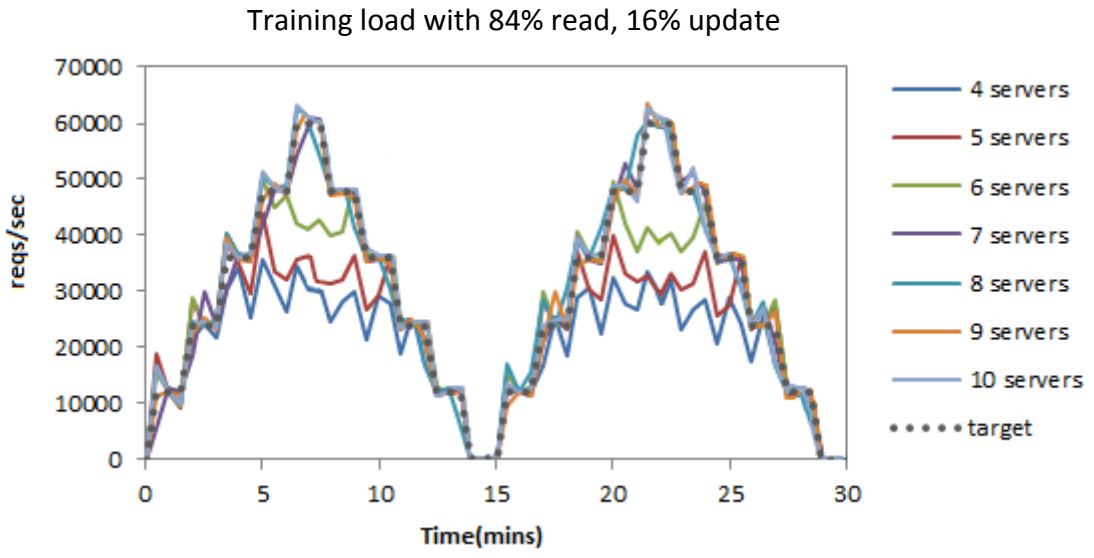
5.2.1 Προετοιμασία του TIRAMOLA μέσω εκπαίδευσης

Αρχικά, απαιτείται η συλλογή ενός συνόλου μετρικών (training set) για να χρησιμοποιηθεί κατά την πρώτη φάση λειτουργίας του TIRAMOLA, ώστε να μπορέσει η μονάδα Λήψης Αποφάσεων να υπολογίσει τις τιμές για την συνάρτηση ανταμοιβής $r(s)$. Για αυτό το σκοπό δημιουργήσαμε ένα φορτίο ημιτονοειδούς μορφής το οποίο υλοποιήθηκε προσθέτοντας/ αφαιρώντας σταδιακά φορτίο από άλλους πελάτες. Συνολικά έχουμε στη διάθεσή μας 10 VMs-πελάτες στους οποίους χρησιμοποιήθηκαν οι ρυθμίσεις που επιλέχθηκαν στην προηγούμενη ενότητα, επομένως η μέγιστη τιμή του ημιτόνου ήταν τα 60K αιτήματα/δευτ. ενώ επιλέχθηκε περίοδος 15 λεπτών. Δημιουργήσαμε 7 τέτοια φορτία, για διαφορετικές αναλογίες ερωτημάτων ανάγνωσης/ενημέρωσης (από 100% ανάγνωση μέχρι 100% ενημέρωση). Τα επιθυμητά ποσοστά επιτεύχθηκαν ρυθμίζοντας κατάλληλα σε κάθε πελάτη τον λόγο του ρυθμού αποστολής αιτημάτων από τα δύο ειδών φορτία (ανάγνωσης και ενημέρωσης) που ενσωματώνει. Για τη δημιουργία του training set εκτελέσαμε όλα τα είδη φορτίων για μια ώρα το καθένα (4 περιόδους) για κάθε μέγεθος της συστοιχίας που μελετάμε (4-10 κόμβοι) και καταγράψαμε τα επιθυμητά μετρικά. Με αυτό τον τρόπο το training set περιλαμβάνει πληροφορίες για διαφορετικά είδη και μεγέθη φορτίων που μπορούν να εφαρμοστούν σε οποιοδήποτε μέγεθος της συστοιχίας, δηλαδή για πολλές καταστάσεις λειτουργίας του συστήματος. Να υπενθυμίσουμε ότι πριν την εκτέλεση κάθε φορτίου έχουμε προ-φορτώσει την BlockCache με τα δεδομένα ανάγνωσης για να έχουμε εξαρχής την μέγιστη απόδοση στην ανάγνωση όπως έχουμε ούτως ή άλλως στην ενημέρωση.

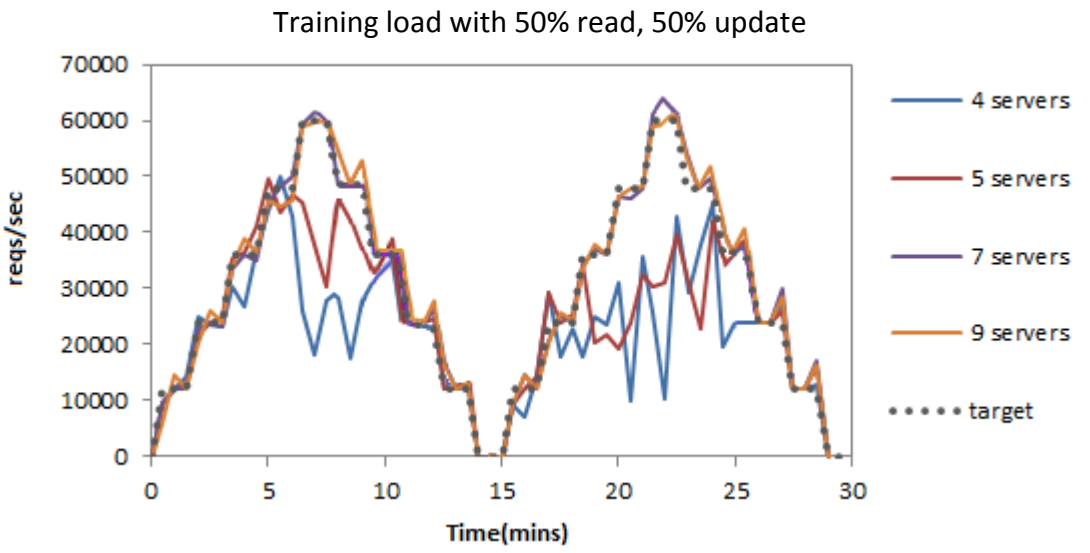


Σχήμα 10: Ρυθμός εξυπηρέτησης σε φορτίο εκπαίδευσης με 100% ερωτήματα ανάγνωσης

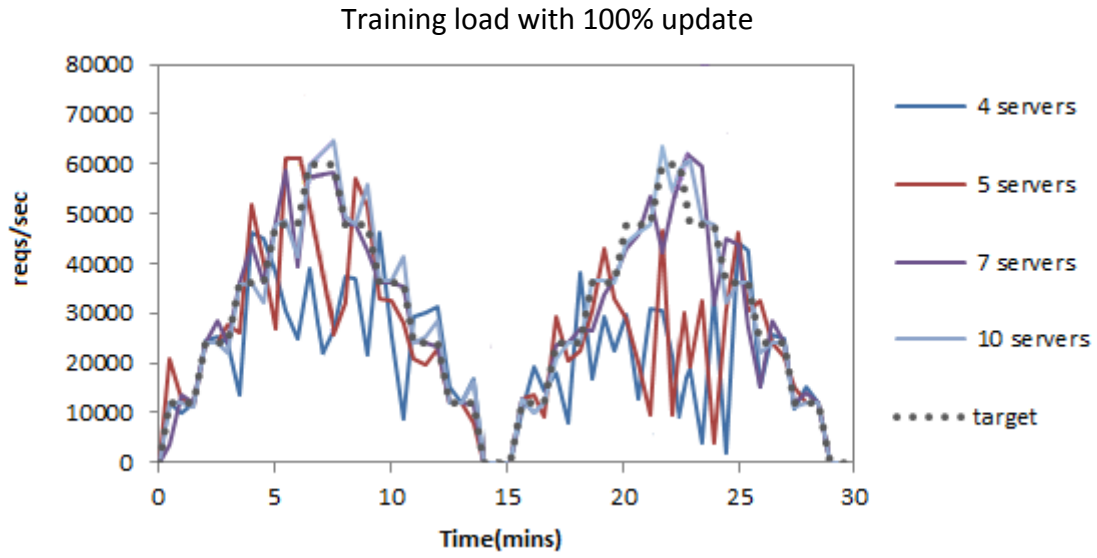
Στο Σχήμα 10 φαίνεται ενδεικτικά ο ρυθμός εξυπηρέτησης ερωτημάτων που καταγράφηκε κατά την πρώτη μισή ώρα εκτέλεσης του ημιτονοειδούς φορτίου με ερωτήματα ανάγνωσης μόνο, για όλα τα μεγέθη της συστοιχίας. Παρατηρώντας το διάγραμμα γίνεται φανερή η σχέση που υπάρχει μεταξύ του αριθμού των κόμβων της συστοιχίας και του ρυθμού εξυπηρέτησης ερωτημάτων. Συγκεκριμένα βλέπουμε ότι με 4 κόμβους (δηλαδή 3 HRegionServers και τον HMaster) η συστοιχία δεν μπορεί να εξυπηρετήσει περισσότερα από 30K αιτήματα/δευτ. και αντίστοιχα με 5 κόμβους εξυπηρετεί μέχρι 40K και με 6 κόμβους μέχρι 50K. Δηλαδή κάθε εξυπηρετητής μπορεί να απαντήσει το πολύ σε 10K αιτήματα ανάγνωσης/δευτερόλεπτο με την υπολογιστική ισχύ που διαθέτει.



Σχήμα 11: Ρυθμός εξυπηρέτησης σε φορτίο εκπαίδευσης με 84% ερωτήματα ανάγνωσης



Σχήμα 12: Ρυθμός εξυπηρέτησης σε φορτίο εκπαίδευσης με 50% ερωτήματα ανάγνωσης



Σχήμα 13: Ρυθμός εξυπηρέτησης σε φορτίο εκπαίδευσης με 100% ερωτήματα ενημέρωσης

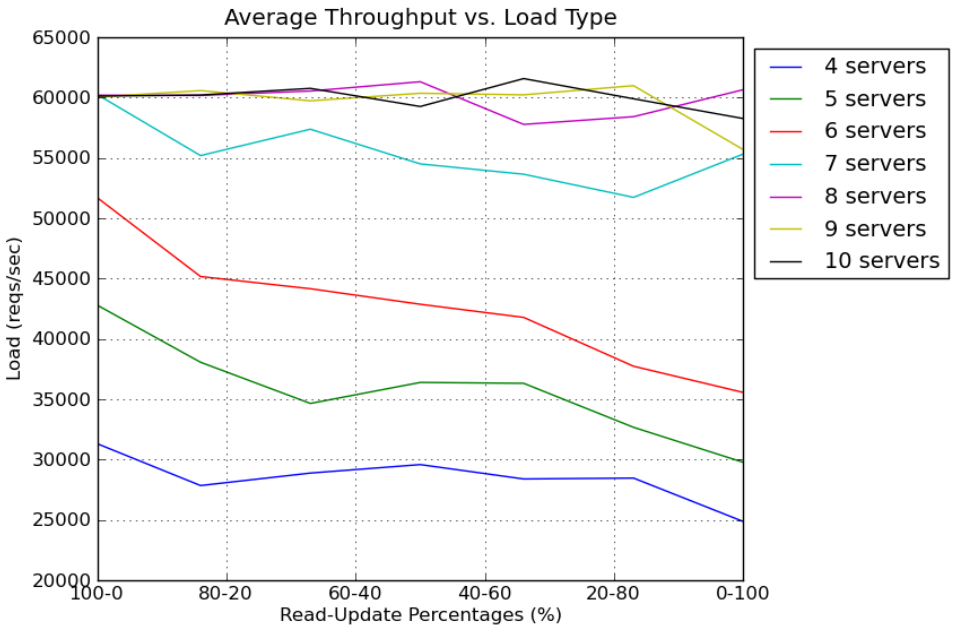
Παρόμοια ήταν τα αποτελέσματα και για όλα τα υπόλοιπα είδη φορτίων, από τα οποία δείχνουμε κάποια ενδεικτικά ποσοστά στα σχήματα 11-13. Η μοναδική διαφορά που παρατηρούμε σε αυτά τα σχήματα είναι ότι όσο αυξάνεται το ποσοστό αιτημάτων ενημέρωσης τόσο πιο "ασταθής" είναι η απόδοση κυρίως στα μικρά μεγέθη της συστοιχίας, χωρίς ωστόσο να φαίνεται κάποια σημαντική μείωση σε αυτή. Αυτή η συμπεριφορά σχετίζεται με την περιορισμένη διαθέσιμη μνήμη και τον υψηλό ρυθμό εγγραφής στα MemStores (στη μνήμη). Επειδή δηλαδή τα MemStores γεμίζουν γρήγορα

με τα δεδομένα των αιτημάτων ενημέρωσης, γίνονται πιο συχνά flushes τον δεδομένων στο δίσκο και αυτό επηρεάζει, έστω και στιγμιαία, το σύστημα.

Για να γίνει πιο κατανοητή η αδυναμία της HBase να απαντήσει σε περισσότερα ερωτήματα όταν αποτελείται από μικρό αριθμό κόμβων, παραθέτουμε στη επόμενη ενότητα ενδεικτικά ορισμένα διαγράμματα με μετρικά που καταγράφηκαν όταν η συστοιχία δέχεται το μέγιστο φορτίο (60K αιτήματα/δευτ.). Για κάθε είδος φορτίου υπολογίσαμε το μέσο όρο των καταγεγραμμένων μετρικών ώστε να μπορέσουμε να παρατηρήσουμε καλύτερα πως αλλάζει η συμπεριφορά της HBase όταν αλλάζει το είδος του φορτίου, χωρίς να επηρεαζόμαστε από την παραπάνω "αστάθεια" των μετρήσεων.

5.2.2 Σύγκριση της επίδοσης της HBase με διαφορετικά είδη φορτίου

Στα διαγράμματα που ακολουθούν παρουσιάζονται ο μέσος χρόνος αναμονής και ο μέσος ρυθμός εξυπηρέτησης για τα ερωτήματα ανάγνωσης και ενημέρωσης, για όλα τα μεγέθη της συστοιχίας με όλα τα είδη φορτίων.



Σχήμα 14: Μέσος συνολικός ρυθμός εξυπηρέτησης ερωτημάτων ανά είδος φορτίου

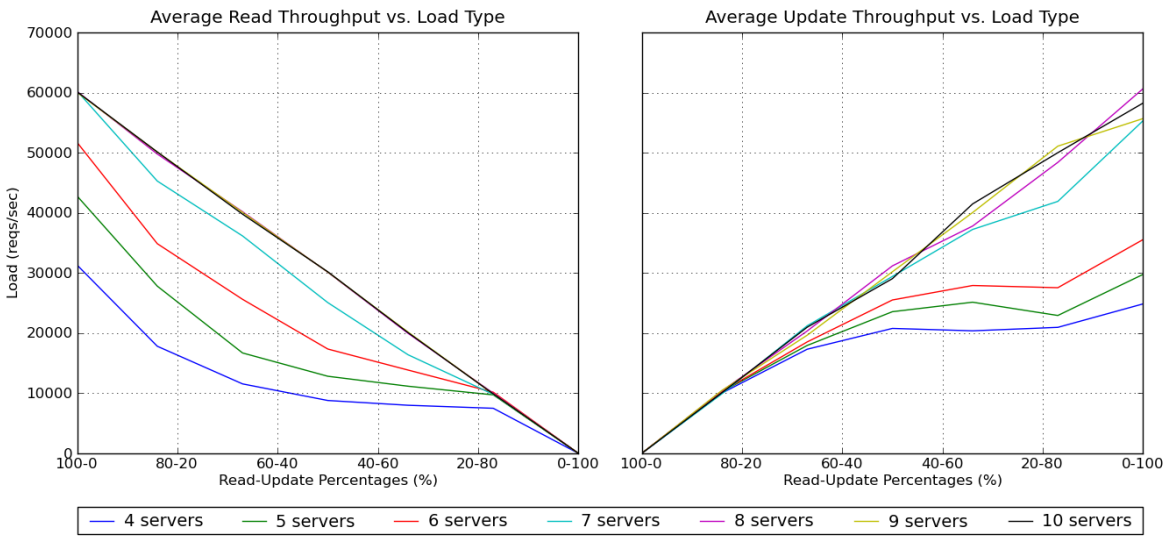
Ο ρυθμός εξυπηρέτησης πιάνει τον επιθυμητό στόχο των 60K αιτημάτων/δευτ. για οποιοδήποτε είδος φορτίου όταν η συστοιχία αποτελείται από 8 ή περισσότερους κόμβους. Για μικρότερο αριθμό κόμβων δεν μπορεί να εξυπηρετήσει άμεσα όλα τα

ερωτήματα που δέχεται λόγω περιορισμένης υπολογιστικής ισχύος, όπως αναφέραμε και στην προηγούμενη ενότητα.

Στην περίπτωση του μικρού αριθμού κόμβων (4-7), αξίζει να σημειωθεί η μικρή μείωση του ρυθμού εξυπηρέτησης καθώς αλλάζει το είδος του φορτίου και αυξάνονται τα αιτήματα ενημέρωσης ενώ μειώνονται της ανάγνωσης. Η μείωση της απόδοσης του συστήματος όταν αυξάνεται το ποσοστό των ερωτημάτων ενημέρωσης ενώ δέχεται πολύ μεγάλο φόρτο εργασίας είναι αναμενόμενη, αφού η διαδικασία ενημέρωσης κελιών απαιτεί συνεχή εγγραφή δεδομένων στα MemStores (στη μνήμη) και όταν αυτά γεμίζουν στο δίσκο σε αντίθεση με τις λειτουργίες ανάγνωσης που δεν απαιτούν κάτι περισσότερο από απλή πρόσβαση στη μνήμη. Δηλαδή η περιορισμένη διαθέσιμη μνήμη σε συνδυασμό με το μεγάλο φορτίο αιτημάτων ενημέρωσης οδηγεί σε συχνές εγγραφές δεδομένων από τα MemStores στο δίσκο, γεγονός που επιβαρύνει το σύστημα. Επιπλέον, όσο αυξάνεται το ποσοστό των αιτημάτων ενημέρωσης τόσο περισσότερες πιθανότητες έχουμε να ενημερώνονται τα δεδομένα τα οποία διαβάζουμε. Να υπενθυμίσουμε εδώ ότι τα ερωτήματα ανάγνωσης εφαρμόζονται σε 20.000 εγγραφές οι οποίες πιθανότατα ανήκουν σε διαφορετικά blocks που σημαίνει ότι έχουν μεταφερθεί 20.000 blocks στην BlockCache ή $20.000\text{blocks} \cdot \frac{64\ \text{KB/block}}{1{,}5\ \text{KB/record}} = 840.000$ εγγραφές δηλαδή 4,2% του συνόλου των δεδομένων. Αντίθετα τα ερωτήματα ενημέρωσης γίνονται σε όλα τα δεδομένα και όσο αυξάνεται ο ρυθμός αποστολής τους τόσο πιο πιθανό είναι να ενημερώνονται κάποιες από τις 840.000 εγγραφές που βρίσκονται στη BlockCache υποχρεώνοντας την HBase να ξαναφέρνει από το δίσκο τα αλλαγμένα blocks γεγονός που επιβαρύνει επιπλέον το σύστημα και έχει εμφανώς αρνητική επίπτωση στην απόδοση για τα ερωτήματα ανάγνωσης.

Παρόλα αυτά, οι λειτουργίες ενημέρωσης δεν επιβάλλουν πολύ μεγαλύτερο μέγεθος συστοιχίας για την επίτευξη του επιθυμητού ρυθμού εξυπηρέτησης αφού με βάση το διάγραμμα η συστοιχία πιάνει τον επιθυμητό ρυθμό με 7 κόμβους όταν το φορτίο έχει μόνο ερωτήματα ανάγνωσης ενώ για όλα τα υπόλοιπα είδη φορτίου (με οποιοδήποτε ποσοστό ενημέρωσης) απαιτεί μόλις ένα κόμβο παραπάνω, δηλαδή 8 κόμβους. Άρα τελικά η απόδοση του συστήματος επηρεάζεται, αλλά όχι σε πολύ μεγάλο βαθμό, από τις λειτουργίες ενημέρωσης.

Στο επόμενο σχήμα φαίνεται ξεχωριστά ο ρυθμός εξυπηρέτησης για τα ερωτήματα ανάγνωσης και ενημέρωσης.

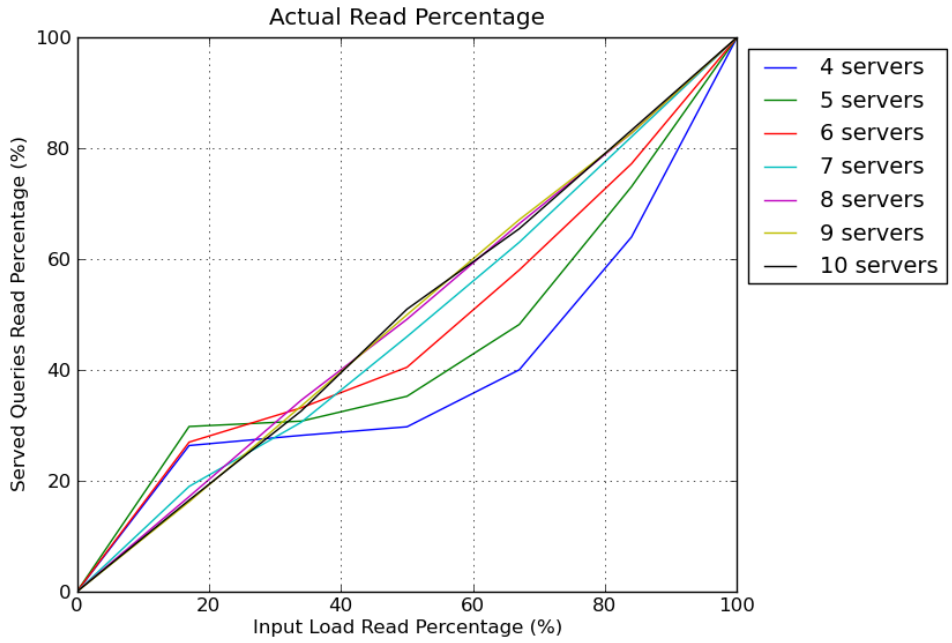


Σχήμα 15: Μέσος ρυθμός εξυπηρέτησης ερωτημάτων ανάγνωσης και ενημέρωσης ανά είδος φορτίου

Στο σχήμα 12 επιβεβαιώνεται η ανάλυσή μας. Αν παρατηρήσουμε τις καμπύλες που αφορούν τα μικρά μεγέθη της συστοιχίας βλέπουμε ότι για τα ερωτήματα ενημέρωσης έχουν διαφορετική μορφή σε σχέση με τα ερωτήματα ανάγνωσης. Συγκεκριμένα για τα ερωτήματα ανάγνωσης έχουμε πολύ μικρότερη απόδοση όταν έχουμε υψηλά ποσοστά ερωτημάτων ενημέρωσης σε σχέση με αυτή που έχουν τα ερωτήματα ενημέρωσης με υψηλά ποσοστά ερωτημάτων ανάγνωσης. Αυτό δείχνει ότι η ανάγνωση επηρεάζεται αρνητικά από τις λειτουργίες ενημέρωσης για τους λόγους που εξηγήσαμε στην προηγούμενη παράγραφο. Επίσης, αν συγκρίνουμε την απόδοση των ερωτημάτων ανάγνωσης στα πολύ χαμηλά ως μηδενικά ποσοστά αιτημάτων ενημέρωσης με την απόδοση των ερωτημάτων ενημέρωσης στα πολύ χαμηλά ποσοστά αιτημάτων ανάγνωσης παρατηρούμε ότι στην ανάγνωση έχουμε σημαντική αύξηση στο ρυθμό εξυπηρέτησης σε αντίθεση με την ενημέρωση που παρουσιάζει μια μικρή μόνο βελτίωση. Αυτό επιβεβαιώνει ότι απαιτούνται περισσότεροι πόροι μνήμης (δηλαδή μεγαλύτερος αριθμός κόμβων) για την εξυπηρέτηση μεγάλου φορτίου λειτουργιών ενημέρωσης και η έλλειψή τους οδηγεί με τον τρόπο που εξηγήσαμε προηγουμένως σε μειωμένη απόδοση. Η ανάγνωση αντιθέτως δεν έχει τέτοια εξάρτηση από τη μνήμη (εφόσον φροντίσαμε με τις ρυθμίσεις που πραγματοποιήσαμε να μην γεμίζει η BlockCache) γι' αυτό παρατηρούμε και καλύτερη απόδοση.

Στο σχήμα 13 που ακολουθεί φαίνεται επίσης η επίδραση της ενημέρωσης στην ανάγνωση. Στον οριζόντιο άξονα έχουμε το ποσοστό ερωτημάτων ανάγνωσης στο φορτίο

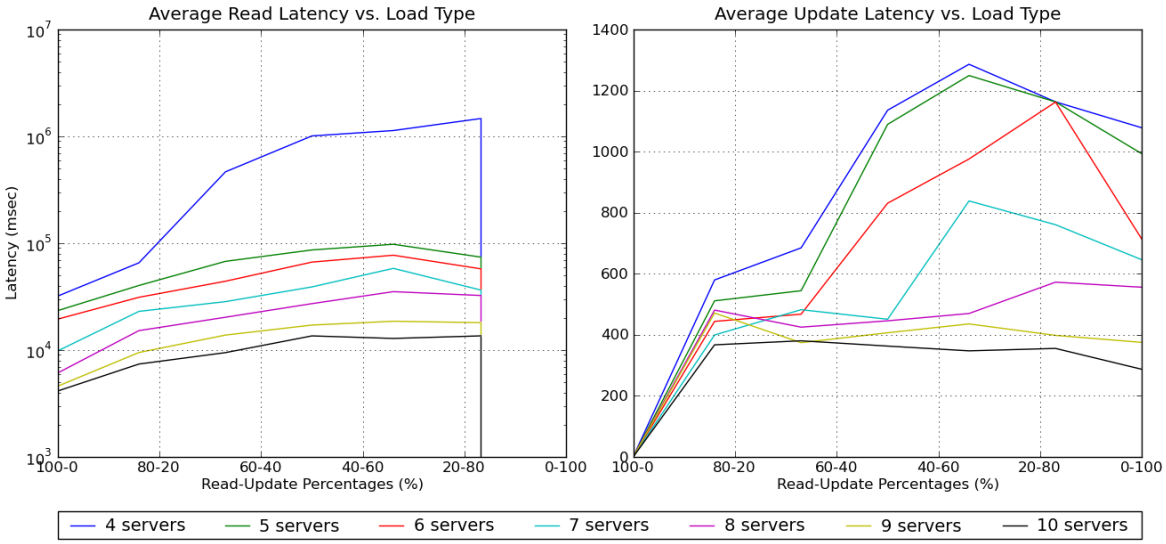
που στέλνουν οι πελάτες στη συστοιχία και στον κατακόρυφο άξονα το ποσοστό ερωτημάτων ανάγνωσης στο σύνολο των ερωτημάτων που εξυπηρετούνται από τη συστοιχία.



Σχήμα 16: Πρακτικά εξυπηρετούμενο ποσοστό ερωτημάτων ανάγνωσης

Σε γενικές γραμμές βλέπουμε ότι για τα μικρά μεγέθη της συστοιχίας, που όπως γνωρίζουμε τα ερωτήματα δεν εξυπηρετούνται με τον ίδιο ρυθμό που στέλνονται, δεν εξυπηρετείται ίδιο μέρος των ερωτημάτων ενημέρωσης και ανάγνωσης. Για υψηλά ποσοστά ερωτημάτων ανάγνωσης στο φορτίο εισόδου η συστοιχία εξυπηρετεί μικρότερο μέρος ερωτημάτων ανάγνωσης και μεγαλύτερο μέρος ενημέρωσης ενώ το αντίθετο συμβαίνει για τα χαμηλά ποσοστά ερωτημάτων ανάγνωσης στο φορτίο εισόδου (<30%). Στην εξυπηρέτηση μικρότερου μέρους ερωτημάτων ανάγνωσης παίζει ρόλο η αρνητική επίδραση από τις λειτουργίες ενημέρωσης, όπως εξηγήσαμε προηγουμένως, ενώ όταν μειωθούν πολύ οι λειτουργίες ανάγνωσης γίνεται μεγαλύτερο το μέρος τους γιατί δεν επηρεάζονται πλέον εύκολα από την ενημέρωση και η συστοιχία κυρίως δεν μπορεί να ανταπεξέλθει στον μεγάλο ρυθμό εφαρμογής ενημερώσεων.

Τέλος, αξίζει να συγκρίνουμε τον χρόνο αναμονής για τα ερωτήματα ανάγνωσης και ενημέρωσης, ο οποίος όπως εξηγήσαμε στις ενότητες 4.1.8-9 αναμένουμε να είναι αρκετά μεγαλύτερος για τα ερωτήματα ανάγνωσης.



Σχήμα 17: Μέσος χρόνος αναμονής ερωτημάτων ανάγνωσης και ενημέρωσης ανά είδος φορτίου

Πράγματι ο χρόνος αναμονής των ερωτημάτων ανάγνωσης ήταν πολύ μεγαλύτερος από αυτόν των ερωτημάτων ενημέρωσης με ένα λόγο της τάξης του 10^4. Η διακύμανση του χρόνου αναμονής για τα ερωτήματα ανάγνωσης ήταν επίσης πολύ μεγάλη και γι' αυτό στο διάγραμμα του χρόνου αναμονής ανάγνωσης έχουμε λογαριθμικό άξονα. Γενικά παρατηρούμε ότι ο χρόνος αναμονής μειώνεται όταν προσθέτουμε εξυπηρετητές, πράγμα που είναι λογικό γιατί έχουμε περισσότερους υπολογιστικούς πόρους που εξυπηρετούν πιο γρήγορα τα ερωτήματα.

Ο χρόνος αναμονής των ερωτημάτων ανάγνωσης βλέπουμε ότι αυξάνεται αρκετά καθώς αυξάνεται το ποσοστό αιτημάτων ενημέρωσης και αυτό εξηγείται από την αρνητική επιρροή που είδαμε ότι έχει η ενημέρωση στην ανάγνωση. Αντίστοιχα ο χρόνος αναμονής των ερωτημάτων ενημέρωσης βλέπουμε ότι αυξάνεται αρκετά όπως αυξάνεται το ποσοστό τους κυρίως για τα μικρά μεγέθη της συστοιχίας, καθώς όπως εξηγήσαμε και προηγουμένως επιβαρύνεται αρκετά η συστοιχία σε αυτές τις περιπτώσεις.

Επομένως τον πιο σημαντικό ρόλο στη διαμόρφωση του συνολικού χρόνου αναμονής παίζει ο χρόνος αναμονής των ερωτημάτων ανάγνωσης, λόγω του μεγέθους του, και αλλάζει σημαντικά όταν αλλάζει το είδος φορτίου.

5.2.3 Ορισμός της συνάρτησης ανταμοιβής για το σύστημα αποφάσεων του TIRAMOLA

Στη συνέχεια πρέπει να τροφοδοτήσουμε τον TIRALOMA με μια κατάλληλα επιλεγμένη συνάρτηση ανταμοιβής ώστε να λαμβάνει σε κάθε περίπτωση τη σωστή κατ' εμάς απόφαση. Δηλαδή για κάθε μέγεθος και είδος φορτίου που έχουμε συμπεριλάβει στο training set θέλουμε να επιλέξει το μέγεθος της συστοιχίας που εμείς θα ορίσουμε με βάση την απόδοση που καταγράφεται στο training set ενσωματώνοντας κατάλληλα τα καταγεγραμμένα μετρικά στη συνάρτησης ανταμοιβής. Τα διαθέσιμα μετρικά υπενθυμίζουμε ότι είναι: ο ρυθμός εξυπηρέτησης (*thr*) και ο χρόνος αναμονής (*lat*) των ερωτημάτων (συνολικά), ο αριθμός VMs, η χρήση της cpu και (ύστερα από τις τροποποιήσεις που κάναμε) ο ρυθμός εξυπηρέτησης των ερωτημάτων ανάγνωσης (thr_r) και ενημέρωσης (thr_u) ξεχωριστά και ο χρόνος αναμονής των ερωτημάτων ανάγνωσης (lat_r) και ενημέρωσης (lat_u) ξεχωριστά.

Κατασκευάσαμε 2 διαφορετικές συναρτήσεις ανταμοιβής για να συγκρίνουμε τη συμπεριφορά του TIRAMOLA: Η μια, στα πρότυπα της βέλτιστης συνάρτησης που υπάρχει στην τελευταία αξιολόγηση του TIRAMOLA, λαμβάνει υπόψη το συνολικό ο ρυθμό εξυπηρέτησης, το συνολικό χρόνο αναμονής και το κόστος (C) των VMs και έχει την εξής μορφή: $r_1(\mathrm{s}) = A \cdot thr - B \cdot lat - C \cdot |\,\mathrm{VMs}\,|$. Η δεύτερη, που είναι η προτεινόμενη από εμάς υλοποίηση, είναι παρόμοια με την πρώτη αλλά λαμβάνει επιπλέον υπόψη το διαφορετικό είδος φορτίου διαχωρίζοντας τον ρυθμό εξυπηρέτησης και τον χρόνο αναμονής για τα 2 είδη ερωτημάτων: $r_2(\mathrm{s}) = A_r \cdot thr_r + A_u \cdot thr_u - B_r \cdot lat_r - B_u \cdot lat_u - C \cdot |\,\mathrm{VMs}\,|$. Ο TIRAMOLA που χρησιμοποιεί την πρώτη συνάρτηση (r_1) θα λέμε στο εξής ότι είναι *load unaware* καθώς δεν λαμβάνει υπόψη το είδος του φορτίου παρά μόνο τη συνολική απόδοση ενώ με τη δεύτερη συνάρτηση (r_2) είναι αντιθέτως *load aware*.

Να σημειώσουμε ότι για τη χρήση της συνάρτησης r_2 χρειάστηκε τροποποίηση και του συστήματος αποφάσεων του TIRAMOLA. Με βάση αυτή τη συνάρτηση συγκροτούνται 4-διάστατα datasets αντί για 2-διάστατα που έχουμε με τη συνάρτηση r_1 (χωρίς να λαμβάνουμε υπόψη τον αριθμό των κόμβων), τα οποία θα τροφοδοτηθούν στο μηχανισμό ομαδοποίησης. Όπως περιγράψαμε στην ενότητα 3.1.1 για το 2-διάστατο πρόβλημα, όταν το σύστημα αποφασίζει για την πιθανή αλλαγή μεγέθους με βάση τον τρέχοντα χρόνο αναμονής ($\mathrm{lat_c}$) και ρυθμό εξυπηρέτησης ($\mathrm{thr_c}$), επιλέγει τα data points που βρίσκονται μέσα στη λωρίδα που ορίζεται γύρω από το $\mathrm{thr_c}$ για να τροφοδοτηθούν στο 2-διάστατο

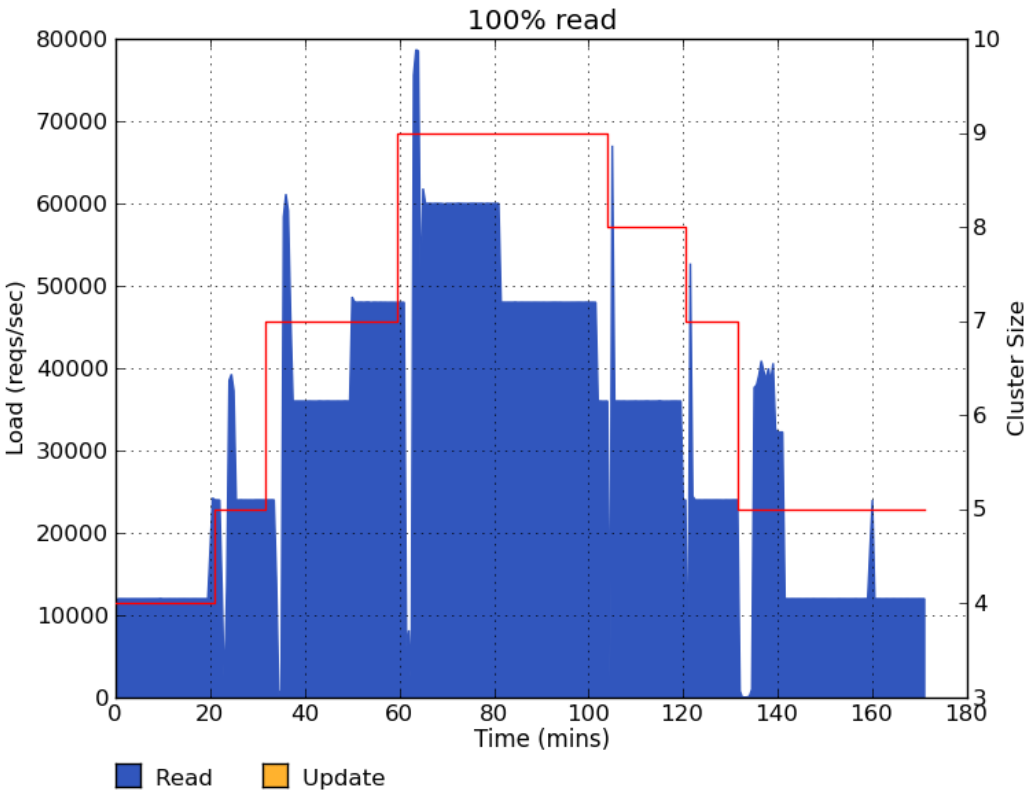
μηχανισμό ομαδοποίησης. Στην περίπτωση του 4-διάστατου προβλήματος που χρησιμοποιεί τον τρέχοντα χρόνο αναμονής ανάγνωσης (lat_{r_c}), χρόνο αναμονής ενημέρωσης (lat_{u_c}), ρυθμό εξυπηρέτησης ερωτημάτων ανάγνωσης (thr_{r_c}) και ερωτημάτων ενημέρωσης (thr_{u_c}), επιλέγονται τα data points που βρίσκονται μέσα στην περιοχή που ορίζεται από τα thr_{r_c} και thr_{u_c}, δηλαδή σε ένα τετράγωνο σωλήνα αντί για μια λωρίδα, και τροφοδοτούνται σε 4-διάστατο μηχανισμό ομαδοποίησης.

Αρχικά βρήκαμε τις επιθυμητές παραμέτρους για τη συνάρτηση r_1 όταν έχουμε φορτίο με ερωτήματα ανάγνωσης μόνο ώστε να επιβεβαιώσουμε την σωστή λειτουργία του TIRAMOLA και τα υπάρχοντα αποτελέσματα από την προηγούμενη αξιολόγηση. Με βάση τα μεγέθη των μετρικών που καταγράφηκαν στο training set επιλέξαμε για τις παραμέτρους τις τιμές A=0.001, B=0.002 και C=1.4. Διατηρήσαμε αυτή τη συνάρτηση με τις συγκεκριμένες τιμές για να ελέγξουμε την συμπεριφορά του TIRAMOLA όταν θα αλλάζει το είδος του φορτίου.

Για τη συνάρτηση ανταμοιβής r_2 ήταν πιο δύσκολος ο προσδιορισμός των βέλτιστων παραμέτρων ώστε να έχουμε τα επιθυμητά αποτελέσματα σε όλα τα είδη φορτίου. Δηλαδή όσο πιο πολλές περιπτώσεις πρέπει να καλύπτει η συνάρτηση τόσο πιο δύσκολος είναι ο προσδιορισμός της. Οι καλύτερες τιμές που βρήκαμε για τις παραμέτρους της συνάρτησης r_2 ύστερα από επανειλημμένη παραμετροποίηση και πειραματισμό είναι: A_r=0.0005, A_u=0.0005, B_r=0.0003, B_u=0.0007 και C=1.2.

5.2.4 Αξιολόγηση της λειτουργίας του TIRAMOLA με διαφορετικά είδη φορτίου

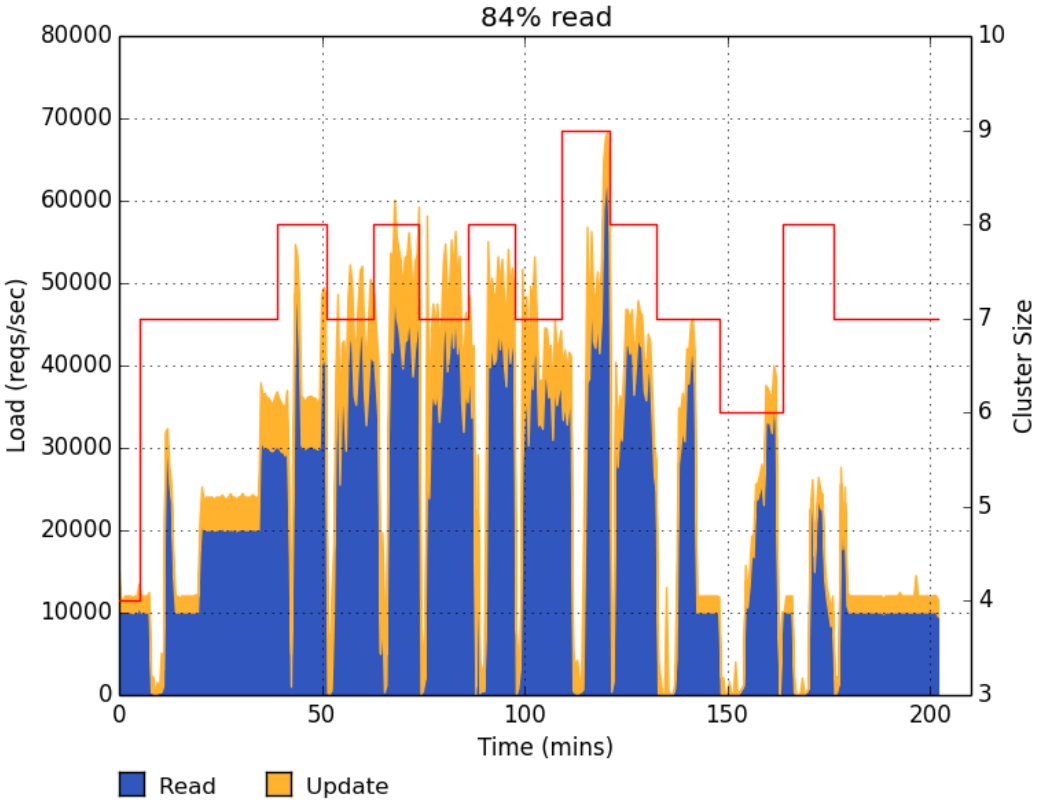
Ο TIRAMOLA παίρνει μια νέα απόφαση κάθε 10 λεπτά. Για να παρατηρήσουμε τη συμπεριφορά του εφαρμόζουμε στη συστοιχία φορτίο ημιτονοειδούς μορφής με περίοδο 160 λεπτών (περίπου 2,5 ώρες) που παράγεται από τα 10 VMs-πελάτες (παρόμοιο δηλαδή σε πλάτος με αυτό που χρησιμοποιήσαμε για την κατασκευή του training set). Αρχικά χρησιμοποιούμε τη συνάρτηση r_1 για τη λήψη των αποφάσεων και επιλέγουμε να έχουμε μόνο ερωτήματα ανάγνωσης στο φορτίο. Στο επόμενο διάγραμμα παρατηρούμε πώς αλλάζει ο TIRAMOLA το μέγεθος της συστοιχίας ανάλογα με το εφαρμοζόμενο φορτίο σε διάστημα 2,5 ωρών.



Σχήμα 18: Η λειτουργία του load unaware TIRAMOLA με 100% ερωτήματα ανάγνωσης

Η αποφάσεις αλλαγής μεγέθους ήταν σωστές και βοήθησαν τη συστοιχία να ανταπεξέλθει στις απαιτήσεις του συνεχώς αυξανόμενου φορτίου προσθέτοντας κόμβους ενώ όταν άρχισε να μειώνεται το φορτίο άρχισε ορθώς να βγάζει κόμβους αφού δεν τους χρειαζόταν πλέον το σύστημα. Για τις αποφάσεις που έλαβε σε αυτή την περίπτωση, ο TIRAMOLA ήταν load unaware δηλαδή δεν γνώριζε το είδος των ερωτημάτων παρά μόνο το μέγεθος του φορτίου.

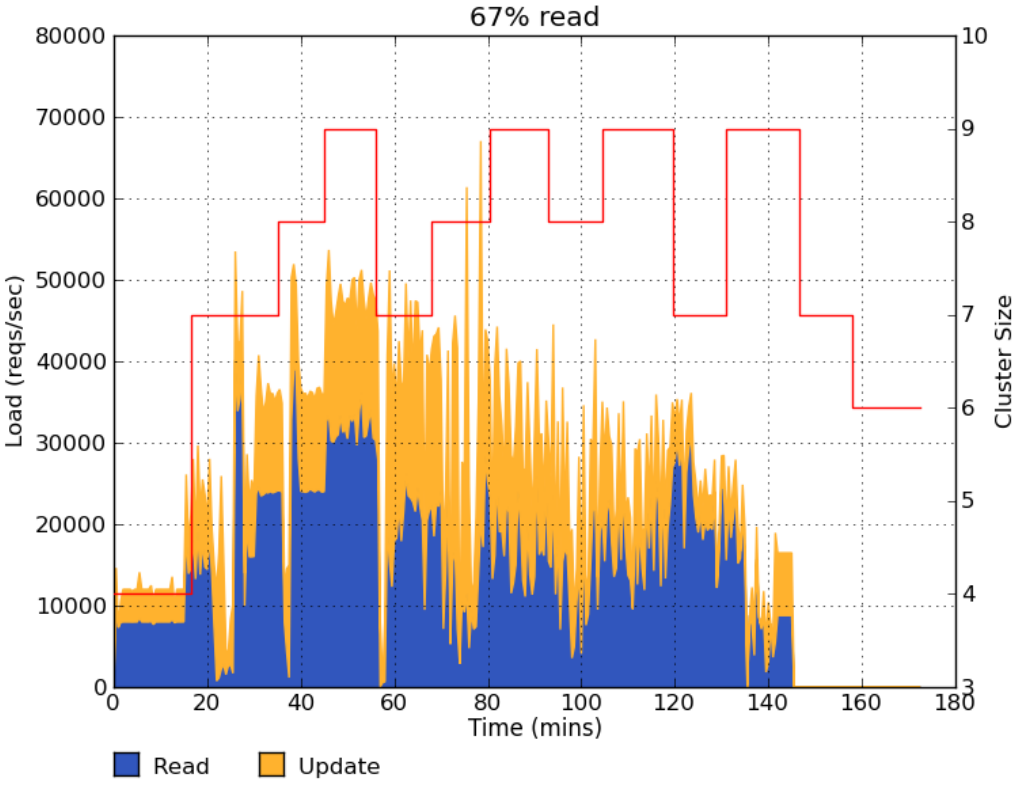
Αν επιχειρήσουμε τώρα να αλλάξουμε μόνο το είδος του φορτίου εισόδου αντικαθιστώντας ένα μικρό μέρος των ερωτημάτων ανάγνωσης με ενημέρωσης (το 16% για την ακρίβεια) και εφαρμόσουμε το φορτίο για 3 ώρες στη συστοιχία έχουμε τα εξής αποτελέσματα:



Σχήμα 19: Η λειτουργία του load unaware TIRAMOLA με 84% ανάγνωση και 16% ενημέρωση

Παρατηρούμε ότι πλέον ο TIRAMOLA δεν λειτουργεί καθόλου αποδοτικά. Συγκεκριμένα στην πρώτη απόφαση που παίρνει αυξάνει κατευθείαν το μέγεθος της συστοιχίας στους 7 κόμβους και στη συνέχεια το μέγεθός της δεν αλλάζει σημαντικά αλλά εμφανίζει μόνο συχνές και μικρές διακυμάνσεις. Η ιδανικά επιθυμητή συμπεριφορά θα ήταν βεβαίως παρόμοια με αυτή του σχήματος 18. Από αυτό το σχήμα γίνεται κατανοητό ότι ο load unaware TIRAMOLA δεν δύναται να πάρει τις σωστές αποφάσεις όταν αλλάξει το είδος του φορτίου παρόλο που έχουμε παρόμοιο συνολικό μέγεθος φορτίου. Η συνάρτηση ανταμοιβής r_1 δεν λαμβάνει υπόψη το είδος του φορτίου παρά μόνο τον συνολικό ρυθμό εξυπηρέτησης και τον συνολικό χρόνο αναμονής επομένως τα data points που συμπεριλαμβάνονται στη thr-λωρίδα προέρχονται από όλα είδη φορτίων που εμφανίζουν παρόμοιο συνολικό ρυθμό εξυπηρέτησης. Αυτό σημαίνει ότι αλλοιώνεται η επιλογή του ενδεικτικού χρόνου αναμονής και τα εκτιμώμενα κέρδη προκύπτουν αρκετά διαφορετικά σε σχέση με το τρέχων κέρδος οδηγώντας σε λάθος επιλογή κατάστασης. Στη διαμόρφωση του παραπάνω αποτελέσματος ο χρόνος αναμονής παίζει αρκετά σημαντικό ρόλο αφού αλλάζει σημαντικά όταν αλλάζει το είδος του φορτίου, όπως παρατηρήσαμε στην ενότητα 5.2.2.

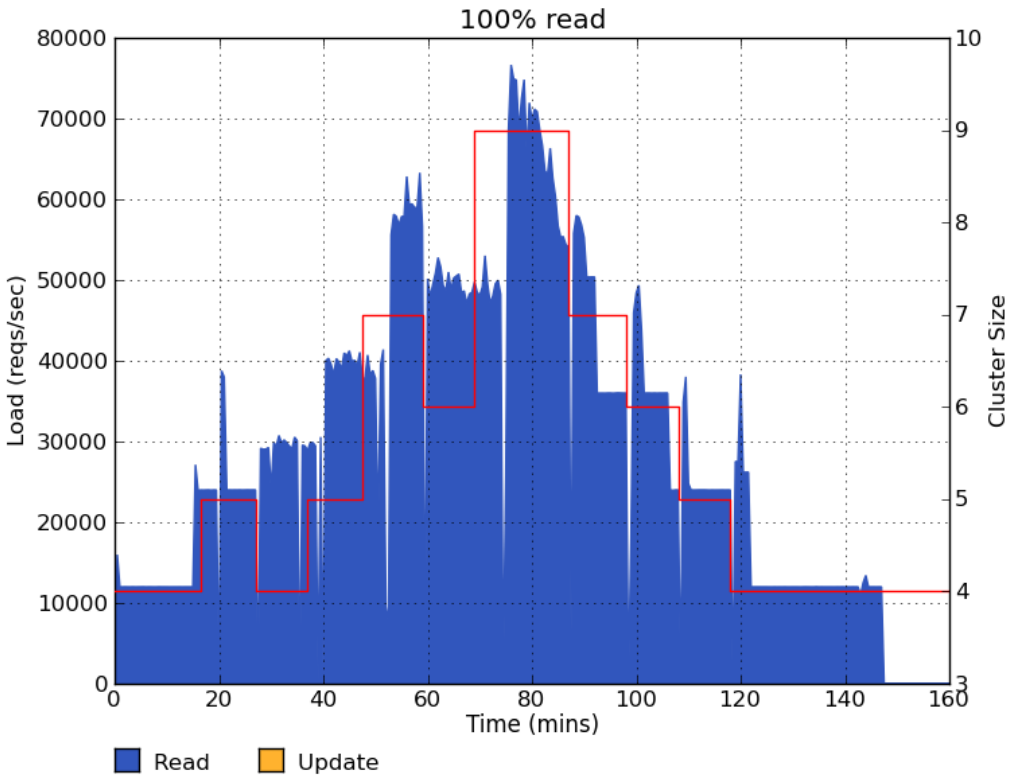
Αν αυξήσουμε λίγο ακόμα το ποσοστό αιτημάτων ενημέρωσης (σχήμα 20) παρατηρούμε την ίδια λανθασμένη συμπεριφορά, γεγονός που επιβεβαιώνει την ανεπάρκεια της χρησιμοποιούμενης συνάρτησης ανταμοιβής.



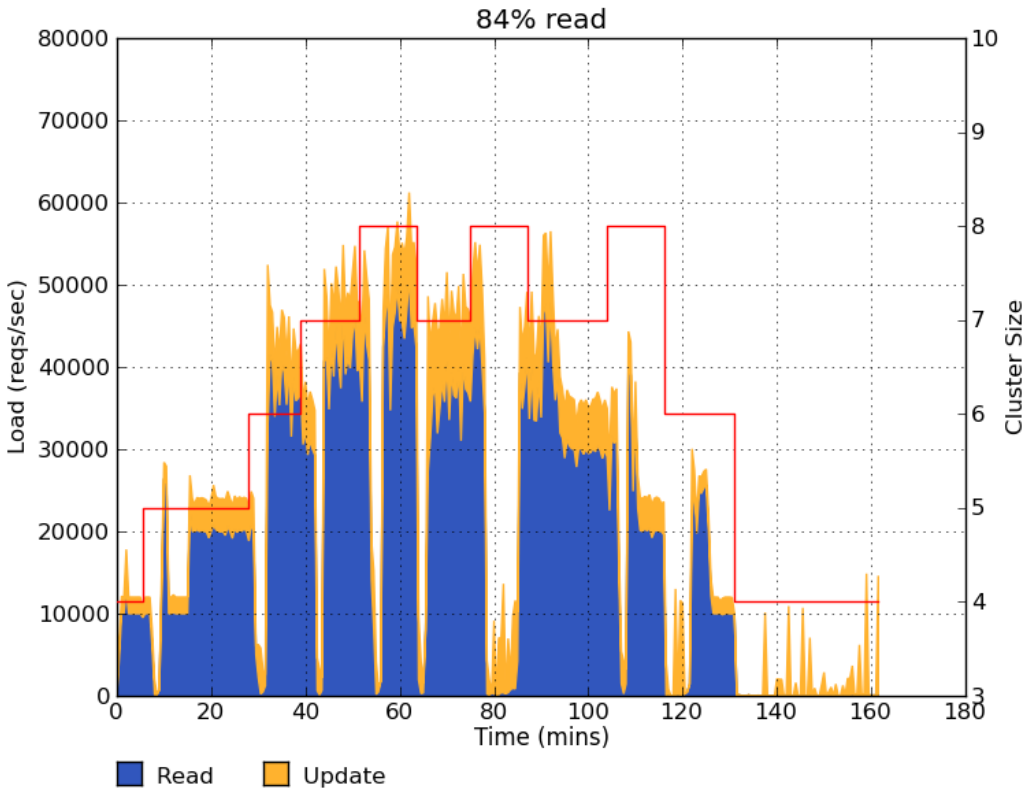
Σχήμα 20: Η λειτουργία του load unaware TIRAMOLA με 67% ανάγνωση και 33% ενημέρωση

Μάλιστα επειδή η συστοιχία δέχεται με πιο αυξημένο ρυθμό πλέον αιτήματα ενημέρωσης, βλέπουμε ότι η απόδοση μειώνεται αρκετά μετά από την πρώτη ώρα εφαρμογής του φορτίου καθώς το σύστημα επιβαρύνεται από το συνεχόμενο flushing των MemStores που έχουν γεμίσει.

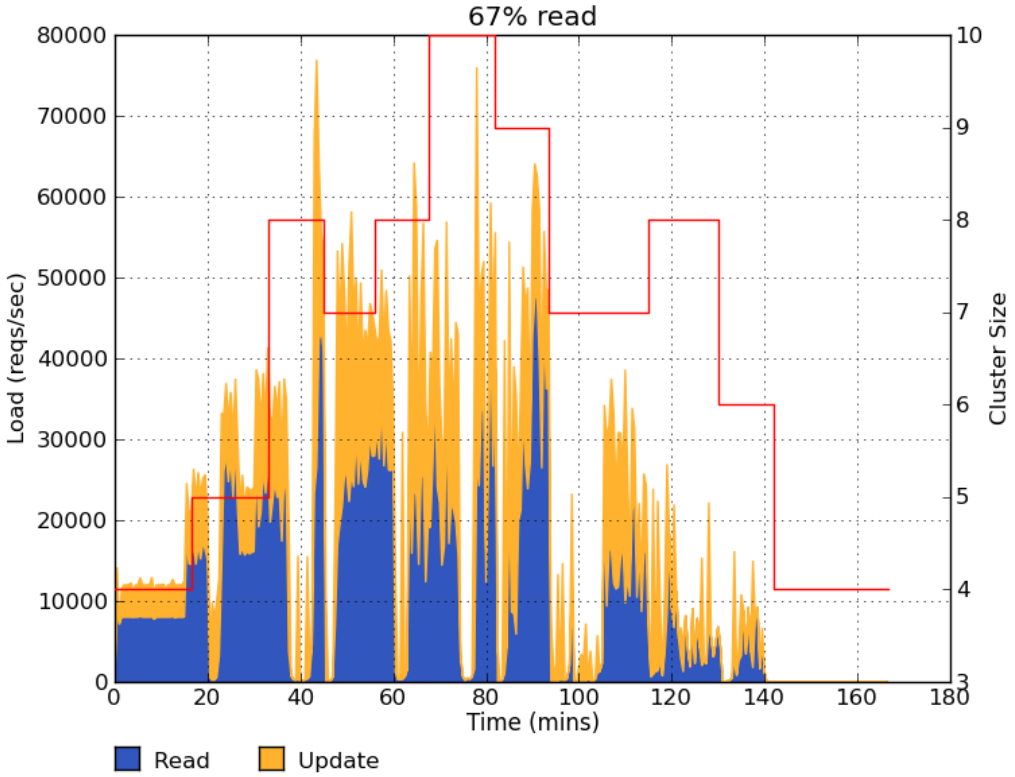
Δεδομένου του αρκετά διαφορετικού χρόνου αναμονής για διαφορετικά είδη φορτίων απαιτείται επομένως η χρήση μιας συνάρτησης ανταμοιβής που λαμβάνει υπόψη το είδος φορτίου, όπως η συνάρτηση r_2, ώστε η απόφαση να λαμβάνεται χρησιμοποιώντας στο μηχανισμό ομαδοποίησης μόνο τα δεδομένα που αφορούν το συγκεκριμένο είδος φορτίου. Αν επαναλάβουμε τα παραπάνω πειράματα χρησιμοποιώντας τη συνάρτηση r_2 έχουμε τα εξής αποτελέσματα:



Σχήμα 21: Η λειτουργία του load aware TIRAMOLA με 100% ερωτήματα ανάγνωσης



Σχήμα 22: Η λειτουργία του load aware TIRAMOLA με 84% ανάγνωση και 16% ενημέρωση



Σχήμα 23: Η λειτουργία του load aware TIRAMOLA με 67% ανάγνωση και 33% ενημέρωση

Παρατηρούμε ότι και στα τρία ποσοστά ο load aware TIRAMOLA εμφανίζει πολύ καλύτερη συμπεριφορά σε σύγκριση με τον load unaware. Συγκεκριμένα, αυξάνει ορθώς το μέγεθος της συστοιχίας όταν αυξάνεται το μέγεθος του φορτίου και επίσης το μειώνει όταν μειώνεται το φορτίο σε όλα τα είδη φορτίου που χρησιμοποιήσαμε. Η λειτουργία του μπορεί να μην είναι ιδανική όπως αυτή του σχήματος 18 (που επιτεύχθηκε επειδή είχαμε βελτιστοποιήσει τη συνάρτηση r_1 συγκεκριμένα για χρήση με 100% ανάγνωση) αλλά είναι πλήρως αποδοτική και διαχειρίζεται επιτυχώς πολλά διαφορετικά είδη φορτίου.

5.3 Σύνοψη συμπερασμάτων αξιολόγησης

Από την πειραματική αξιολόγηση του συστήματος του TIRAMOLA παρατηρήσαμε ότι έχει τη δυνατότητα να λειτουργήσει σωστά σε περίπτωση που στη συστοιχία εφαρμοστεί φορτίο με διαφορετικά είδη ερωτημάτων. Για αυτό απαιτείται όμως τροποποίηση της συνάρτησης ανταμοιβής και του συστήματος λήψης αποφάσεων με τον τρόπο που περιγράψαμε στην ενότητα 5.2.4 ώστε να μπορεί να αντιλαμβάνεται εμμέσως την ύπαρξη

διαφορετικών ειδών ερωτημάτων και να το λαμβάνει υπόψη. Επίσης, για την εύρεση των βέλτιστων παραμέτρων για τη συνάρτηση ανταμοιβής απαιτήθηκε επανειλημμένη παραμετροποίηση και πειραματισμός. Λόγω της διαφοροποίησης της συνάρτησης με την προσθήκη επιπλέον παραμέτρων και την δημιουργία μεγαλύτερου training set που περιλαμβάνει μετρικά από αρκετά διαφορετικές καταστάσεις λειτουργίας (όταν δέχεται η συστοιχία διαφορετικό είδος φορτίου), είναι πιο δύσκολος ο προσδιορισμός όλων των παραμέτρων ώστε να έχουμε την επιθυμητή συμπεριφορά σε τόσες διαφορετικές περιπτώσεις.

Κεφάλαιο 6

Επίλογος

6.1 Σύνοψη και συμπεράσματα

Η ανάπτυξη του cloud computing προσφέρει πολλές νέες δυνατότητες, μια από τις οποίες είναι η ελαστική δέσμευση πόρων από χρήστες μέσα από πλατφόρμες που προσφέρουν υπηρεσίες IaaS για την εκτέλεση των εφαρμογών τους. Παρόλο που οι υπηρεσίες IaaS έχουν αποκτήσει μεγάλη δημοτικότητα, η αυτόματη διαχείριση του νέφους συνεχίζει να είναι μια από τις μεγαλύτερες προκλήσεις καθώς δεν είναι εύκολο για τους χρήστες να αυξομειώνουν τους πόρους ώστε να ικανοποιούνται οι απαιτήσεις της εφαρμογής τους. Στην παρούσα εργασία αξιολογήσαμε την επίδοση του TIRAMOLA, ενός συστήματος που επιτρέπει την αυτόματη αυξομείωση του μεγέθους μιας NoSQL συστοιχίας με βάση μια τακτική ορισμένη από το χρήστη, σε διαφορετικές συνθήκες λειτουργίας της συστοιχίας.

Αρχικά ορίσαμε κατάλληλα τις παραμέτρους λειτουργίας της HBase και το φορτίο που παράγεται από το YCSB ώστε να επιτύχουμε τη μέγιστη απόδοση στη συστοιχία με οποιοδήποτε είδος φορτίου. Στη συνέχεια, συλλέξαμε μετρικά εφαρμόζοντας στη συστοιχία φορτία διαφορετικού πλάτους και είδους (ποσοστά ερωτημάτων ανάγνωσης και ενημέρωσης), τα οποία χρησιμοποιήσαμε για την εκπαίδευση του TIRAMOLA. Επίσης χρησιμοποιήσαμε αυτά τα μετρικά για να μελετήσουμε αναλυτικά την απόδοση της συστοιχίας πριν επιχειρήσουμε να ορίσουμε την τακτική που θέλουμε να χρησιμοποιεί ο TIRAMOLA για τη λήψη της απόφασης. Παρατηρώντας την απόδοση του συστήματος διαπιστώσαμε ότι καθώς αλλάζει το είδος του φορτίου, και συγκεκριμένα καθώς αυξάνεται το ποσοστό των αιτημάτων ενημέρωσης και μειώνεται αυτό της ανάγνωσης, μειώνεται λίγο ο ρυθμός εξυπηρέτησης ερωτημάτων και αυξάνεται αρκετά ο χρόνος αναμονής, ο οποίος διαμορφώνεται κυρίως από τα ερωτήματα ανάγνωσης. Βλέποντας αυτές τις διαφορές προτείναμε τη χρήση μιας συνάρτησης ανταμοιβής που ενσωματώνει ξεχωριστά την απόδοση για τα δύο είδη ερωτημάτων και την τροποποίηση του συστήματος λήψης αποφάσεων του TIRAMOLA ώστε να μπορεί να διαχειρίζεται κατάλληλα τα μετρικά για τα διαφορετικά είδη ερωτημάτων. Συγκρίναμε

επίσης τα αποτελέσματα από τη χρήση αυτής της συνάρτησης με μια πιο απλή συνάρτηση που λαμβάνει υπόψη μόνο τη συνολική απόδοση του συστήματος.

Με βάση τα πειραματικά αποτελέσματα που παρουσιάστηκαν αποδεικνύεται η καταλληλότητα της προτεινόμενης μεθόδου για την ορθή λειτουργία του TIRAMOLA με διαφορετικά είδη φορτίων εισόδου. Με αυτό τον τρόπο μπορεί να λειτουργήσει σωστά υπό οποιεσδήποτε συνθήκες με διαφορετικό μέγεθος αλλά και είδος φορτίου, καθιστώντας τον ένα αποδοτικό και αξιόπιστο εργαλείο που μπορεί να καλύψει οποιαδήποτε ανάγκη του χρήστη.

6.2 Προτάσεις για μελλοντική έρευνα

Παρότι η χρήση της προτεινόμενης μεθόδου καθιστά πιο αποδοτική τη λειτουργία του TIRAMOLA σε περιπτώσεις που αλλάζει το είδος των ερωτημάτων, εισάγει και ένα βαθμό πολυπλοκότητας στον ορισμό της συνάρτησης ανταμοιβής, η οποία πρέπει να καλύπτει πολλές διαφορετικές περιπτώσεις φορτίου. Για την εύρεση των παραμέτρων απαιτήθηκε από τη μεριά μας να παρατηρήσουμε τη σχέση μεγέθους των μετρικών που χρησιμοποιήθηκαν στη συνάρτηση και να εκτελέσουμε πειραματικές δοκιμές για την βελτιστοποίησή τους. Στα πλαίσια των δυσκολιών που αντιμετωπίσαμε προτείνουμε την υλοποίηση στο άμεσο μέλλον ενός συστήματος, που χρησιμοποιώντας τα δεδομένα του training set και αφήνοντας το χρήστη να ορίσει σε κάθε περίπτωση το επιθυμητό μέγεθος της συστοιχίας (αφού προηγουμένως έχει μελετήσει την απόδοση που έχει καταγραφεί στο training set), θα υπολογίζει αυτόματα τις παραμέτρους για τη δοσμένη μορφή της συνάρτησης ανταμοιβής.

Ακόμα, στα πειράματα που εκτελέσαμε για να έχουμε τη μέγιστη απόδοση στην HBase συστοιχία μας φροντίσαμε τα δεδομένα τα οποία διαβάζουμε να χωράνε σε κάθε περίπτωση στη BlockCache και επιπλέον φορτώναμε πάντα τα δεδομένα στη μνήμη πριν εκτελέσουμε οποιοδήποτε φορτίο. Θα ήταν ενδιαφέρον να εξετάσουμε μελλοντικά τον τρόπο με τον οποίο αλλάζει η απόδοση κατά την προσθαφαίρεση κόμβων όταν τα δεδομένα δεν χωράνε εξαρχής στη BlockCache.

Τέλος, θα ήταν ενδιαφέρον να σχεδιαστούν και να μελετηθούν συναρτήσεις ανταμοιβής με μη-γραμμική μορφή για τη διαχείριση διαφορετικού είδους ερωτημάτων. Μια τέτοια συνάρτηση θα χρησιμοποιούσε παραδείγματος χάριν το ποσοστό των ερωτημάτων ενός συγκεκριμένου είδους ως πολλαπλασιαστικό παράγοντα σε συγκεκριμένα μετρικά,

δίνοντας τους κατάλληλο βάρος στη διαμόρφωση του κέρδους ανάλογα με το ποσοστό του συγκεκριμένου είδους στο συνολικό φορτίο. Δεν αποκλείεται μια συνάρτηση ανταμοιβής με τέτοια μορφή να δώσει ακόμα καλύτερα αποτελέσματα ως προς τη συμπεριφορά του TIRAMOLA με διαφορετικά είδη φορτίων.

Κεφάλαιο 7

Βιβλιογραφία

[1] Apache ZooKeeper. http://zookeeper.apache.org/

[2] Hadoop Distributed File System. http://hadoop.apache.org/hdfs/

[3] HBase Homepage. [Online]. Available: http://hbase.apache.org

[4] OpenStack: The open source, open standards cloud. [Online]. Available: http://openstack.org

[5] Tiramola open-source project. [Online]. Available: http://tiramola.googlecode.com

[6] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A Distributed Storage System for Structured Data," in *OSDI*, 2006.

[7] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking Cloud Serving Systems with YCSB," in *ACM SOCC*, 2010.

[8] L. George, *HBase: The Definitive Guide*. O'Reilly Media, 2011.

[9] Y. Jiang, *HBase Administration Cookbook*. Packt Publishing, 2012.

[10] I. Konstantinou, E. Angelou, D. Tsoumakos, C. Boumpouka, N. Koziris, and S. Sioutas, "TIRAMOLA: Elastic NoSQL Provisioning Through a Cloud Management Platform," in *SIGMOD*, 2012.

[11] I. Konstantinou, E. Angelou, C. Boumpouka, D. Tsoumakos, and N. Koziris, "On the Elasticity of NoSQL Databases over Cloud Management Platforms," in CIKM, 2011.

[12] D. Tsoumakos, I. Konstantinou, C. Boumpouka, S. Sioutas and N. Koziris, "Automated, Elastic Resource Provisioning for NoSQL Clusters Using TIRAMOLA", in *CCGrid*, 2013.

[13] M. L. Massie, B. N. Chun, and D. E. Culler, "The Ganglia Distributed Monitoring System: Design, Implementation, and Experience," *Parallel Computing*, vol. 30, no. 7, pp. 817–840, 2004.

[14] H. C. Lim, S. Babu, and J. S. Chase, "Automated Control for Elastic Storage," in *ICAC*, 2010.

[15] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin, and S. Babu, "Starfish: A Self-tuning System for Big Data Analytics," in *CIDR*, 2011.

[16] K. Tsakalozos, M. Roussopoulos, V. Floros, and A. Delis, “Nefeli: Hint-Based execution of workloads in clouds,” in *ICDCS*, 2010, pp. 74–85.

[17] K. Tsakalozos, H. Kllapi, E. Sitaridi, M. Roussopoulos, D. Paparas, and A. Delis, “Flexible Use of Cloud Resources through Profit Maximization and Price Discrimination,” in *ICDE*, 2011.

[18] A. Qureshi, R. Weber, H. Balakrishnan, J. Guttag, and B. Maggs, “Cutting the Electric Bill for Internet-Scale Systems,” in *SIGCOMM*, 2009.

[19] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao and F. Zhao, “Energy-Aware Server Provisioning and Load Dispatching for Connection-Intensive Internet Services,” in *NSDI*, 2008.

[20] Lamport, Leslie. “Paxos Made Simple.” ACM Sigact News 32, no. 4 (2001): 18–25.

[21] Prabhakaran, Vijayan, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. “Analysis and Evolution of Journaling File Systems.” In *USENIX Annual Technical Conference, General Track*, 105–120, 2005. http://www.usenix.org/event/usenix05/tech/general/full_papers/prabhakaran/prabhakaran_html/.

[22] T. White, *Hadoop: The Definitive Guide*. O’Reilly Media, 2010.

[23] F. Cruz, F. Maia, M. Matos, R. Oliveira, J. Paulo, J. Pereira and R. Vilaça, “MET: Workload aware elasticity for NoSQL,” in *ACM ECCS*, 2013.

[24] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, “A View of Cloud Computing,” Commun. ACM, vol. 53, no. 4, pp. 50–58, Apr. 2010.