ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Συγκριτική Αξιολόγηση Συστημάτων Μη Σχεσιακών Βάσεων Δεδομένων Τύπου Κλειδιού-Τιμής

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΙΩΑΝΝΗΣ Α. ΓΡΗΓΟΡΑΚΟΣ

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

Αθήνα, Οκτώβριος 2020

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

Συγκριτική Αξιολόγηση Συστημάτων Μη Σχεσιακών Βάσεων Δεδομένων Τύπου Κλειδιού-Τιμής

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΙΩΑΝΝΗΣ Α. ΓΡΗΓΟΡΑΚΟΣ

Επιβλέπων : Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 14η Οκτωβρίου 2020.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Νεκτάριος Κοζύρης Καθηγητής Ε.Μ.Π.	Γεώργιος Γκούμας Επίκουρος Καθηγητής Ε.Μ.Π.	Ιωάννης Κωνσταντίνου Επίκουρος Καθηγητής Π.Θ.

Αθήνα, Οκτώβριος 2020

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

(Υπογραφή)

.....

ΙΩΑΝΝΗΣ Α. ΓΡΗΓΟΡΑΚΟΣ

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

Περίληψη

Ο σκοπός της παρούσας διπλωματικής εργασίας είναι η συγκριτική αξιολόγηση μερικών εκ των πιο διαδεδομένων συστημάτων μη σχεσιακών βάσεων δεδομένων τύπου κλειδιού-τιμής ανοιχτού κώδικα. Ο στόχος της αξιολόγησης είναι η καταγραφή και σύγκριση του τρόπου με τον οποίο τα διαφορετικά συστήματα συμπεριφέρονται ανάλογα με τη διάταξή τους, αν δηλαδή βρίσκονται σε κατάσταση μονού κόμβου η σε σύμπλεγμα, ανάλογα με το φόρτο εργασίας που τους υποβάλλεται προς εκτέλεση, ανάλογα με την παραμετροποίησή τους και ανάλογα με το περιβάλλον που εκτελείται το κάθε σύστημα. Η γενικότερη μεθοδολογία που χρησιμοποιήθηκε είναι η ανάλυση των διαφορετικών αυτών συστημάτων αρχικά σε θεωρητικό επίπεδο, με βάση τις προδιαγραφές και τις λεπτομέρειες της υλοποίησής τους προκειμένου να μπορούν να τεκμηριωθούν επιστημονικά τα ευρήματα της πειραματικής αξιολόγησης, η οποία ακολούθησε. Συγκεκριμένα, κάθε σύστημα αξιολογήθηκε με το εργαλείο YCSB, με τη βοήθεια του οποίου μετρήθηκαν και καταγράφηκαν οι επιδόσεις του, τόσο σε εικονικό περιβάλλον (virtual machines) όσο και σε πραγματικό περιβάλλον (bare metal). Επίσης, μετρήθηκε η χρησιμοποίηση των υπολογιστικών πόρων (επεξεργαστικής ισχύος και κύριας μνήμης). Οι παραπάνω μετρήσεις παρουσιάζονται εκτενώς και σχολιάζονται, με σκοπό την αντικειμενική σύγκριση μεταξύ των συστημάτων και την εξαγωγή συμπερασμάτων σχετικά με τα πλεονεκτήματα και μειονεκτήματα του καθενός από αυτά.

Λέξεις Κλειδιά: <<YCSB, Redis, Aerospike, ArangoDB, βάσεις δεδομένων κλειδιού-τιμής, διεκπεραιωτική ικανότητα, χρόνος καθυστέρησης εκτέλεσης εντολών, εγγραφή, φόρτος εργασίας, ευρετήρια, νήμα εκτέλεσης, πίνακας κατακερματισμού, κόμβος, σύμπλεγμα>>

Abstract

The purpose of this dissertation is the comparative evaluation of some of the most known open source key-value relational databases. The goal of the evaluation is to record and compare the way in which the different systems behave according to their arrangement, ie whether they are in a standalone state or being a node of a cluster, depending on the workload submitted to them, depending on the configuration and depending on the environment in which each system runs. The general methodology used is the analysis of these different systems initially at a theoretical level, based on the specifications and details of their implementation in order to be able to scientifically substantiate the findings of the experimental evaluation that followed. In particular, each system was evaluated with the YCSB tool, with the help of which its performance was measured and recorded, both in a virtual environment (virtual machines) and in a real environment (bare metal machine). Also, the use of computing resources (processing power and main power) was measured. The above measurements are presented in detail and commented, in order to objectively compare the systems and draw conclusions about the advantages and disadvantages of each of them.

Keywords: <<YCSB, Redis, Aerospike, ArangoDB, key-value store database, throughput, latency, record, workload, index, threads, hash table, node, cluster>>

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή Νεκτάριο Κοζύρη, για την εμπιστοσύνη που μου έδειξε με την ανάθεση της συγκεκριμένης διπλωματικής εργασίας καθώς και τον υποψήφιο διδάκτορα Νίκο Χαλβαντζή για την εξαιρετική συνεργασία και την πολύτιμη καθοδήγηση που μου παρείχε κατά την εκπόνηση της παρούσας εργασίας.

Από καρδίας θα ήθελα να ευχαριστήσω τους γονείς μου, Αριστοτέλη και Μαρίνα, για την αγάπη, την αφοσίωση, την υπομονή και τη συμπαράστασή τους σε κάθε βήμα της ζωής μου.

Ευχαριστώ από καρδιάς τις αδερφές μου, Έλενα, Βίκυ και Κατερίνα για τη στήριξη και την αγάπη τους.

Τέλος, ευχαριστώ από καρδιάς τους φίλους μου, για τη συντροφιά, την υποστήριξη και τη συνεργασία τους σε όλα τα στάδια της ζωής μου.

Ιωάννης Α. Γρηγοράκος,
Αθήνα, 14η Οκτωβρίου 2020

Πίνακας περιεχομένων

Πίνακας Εικόνων

Πίνακας Διαγραμμάτων

1. Εισαγωγή

1.1 Γενική παρουσίαση key-value stores

Τα τελευταία χρόνια έχει παρατηρηθεί αυξανόμενος πολλαπλασιασμός στη δημιουργία μη σχεσιακών βάσεων δεδομένων (NoSQL) [1]. Μεταξύ των βασικών πλεονεκτημάτων τους είναι η υπόσχεση ταχύτερης και αποτελεσματικότερης απόδοσης από τα κλασικά Συστήματα Διαχείρισης Βάσεων Δεδομένων (RDBMS) [2]. Οι βάσεις δεδομένων NoSQL είναι επίσης προσαρμοσμένες στον ταχέως αναπτυσσόμενο κόσμο του υπολογιστικού νέφους (cloud computing) [3], επιτρέποντας μαζική κλιμακωσιμότητα «κατ'απαίτηση» (ελαστικότητα) και απλοποιημένη ανάπτυξη εφαρμογών. Ωστόσο, δε δημιουργούνται όλες οι βάσεις δεδομένων NoSQL με σκοπό την απόδοση. Μια κύρια κατηγορία των μη σχεσιακών βάσεων δεδομένων είναι οι βάσεις δεδομένων τύπου κλειδιού-τιμής. Μια βάση δεδομένων κλειδιού-τιμής είναι μια απλή βάση δεδομένων που χρησιμοποιεί έναν πίνακα συσχέτισης (όπως ένας χάρτης ή ένα λεξικό) ως το βασικό μοντέλο δεδομένων όπου κάθε κλειδί σχετίζεται με μία και μόνο μία τιμή σε μια συλλογή[4]. Αυτή η σχέση αναφέρεται ως ζεύγος κλειδιού-τιμής. Σε κάθε ζεύγος κλειδιού-τιμής το κλειδί αντιπροσωπεύεται από μια αυθαίρετη συμβολοσειρά, όπως ένα όνομα αρχείου, URI ή έναν κατακερματισμό. Η τιμή μπορεί να είναι οποιοδήποτε είδος δεδομένων, όπως εικόνα, αρχείο προτιμήσεων χρήστη ή έγγραφο. Η τιμή αποθηκεύεται ως έχει και δεν απαιτεί εκ των προτέρων μοντελοποίηση δεδομένων ή κάποιο ορισμό σχήματος όπως συμβαίνει με τις σχεσιακές βάσεις δεδομένων. Η αποθήκευση της τιμής ως έχει αφαιρεί την ανάγκη ευρετηριοποίησης των δεδομένων για τη βελτίωση της απόδοσης. Ωστόσο, δεν υπάρχει δυνατότητα φιλτραρίσματος ή να ελέγχου στο τι επιστρέφεται από ένα αίτημα με βάση την τιμή, επειδή η τιμή είναι αδιάφανη. Συνήθως, οι βάσεις δεδομένων κλειδιού-τιμής δεν υποστηρίζουν κάποια γλώσσα ερωτημάτων. Παρέχουν έναν τρόπο αποθήκευσης, ανάκτησης και ενημέρωσης δεδομένων χρησιμοποιώντας απλές εντολές λήψης, τοποθέτησης και διαγραφής, όπου η διαδρομή για την ανάκτηση δεδομένων είναι ένα άμεσο αίτημα προς το αντικείμενο στη μνήμη ή στο δίσκο. Η απλότητα αυτού του μοντέλου καθιστά μια τέτοια βάση γρήγορη, εύκολη στη χρήση, επεκτάσιμη, φορητή και ευέλικτη.

1.2 Αντικείμενο διπλωματικής

Το αντικείμενο αυτής της διπλωματικής είναι η σύγκριση τριών δημοφιλών βάσεων δεδομένων τύπου κλειδιού-τιμής με βάση τη δημοτικότητα τους [5]. Οι βάσεις αυτές είναι η Redis [6], η Aerospike [7] και η ArangoDB [8]. Τα συστήματα αυτά θα τα αξιολογήσουμε τόσο σε πραγματικό περιβάλλον με πραγματικά (bare metal) μηχανήματα, όσο και σε εικονικό περιβάλλον χρησιμοποιώντας διάφορες προσομοιώσεις μηχανημάτων, ονόματι Virtual Machines (VMs) [9]. Η αξιολόγηση αυτή θα είναι τόσο σε θεωρητικό επίπεδο, όσο και σε πρακτικό επίπεδο μέσω διαφόρων τεστ αναδεικνύοντας τις δυνατότητες κάθε βάσης. Πολλές φορές όμως οι συγκρίσεις μεταξύ βάσεων δεδομένων είναι αρκετά δύσκολη διαδικασία και συνοδεύεται από πολλά λάθη, καθώς κάθε σύστημα διαφέρει στην υλοποίησή του οπότε τα διάφορα τεστ δεν εκτελούνται κάτω από αντίστοιχες συνθήκες και δεν υπάρχει αντικειμενικότητα. Το πρόβλημα αυτό προσπαθεί να λύσει το εργαλείο YCSB [10] που έχει αναπτυχθεί από την εταιρία Yahoo!. Σε αυτή την εργασία θα χρησιμοποιήσουμε το εργαλείο αυτό ως την πλατφόρμα για τα τεστ που εκτελέσουμε πάνω στις παραπάνω βάσεις δεδομένων.

1.2.1 Συνεισφορά

Η συνεισφορά της διπλωματικής συνοψίζεται ως εξής:

1. Μελετήσαμε διεξοδικά τα συστήματα Redis, Aerospike, ArangoDB και YCSB
2. Αξιολογήσαμε εκτενώς την επίδοση των τριών βάσεων δεδομένων σε πραγματικό και σε εικονικό περιβάλλον
3. Τεκμηριώσαμε με επιστημονικό τρόπο κάθε αποτέλεσμα το οποίο συλλέξαμε
4. Ανακαλύψαμε ελλιπείς υλοποιήσεις και προβλήματα στο εργαλείο ανοιχτού κώδικα YCSB
5. Προτείναμε λύσεις για την κάλυψη των παραπάνω ελλείψεων και προβλημάτων

1.3 Οργάνωση Κειμένου

Στο Κεφάλαιο 2 παρουσιάζουμε εργασίες σχετικές με το αντικείμενο της διπλωματικής.

Στο Κεφάλαιο 3 αναλύουμε εκτενώς σε θεωρητικό επίπεδο τις τρεις βάσεις δεδομένων που θα αξιολογήσουμε.

Στο Κεφάλαιο 4 αναλύουμε εκτενώς το εργαλείο YCSB και περιγράφουμε τη μεθοδολογία που θα χρησιμοποιήσουμε για τη διεκπεραίωση αυτών των τεστ.

Στο Κεφάλαιο 5 παρουσιάζουμε τον τρόπο με τον οποίο εγκαθιστούμε και παραμετροποιούμε την κάθε βάση και παρουσιάζουμε και αναλύουμε τα αποτελέσματα που συλλέξαμε από κάθε φόρτο εργασίας.

Στο Κεφάλαιο 6 παρουσιάζουμε τα συμπεράσματά μας και το πως η εργασία θα μπορούσε να επεκταθεί μελλοντικά.

Στο Κεφάλαιο 7 παρουσιάζουμε όλες τις πηγές που χρησιμοποιήσαμε για την εκπόνηση αυτής της εργασίας.

2. Σχετικές Εργασίες

Στη βιβλιογραφία εντοπίζουμε πολλές εργασίες οι οποίες είχαν αρκετές ομοιότητες με τη δική μας, αλλά και πολλές διαφορές. Στο [11] συγκρίνονται 6 συστήματα βάσεων δεδομένων διαφόρου τύπου. Αυτές ήταν η Redis και το Project Voldemort[12] από τύπου μη σχεσιακών βάσεων δεδομένων κλειδιού-τιμής, το MySQL Cluster[13] και η VoltDB[14] από τύπου σχεσιακών βάσεων δεδομένων με καλή ιδιότητα κλιμακωσιμότητας και η HBase[15] και η Cassandra[16] από τύπου επεκτάσιμων εγγράφων βάσεων δεδομένων. Το εργαλείο που χρησιμοποιείται σε αυτές τις βάσεις δεδομένων είναι το YCSB. Αυτή η εργασία αξιολογεί τις βάσεις σε ένα πραγματικό περιβάλλον με πραγματικά μηχανήματα. Η βασική διαφορά των εργασιών αυτών είναι ότι εμείς συγκρίνουμε στην εργασία μας καθαρά μη σχεσιακές βάσεις δεδομένων τύπου κλειδιού-τιμής. Σε αυτή την εργασία οι κόμβοι που χρησιμοποιούνται σε κάθε πείραμα είναι περισσότεροι σε σύγκριση με τη δική μας. Η δική μας εργασία βέβαια καλύπτει και περιπτώσεις με διαφορετικό αριθμό παράλληλων νημάτων εκτέλεσης του πελάτη YCSB, κάτι το οποίο δε συμβαίνει με την προηγούμενη εργασία. Οι εργασίες αυτές επίσης εξετάζουν την διεκπεραιωτική ικανότητα και την καθυστέρηση του χρόνου εκτέλεσης των λειτουργιών. Η προαναφερθείσα εργασία επίσης μετράει και τη χρησιμοποίηση του σκληρού δίσκου, ενώ η δική μας εργασία μετράει τη ποσοστιαία χρησιμοποίηση της μνήμης ram και τη επεξεργαστικής ισχύος. Επιπρόσθετα η δική μας εργασία εξετάζει τις βάσεις και σε εικονικό περιβάλλον. Η επόμενη εργασία [17] δεν έχει τόσα κοινά στοιχεία όσα και η προηγούμενη. Οι διαφορές είναι ότι αυτή η έρευνα παρουσιάζει μια καινούργια μη σχεσιακή βάση δεδομένων τύπου κλειδιού-τιμής. Στο 5ο κεφάλαιο την συγκρίνει με άλλες δύο βάσεις αντίστοιχου τύπου. Την Memcached [18] και την Redis. Η εργασία αυτή γίνεται σε πραγματικό περιβάλλον με πραγματικά μηχανήματα. Οι μετρήσεις γίνονται χρησιμοποιώντας όλους τους ήδη υπάρχοντες φόρτους εργασίας του YCSB. Επίσης μόνο ο αριθμό των κόμβων της Redis αλλάζει προκειμένου να επιτευχθεί η αντιστοίχιση της με υπόλοιπες δύο βάσεις στο θέμα της πολυνηματικής εκτέλεσης. Επίσης σε αυτή την εργασία χρησιμοποιείται διαφορετική υλοποίηση της κάθε βάσης. Για παράδειγμα γίνεται αξιολόγηση της Redis χρησιμοποιώντας τον τύπο Συμβολοσειράς (String) και γίνεται και αξιολόγηση της Redis χρησιμοποιώντας τον αντίστοιχο τύπο ZSet. Η συγκεκριμένη εργασία όμως δεν παρέχει αποτελέσματα όσον αφορά τη χρησιμοποίηση της ram και τη χρησιμοποίηση της επεξεργαστικής ισχύος. Μια ακόμα εργασία η οποία μας έδωσε αρκετά στοιχεία ήταν η [19]. Σε αυτή την εργασία συγκρίνονται 3 βάσεις δεδομένων, η Redis, η Aerospike και η Memcahced. Η συγκεκριμένη εργασία προσμετρά την διεκπεραιωτική ικανότητα και το χρόνο καθυστέρησης εκτέλεσης των εντολών σε λειτουργία μονού κόμβου και σε λειτουργία συμπλέγματος. Τέλος αξιοποιήσαμε τις πληροφορίες που μας προσέφερε η [10]. Σε αυτή την εργασία παρουσιάζεται το εργαλείο YCSB και συγκρίνονται οι βάσεις Cassandra, HBase, MySQL και η PNUTS[20].

3. Παρουσίαση συστημάτων - Θεωρητικό Υπόβαθρο

Στη συγκεκριμένη εργασία αποφασίσαμε να αναλύσουμε εκτενώς τρεις βάσεις δεδομένων και να κάνουμε τα πειράματα μας σε αυτές. Οι βάσεις δεδομένων που καταλήξαμε είναι η Redis, η Aerospike και η ArangoDB. Επειδή όμως θέλουμε να κατανοήσουμε ακριβώς το κάθε αποτέλεσμα σε κάθε φόρτο εργασίας που θα τρέξουμε στην εκάστοτε βάση δεδομένων κρίνεται σκόπιμο να αναλύσουμε θεωρητικά τα τρία αυτά μοντέλα. Αυτή η αναύση θα μας δώσει την επίγνωση ώστε να μπορούμε να τεκμηριώσουμε ορθά κάθε αποτελεσμα.

3.1 Redis

Η Redis είναι ένα πρότζεκτ αποθήκευσης δομών δεδομένων στη μνήμη (ram) το οποίο υλοποιεί μια κατανεμημένη βάση δεδομένων κλειδιού-τιμής με προαιρετική αποθήκευση στη μόνιμη μνήμη του υπολογιστή (σκληρός δίσκος). Το όνομα Redis προέρχεται από τα αρχικά REmote DIctionary Server το οποίο μεταφράζεται ως "Απομακρυσμένος Διακομιστής Λεξικού". Ως λεξικό αναφέρεται μια δομή δεδομένων η οποία αποθηκεύει τα στοιχεία της ως ένα μη οργανωμένο σετ από ζευγάρια κλειδιού-τιμής (key-value pairs). Το πρότζεκτ αυτό ξεκίνησε όταν ο Salvatore Sanfilippo, ο κύριος προγραμματιστής της Redis, ο οποίος είναι γνωστός με το παρατσούκλι antirez στο διαδίκτυο, προσπαθούσε να βελτιώσει την κλιμακωσιμότητα της ιταλικής του εφαρμογής (startup), αναπτύσσοντας έναν αναλυτή καταγραφής ιστού σε πραγματικό χρόνο. Αφού αντιμετώπισε σημαντικά προβλήματα στην κλιμάκωση ορισμένων τύπων φόρτου εργασίας (workloads) με τη χρήση παραδοσιακών συστημάτων βάσεων δεδομένων, ο Sanfilippo άρχισε να δημιουργεί το πρωτότυπο της αρχικής έκδοσης της Redis, η οποία αναπτύχθηκε αρχικά στη γλώσσα προγραμματισμού Tcl. Αργότερα ο Sanfilippo μετέφρασε αυτό το πρωτότυπο στη γλώσσα C και δημιούργησε τον πρώτο σύνθετο τύπο δεδομένων, τη λίστα. Μετά από λίγες εβδομάδες επιτυχούς χρήσης του έργου του, ο Sanfilippo αποφάσισε να το δημοσιεύσει, ανακοινώνοντας το έργο του στο Hacker News. Η Redis άρχισε να προσελκύει ενδιαφέρον, ειδικά στην κοινότητα της προγραμματιστικής γλώσσας Ruby, με το Github και το Instagram να είναι από τις πρώτες εταιρίες που την υιοθέτησαν. Τον Ιούνιο του 2015 η Redis χρηματοδοτήθηκε από την εταιρία Redis Labs όπου και μέχρι σήμερα αναλαμβάνουν για την περαιτέρω ανάπτυξη της.

Η Redis είναι ένα πρότζεκτ ανοιχτού κώδικα με άδεια BSD και πέρα από βάση δεδομένων μπορεί να χρησιμοποιηθεί και ως προσωρινή μνήμη (cache) και μεσολαβητής μηνυμάτων (message broker). Υποστηρίζει πολλούς τύπους δεδομένων όπως συμβολοσειρές (strings), κατακερματισμούς (hashes), λίστες (lists), σύνολα (sets), ταξινομημένα σύνολα με ερωτήματα εύρους (sorted sets with range), σύνολα από εντολές πάνω σε μπιτ τα οποίο ονομάζονται bitmaps, τα hyperloglogs, γεωχωρικά ευρετήρια με ερωτήματα εμβέλειας και ροών (geospatial indexes with radius queries and

streams). Επίσης η Redis διαθέτει ενσωματωμένη αντιγραφή, δυνατότητα γραφής μικρών προγραμμάτων (scripts) στη γλώσσα προγραμματισμού Lua, χρησιμοποιεί μιά τεχνική για να αποδεσμεύει δεδομένα που δεν έχουν χρησιμοποιηθεί πρόσφατα από τη μνήμη (LRU eviction), δοσοληψίες (transactions), διαφορετικά επίπεδα αποθήκευσης δεδομένων στο δίσκο και παρέχει υψηλή διαθεσιμότητα μέσω του Redis Sentinel και του αυτόματου διαμερισμού δεδομένων με το Redis Cluster. Επειδή η Redis είναι μια εφαρμογή η οποία χρησιμοποιεί μόνο ένα νήμα εκτέλεσης (single threaded application), όλοι οι παραπάνω τύπο δεδομένων δέχονται ατομικές εντολές (atomic operations). Προκειμένου να πετύχει τη μεγάλη της απόδοση, η Redis λειτουργεί με ένα σύνολο δεδομένων τα οποία διατηρεί στη μνήμη της. Ανάλογα με την περίπτωση μπορεί να αποθηκεύσει τα δεδομένα στο δίσκο είτε αποθηκεύοντας τα στο δίσκο κάθε συγκεκριμένη χρονική περίοδο, είτε καταγράφοντας κάθε εντολή που έχει εκτελεστεί σε ένα αρχείο. Αυτή η διαδικασία αποθήκευσης μπορεί εναλλακτικά να απενεργοποιηθεί. Επιπρόσθετα η Redis υποστηρίζει ασύγχρονη αντιγραφή σε κόμβους τύπου κύριου-υποτελούς (master-slave) με σχεδόν ασήμαντη προσπάθεια εγκατάστασης, με πολύ γρήγορο αμπλοκάριστο πρώτο συγχρονισμό και αυτόματη επανασύνδεση με μερικό επανασυγχρονισμό σε περίπτωση προβλήματος στο δίκτυο.

Παρακάτω θα αναλύσουμε τους βασικούς τύπους δεδομένων που η Redis προσφέρει και πως υλοποιούνται πραγματικά .

- Συμβολοσειρές (Strings) [21]: Οι συμβολοσειρές είναι απλές συμβολοσειρές έτσι όπως υπάρχουν στη γλώσσα προγραμματισμού C. Μια συμβολοσειρά μπορεί να εμπεριέχει μια ακολουθία γραμμάτων, ακέραιους αριθμούς ακόμα και δεκαδικούς αριθμούς. Μια συμβολοσειρά δεν μπορεί να ξεπεράσει το μέγεθος των 512 megabyte (MB) κειμένου ή δυαδικών δεδομένων. Είναι η βάση όλων των υπολοίπων πιο σύνθετων δομών δεδομένων.
- Λίστες (Lists): Οι λίστες έχουν δημιουργηθεί χρησιμοποιώντας συνδεδεμένες λίστες ή τα ziplists, οπότε οι εισαγωγές και οι διαγραφές των στοιχείων τους από την αρχή ή από το τέλος έχουν χρόνο εκτέλεσης O(1). Ένα ziplist είναι ουσιαστικά μια διπλά συνδεδεμένη λίστα σχεδιασμένη να καταναλώνει όσο το δυνατόν λιγότερη μνήμη. Σε ένα ziplist οι ακέραιοι αριθμοί αποθηκεύονται ως πραγματικοί ακέραιοι αριθμοί και όχι ως μια ακολουθία χαρακτήρων όπως συνηθίζεται σε άλλες περιπτώσεις. Ένα ziplist αποθηκεύει δεδομένα με την εξής σειρά. Μήκος, μήκος, συμβολοσειρά. Στο πρώτο μήκος αποθηκεύει το μέγεθος του προηγούμενου ziplist που εισάχθηκε (για πιο γρήγορη αναζήτηση και στην αρχή και στο τέλος), το δεύτερο μήκος αποθηκεύει το μέγεθος του ίδιου του ziplist και στη συμβολοσειρά αποθηκεύει τα δεδομένα. Ενώ ένα ziplist έχει σχεδιαστεί να είναι όσο πιο βελτιωμένο απο άποψη κατανάλωσης μνήμης, η αναζήτηση δεν πραγματοποιείται σε σταθερό χρόνο O(1). Η λίστα μπορεί να κωδικοποιηθεί και να βελτιστοποιηθεί, όσον αφορά την δέσμευση μνήμης της, αν περιέχει λιγότερα στοιχεία από μια παράμετρο η οποία ονομάζεται list-max-ziplist-entries και κάθε στοιχείο της είναι μικρότερο σε μέγεθος byte απο την παράμετρο list-max-ziplist-value. Σε αυτή την περίπτωση αντι για διπλά συνδεδεμένη λίστα χρησιμοποιείται ένα ziplist. Ο μεγαλύτερος αριθμός στοιχείων που μπορούν να αποθηκευτούν σε μια λίστα είναι ο 2^32 -1 , που σημαίνει ότι μπορούν να αποθηκευτούν πάνω από 4 δισεκατομμύρια στοιχεία ανά λίστα.

- Κατακερματισμοί (Hashes)[22]: Τα Hashes είναι ουσιαστικά μιά απεικόνιση μιας συμβολοσειράς σε μία άλλη συμβολοσειρα. Μπορούν να υλοποιηθούν με μια δομή δεδομένων που ονομάζεται ziplist ή με έναν πίνακα κατακερματισμού (hash table). Ενώ ένα ziplist έχει σχεδιαστεί να είναι όσο πιο βελτιωμένο απο άποψη κατανάλωσης μνήμης, η αναζήτηση δεν πραγματοποιείται σε σταθερό χρόνο O(1). Από την άλλη μεριά αν ένα Hash υλοποιηθεί με έναν πίνακα κατακερματισμού, τότε η αναζήτηση στοιχείου επιτυγχάνεται σε σταθερό χρόνο O(1), αλλά αυτή η υλοποίηση δεν είναι βέλτιστη από άποψη κατανάλωσης μνήμης. Τα Hashes έχουν σχεδιαστεί ώστε να είναι πολύ αποδοτικά σε κατανάλωση μνήμης, αλλά και σε χρόνο αναζήτησης. Προκειμένου ένα Hash να υλοποιηθεί ως ziplist πρέπει να ικανοποιηθούν δύο συνθήκες. Ένα hash πρέπει να έχει λιγότερα πεδία από τη μεταβλητές hash-max-ziplist-entries. Η προκαθορισμένη τιμή αυτής της τιμής είναι 512 mb. Επίσης κανένα πεδίο δεν πρέπει να είναι μεγαλύτερο από την τιμή hash-max-ziplist-value η οποία έχει σαν προκαθορισμένη τιμή τα 64 mb. Ο μεγαλύτερος αριθμός στοιχείων που μπορούν να αποθηκευτούν σε ένα Hash είναι ο 2^32 -1 , που σημαίνει ότι μπορούν να αποθηκευτούν πάνω από 4 δισεκατομμύρια στοιχεία ανά λίστα.
- Σύνολα (Sets): Τα σύνολα είναι ανοργάνωτες συλλογές από διακριτές συμβολοσειρές. Αυτό σημαίνει ότι δεν μπορούν να αποθηκευτούν δύο ίδιες εγγραφές. Το σύνολο υλοποιείται από έναν πίνακα κατακερματισμού, το οποίο σημαίνει ότι μερικές διαδικασίες είναι βελτιστοποιημένες, όπως η εισαγωγή, η αναζήτησή και η διαγραφή στοιχείων οι οποίες εκτελούνται σε σταθερό χρόνο O(1). Η μνήμη που καταναλώνει ένα σύνολο μπορεί να ελαχιστοποιηθεί εάν όλα τα μέλη του συνόλου είναι ακέραιοι αριθμοί και εφόσον ο αριθμός των εγγραφών σε ένα σύνολο δεν ξεπερνάει την παράμετρο set-max-intset-entries. Ο μεγαλύτερος αριθμός στοιχείων που μπορούν να αποθηκευτούν σε ένα Σύνολο είναι ο 2^32 -1 , που σημαίνει ότι μπορούν να αποθηκευτούν πάνω από 4 δισεκατομμύρια στοιχεία ανά λίστα.
- Ταξινομημένα Σύνολα (Sorted Sets): Τα ταξινομημένα σύνολα είναι παρεμφερή με τα σύνολα που περιγράψαμε από πάνω. Η διαφορά έγκειται στο ότι κάθε εγγραφή συνοδεύεται και από ένα σκορ. Με άλλα λόγια ένα ταξινομημένο σύνολο είναι μια συλλογή από μη επαναλαμβανόμενες συμβολοσειρές ταξινομημένες με κριτήριο το σκορ τους. Είναι πιθανό να έχουμε στοιχεία τα οποία έχουν το ίδιο σκορ. Σε αυτή την περίπτωση τα στοιχεία ταξινομούνται με αλφαβητική σειρά. Οι εντολές με ταξινομημένα σύνολα είναι αρκετά γρήγορες, αλλά όχι τόσο γρήγορες όσο των απλών συνόλων καθώς τα σκορ συνεχώς συγκρίνονται μεταξύ τους. Η εισαγωγή, αναζήτησή και η διαγραφή ενός στοιχείου σε ένα ταξινομημένο σύνολο απαιτεί λογαριθμικό χρόνο εκτέλεση O(log(N)), όπου το N είναι ο αριθμός στοιχείων που υπάρχουν σε ένα ταξινομημένο σύνολο. Τα ταξινομημένα σύνολα υλοποιούνται είτε ως μια δομή δεδομένων γνωστή ως skiplist είτε ως ziplist τα οποία περιγράψαμε προηγουμένως.
- Bitmaps: Τα bitmaps δεν είναι μια πραγματική δομή δεδομένων της Redis. Ουσιαστικά ένα bitmap υλοποιείται με μια συμβολοσειρά. Η διαφορά από μια απλή συμβολοσειρά είναι ότι τα bitmaps περιέχουν μόνο ακολουθίες από τα μπιτ 0 και 1. Οι λειτουργίες τους χωρίζονται σε δύο ομάδες. Τις λειτουργίες ενιαίου μπιτ σταθερού χρόνου, όπως ρύθμιση μπιτ σε 1 ή 0 ή λήψη της τιμής τους, και λειτουργίες σε ομάδες απο μπιτς. Ένα από τα μεγαλύτερα πλεονεκτήματα των bitmaps είναι

ότι συχνά παρέχουν εξαιρετική εξοικονόμηση χώρου κατά την αποθήκευση πληροφοριών. Τα bitmaps είναι πολύ αποδοτικά στην αναζήτηση και δεδομένου ότι το μέγιστο μήκος των συμβολοσειρών είναι 512 MB τότε σε μια συμβολοσειρά μπορούμε να έχουμε 2^32 μπιτς.

- HyperLogLogs: Τα HyperLogLogs δεν είναι μια πραγματική δομή δεδομένων της Redis. Ουσιαστικά είναι ένας αλγόριθμος που χρησιμοποιεί την τυχαιότητα για να παρέχει μια αρκετά καλή εκτίμηση του αριθμού των μοναδικών στοιχείων που υπάρχουν σε ένα σύνολο, που αλλιώς ονομάζεται και καρδιότητα (cardinality). Ο αλγόριθμος αυτός εκτελείται σε σταθερό χρόνο O(1) και χρησιμοποιεί ελάχιστη μνήμη, περίπου 12 KB ανά κλειδί. Επειδή ο αλγόριθμος αυτός χρησιμοποιεί πιθανότητες, δεν είναι 100% εύστοχος. Η υλοποίηση του στη Redis έχει ένα στάνταρ ποσοστό λάθους 0.81%.

Αναλύοντας λοιπόν όλους τους βασικούς τύπους δεδομένων που χρησιμοποιεί η Redis θα προχωρήσουμε στην ανάλυση του ευρετηριασμού (indexing) που χρησιμοποιεί καθώς θα μας βοηθήσει στο να κατανοήσουμε τη συμπεριφορά της στα διάφορα τεστ που θα της κάνουμε αργότερα. Ο πιο ευνόητος τρόπος να παρουσιάσουμε ένα ευρετήριο στη Redis είναι σαν ένα μεγάλο πίνακα κατακερματισμού όπου όλα τα κλειδιά είναι ουσιαστικα ακολουθίες χαρακτήρων. Κάθε κελί περιέχει συνδεδεμένες λίστες οι οποίες αποτελούνται από 3 δείκτες.

Οι συγκρούσεις (collisions) που συμβαίνουν σε αυτό τον πίνακα κατακερματισμού, όταν έχουμε εισαγωγή κλειδιού του οποίου η τιμή κατακερματισμού υπάρχει ήδη στον πίνακα, αντιμετωπίζονται με τη τεχνική της ξεχωριστής αλυσίδας (separate chaining).[23] Σε αντίθεση όμως με τις συμβατικές υλοποιήσεις της τεχνικής της ξεχωριστής αλυσίδας, οι λίστες είναι αντεστραμένες. Δηλαδή ο πρώτος δείκτης δείχνει στην επόμενη εγγραφή ή στον πίνακα κατακερματισμού, ο δεύτερος δείκτης δείχνει στο κλειδί και ο τρίτος δείκτης δείχνει στη τιμή αυτού του κλειδιού. Στην τεχνική αυτή, όταν υπάρχει μια σύγκρουση ανάμεσα στα κλειδιά τότε στη θέση που ανήκουν αυτά τα δύο κλειδιά, δημιουργείται μια καινούργια λίστα. Ο πρώτος δείκτης της προηγούμενης εγγραφής δείχνει πλέον στην καινούργια εγγραφή (και όχι στον πίνακα όπως έδειχνε πρωτίστως) και η καινούργια λίστα γεμίζεται με τον δεύτερό της δείκτη να δείχνει στο καινούργιο κλειδί που εισάγεται, με τον τρίτο δείκτη να δείχνει στην τιμή αυτού του κλειδιού και τον πρώτο δείκτη να δείχνει στον πίνακα .

Το μέγεθος αυτού του πίνακα επιλέγεται ως η κοντινότερη τιμή, του διπλασιασμένου αριθμού καταχωρήσεων που πρέπει να περιέχει ο πίνακας κατακερματισμού, στη δύναμη του 2. Αυτό σημαίνει ότι ο πραγματικός συντελεστής φορτίου κυμαίνεται από 0,5 έως 1,0 με μέσο όρο 0,75. Η αύξηση μεγέθους του πίνακα κατακερματισμού ενεργοποιείται, όταν ο αριθμός των καταχωρήσεων του πίνακα πλησιάζει το αρχικό μέγεθός του. Αυτό μπορεί να απενεργοποιηθεί και πλέον η αλλαγή μεγέθους να ενεργοποιείται μόνο όταν υπάρχουν 5 φορές περισσότερες καταχωρήσεις από το αρχικό μέγεθος του πίνακα. Ο υπολογισμός της νέας συνάρτησης κατακερματισμού εφαρμόζεται με τρόπο ο οποίος δε μπλοκάρει τις άλλες διαδικασίες και όταν χρειαστεί, δημιουργείται ένας νέος πίνακας κατακερματισμού και εκτελούνται οι εξής ενέργειες:

- Σε κάθε ενέργεια του πίνακα κατακερματισμού, συμπεριλαμβανομένου και των εντολών μόνο ανάγνωσης (GET και EXISTS), μια “αλυσίδα” μετακινείται από τον παλιό πίνακα στον

καινούργιο. Με τον όρο “αλυσίδα” εννοούμε μια συνδεδεμένη λίστα η οποία κατοικεί σε ένα κελί του πίνακα. Όταν αυτή η αλυσίδα μετακινείται, η τιμή κατακερματισμού της επαναυπολογίζεται για τον καινούργιο πίνακα και έπειτα τοποθετείται στην κατάλληλη θέση.

- Όταν όλες οι αλυσίδες μετακινηθούν από τον παλιό πίνακα στον νέο, ο παλιός πίνακας διαγράφεται και ο νέος πίνακας κατακερματισμού επιστρέφει στον “κανονικό" του αλγόριθμο.
- Κατά τη διάρκεια της παραπάνω διαδικασίας, η αναζήτηση κλειδιού πραγματοποιείται πρώτα στον παλιό πίνακα και εάν το κλειδί απουσιάζει στον παλιό πίνακα, γίνεται αναζήτηση στον νέο πίνακα.
- Οι νέες εγγραφές εισάγονται στον καινούργιο πίνακα κατακερματισμού.

Η Redis όμως δεν είναι απλά μια βάση δεδομένων τύπου κλειδιού-τιμής καθώς οι τιμές μπορεί να είναι πολύπλοκες δομές δεδομένων. Χρησιμοποιώντας αυτές τις πολύπλοκες δομές δεδομένων η Redis μπορεί να δημιουργήσει δευτερεύον ευρετήρια (secondary indexing) με τα παρακάτω:

- Ταξινομημένα Σύνολα ως Ευρετήρια (Sorted Sets as Indexes): Το απλούστερο δευτερεύον ευρετήριο που μπορεί να δημιουργηθεί είναι χρησιμοποιώντας τον τύπο δεδομένων ταξινομημένου συνόλου. Δεδομένου ότι το σκορ είναι δεκαδικός αριθμός διπλής ακρίβειας, οι δείκτες που μπορούν να δημιουργηθούν είναι περιορισμένοι σε έναν αριθμό εντός δεδομένου εύρους ο οποίος είναι από -2^53 εώς και 2^53.
- Λεξικογραφικά Ευρετήρια (Lexicographical Indexes): Τα Ταξινομημένα Σύνολα έχουν μια ενδιαφέρουσα ιδιότητα. Όταν προστίθενται στοιχεία με το ίδιο σκορ, τότε ταξινομούνται λεξικογραφικά, συγκρίνοντας τις συμβολοσειρές ως δυαδικά δεδομένα με τη συνάρτηση memcmp(). Αυτή η δυνατότητα της Redis είναι βασικά ισοδύναμη με μια δομή δεδομένων που ονομάζεται b-trees που χρησιμοποιείται συχνά για την δημιουργία δεικτών στις πιο παραδοσιακές βάσεις δεδομένων.
- Γεωχωρικά Ευρετήρια (Geospatial indexes): Η Redis έχει πολλές εντολές που σχετίζονται με τη γεωγραφική χωροθέτηση (εντολές GEO), αλλά σε αντίθεση με άλλες εντολές, αυτές οι εντολές δεν διαθέτουν τον δικό τους τύπο δεδομένων. Αυτές οι εντολές στην πραγματικότητα εφαρμόζονται πάνω σε ένα Ταξινομημένο Σύνολο. Αυτό επιτυγχάνεται κωδικοποιώντας το γεωγραφικό πλάτος και μήκος στο σκορ του ταξινομημένου συνόλου χρησιμοποιώντας τον αλγόριθμο geohash.
- Πολυδιάστατα Ευρετήρια (Multidimensional indexes): Ένας πιο σύνθετος τύπος ευρετηρίου είναι ένα ευρετήριο το οποίο επιτρέπει την εκτέλεση ερωτημάτων όπου δύο ή περισσότερες μεταβλητές υποβάλλονται ταυτόχρονα σε ερώτηση για συγκεκριμένα εύρη. Αυτό το ερώτημα μπορεί να εκτελεστεί με δείκτη πολλαπλών στηλών, αλλά αυτό απαιτεί να επιλεχθεί η πρώτη μεταβλητή και, στη συνέχεια να αναζητηθεί η δεύτερη, πράγμα που σημαίνει ότι ο χρόνος εκτέλεσης ενδεχομένως να είναι μεγαλύτερος για τον πραγματικό σκοπό που χρειάζεται. Για την εκτέλεση αυτών των ερωτημάτων η Redis χρησιμοποιεί πολυδιάστατα δέντρα, όπως k-d trees ή r-trees.

Επειδή στα πλαίσια της αξιολόγησης μας θα χρησιμοποιήσουμε τη λειτουργία συμπλέγματος (Cluster Mode) που προσφέρει η Redis, κρίνεται σκόπιμο να αναλυθεί. Όλα τα παρακάτω ισχύουν από την έκδοση 3.0 και έπειτα. Ουσιαστικά η λειτουργία αυτή επιτρέπει στη Redis να έχει παραπάνω από ένα διακομιστή-κόμβο (node) ταυτόχρονα για την ίδια διεργασία. Το Redis Cluster είναι μια κατανεμημένη υλοποίηση της Redis με τους ακόλουθους στόχους, με τη παρακάτω σειρά σπουδαιότητας στο σχεδιασμό:

- Υψηλή απόδοση και γραμμική επεκτασιμότητα έως 1000 κόμβους. Δεν υπάρχουν διακομιστές μεσολάβησης, χρησιμοποιείται ασύγχρονη αναπαραγωγή και δεν εκτελούνται εργασίες συγχώνευσης σε τιμές.
- Αποδεκτός βαθμός ασφάλειας εγγραφής: το σύστημα προσπαθεί (με τον καλύτερο δυνατό τρόπο) να διατηρήσει όλες τις εγγραφές που προέρχονται από πελάτες που συνδέονται με την πλειονότητα των κύριων κόμβων. Συνήθως υπάρχουν μικρά παράθυρα όπου μπορεί να χαθούν αναγνωρισμένες εγγραφές. Τα παράθυρα αυτά είναι μεγαλύτερα όταν οι πελάτες βρίσκονται σε διαμερισμό μειονότητας.
- Διαθεσιμότητα: Το Redis Cluster είναι σε θέση να επιβιώσει σε διαμερισμούς όπου η πλειονότητα των κύριων κόμβων είναι προσβάσιμη και υπάρχει τουλάχιστον ένας προσβάσιμος υποτελής κόμβος για κάθε κύριο κόμβο που δεν είναι πλέον προσβάσιμος. Επιπλέον, χρησιμοποιώντας τη μετεγκατάσταση αντιγράφων, δηλαδή οι κύριοι κόμβοι που δεν έχουν πλέον κανένα υποτελή κόμβο θα λάβουν έναν από έναν άλλο κύριο κόμβο που καλύπτεται από πολλούς υποτελείς κόμβους.

Το Redis Cluster εφαρμόζει όλες τις εντολές μονού κλειδιού που είναι διαθέσιμες στη μη κατανεμημένη έκδοση της Redis. Οι εντολές που εκτελούν πολύπλοκες λειτουργίες πολλαπλών κλειδιών, όπως ενώσεις Συνόλων ή διασταυρώσεις Συνόλων, εφαρμόζονται εφόσον όλα τα κλειδιά είναι κατακερματισμένα στην ίδια θυρίδα (slot). Το Redis Cluster εφαρμόζει μια έννοια που ονομάζεται “hash tags” που μπορεί να χρησιμοποιηθεί για να αναγκάσει ορισμένα κλειδιά να αποθηκευτούν στην ίδια θυρίδα κατακερματισμού. Ωστόσο, κατά το χειροκίνητο θρυμματισμό, οι λειτουργίες πολλαπλών κλειδιών ενδέχεται να μην είναι διαθέσιμες για κάποιο χρονικό διάστημα, ενώ οι λειτουργίες με ένα μονό κλειδί είναι πάντα διαθέσιμες.

Στο Redis Cluster οι κόμβοι είναι υπεύθυνοι για τη διαφύλαξη των δεδομένων και τη λήψη της κατάστασης του συμπλέγματος, συμπεριλαμβανομένου και της αντιστοίχισης των κλειδιών στους σωστούς κόμβους. Αυτό επιτυγχάνεται χρησιμοποιώντας ένα μοντέλο κύριου-υποτελή κόμβου (master-slave), όπου ένας κύριος κόμβος έχει τη δυνατότητα να επικοινωνήσει με N υποτελείς κόμβους. Ο κύριος κόμβος είναι η πηγή αλήθεια για τους υποτελείς κόμβους και μόλις γίνει μια αλλαγή σε αυτόν, τότε ενημερώνει και τους υποτελείς κόμβους. Οι κόμβοι του συμπλέγματος είναι επίσης σε θέση να ανακαλύπτουν αυτόματα άλλους κόμβους, να ανιχνεύουν μη λειτουργικούς κόμβους και να προωθούν τους υποτελείς κόμβους σε κύριους κόμβους όταν χρειάζεται, προκειμένου να συνεχίσουν να λειτουργούν όταν παρουσιαστεί μια αποτυχία. Για την εκτέλεση των καθηκόντων τους όλοι οι κόμβοι συμπλέγματος συνδέονται χρησιμοποιώντας ένα

δίαυλο πρωτοκόλλου TCP και ένα δυαδικό πρωτόκολλο, που ονομάζεται “Redis Cluster Bus”. Κάθε κόμβος συνδέεται με κάθε άλλο κόμβο του συμπλέγματος χρησιμοποιώντας το δίαυλο αυτό. Οι κόμβοι χρησιμοποιούν ένα πρωτόκολλο κουτσομπολιού (gossip protocol) για τη διάδοση πληροφοριών στο σύμπλεγμα, προκειμένου να ανακαλύψουν νέους κόμβους, να στείλουν πακέτα ping για να βεβαιωθούν ότι όλοι οι άλλοι κόμβοι λειτουργούν σωστά και για να στείλουν μηνύματα συμπλέγματος που απαιτούνται για την επισήμανση συγκεκριμένων συνθηκών. Ο δίαυλος συμπλέγματος χρησιμοποιείται επίσης για τη μετάδοση μηνυμάτων “Pub/Sub” σε όλο το σύμπλεγμα και για την ενορχήστρωση των μη αυτόματων αποτυχιών όταν ζητούνται από τους χρήστες (οι μη αυτόματες αποτυχίες είναι αστοχίες που δεν ξεκινούν από τον ανιχνευτή αστοχιών του Redis Cluster, αλλά απευθείας από το διαχειριστή συστήματος). Δεδομένου ότι οι κόμβοι συμπλέγματος δεν είναι σε θέση να ζητήσουν μεσολάβηση αιτημάτων, οι πελάτες ενδέχεται να ανακατευθυνθούν σε άλλους κόμβους χρησιμοποιώντας σφάλματα ανακατεύθυνσης “-MOVED” και “-ASK”. Ο πελάτης είναι θεωρητικά ελεύθερος να στείλει αιτήματα σε όλους τους κόμβους του συμπλέγματος, και να ανακατευθυνθεί εάν χρειαστεί, ώστε ο πελάτης να μη χρειάζεται να διατηρεί την κατάσταση του συμπλέγματος. Ωστόσο, οι πελάτες που μπορούν να αποθηκεύουν προσωρινά τον χάρτη μεταξύ κλειδιών και κόμβων μπορούν να βελτιώσουν την απόδοση των λειτουργιών τους.

Το Redis Cluster χρησιμοποιεί ασύγχρονη αναπαραγωγή μεταξύ κόμβων και το τελευταίο εκλεγμένο κύριο σύνολο δεδομένων αντικαθιστά τελικά όλα τα άλλα αντίγραφα. Υπάρχει πάντα ένα παράθυρο χρόνου όπου είναι δυνατόν να χαθούν εγγραφές κατά τη διάρκεια των διαμερισμών. Ωστόσο, αυτά τα παράθυρα είναι πολύ διαφορετικά στην περίπτωση ενός πελάτη που είναι συνδεδεμένος με την πλειονότητα των κύριων κόμβων και ενός πελάτη που είναι συνδεδεμένος με τη μειονότητα των κύριων κόμβων. Αυτό γιατί το Redis Cluster προσπαθεί περισσότερο να διατηρήσει τις εγγραφές που εκτελούνται από πελάτες που συνδέονται με την πλειοψηφία των κύριων κόμβων, σε σύγκριση με τις εγγραφές που εκτελούνται στην πλευρά της μειονότητας. Μια περίπτωση που μπορεί να προκληθεί απώλεια δεδομένων είναι όταν έρχεται μια εντολή στον κύριο κόμβο. Επειδή η Redis λειτουργεί ασύγχρονα ο κύριος κόμβος εκτελεί αυτή την εντολή και απαντά καταφατικά στον πελάτη χωρίς να ενημερώσει τους υποτελείς κόμβους. Πρώτου όμως προλάβει να περάσει αυτή την εντολή στους υποτελείς κόμβους, ο κύριος κόμβος εμφανίζει κάποιο πρόβλημα και τερματίζει τη λειτουργία του. Όταν εκλεχθεί καινούργιος κύριος κόμβος, από τους υποτελείς, αυτός ο κόμβος δε θα έχει ενημερωθεί με αποτέλεσμα να υπάρξουν ασυμφωνίες. Αυτό μπορεί τις περισσότερες φορές να αποφευχθεί με τη σύγχρονη εκτέλεση αυτής της διαδικασίας με τη χρήση της εντολής WAIT. Όμως σε αυτή τη περίπτωση θυσιάζεται ταχύτητα για συνέπεια.

Όσον αφορά τη διαθεσιμότητα, το Redis Cluster δεν είναι διαθέσιμο στην πλευρά της μειονότητας του διαμερισμού. Στην πλειοψηφία του διαμερισμού υποθέτοντας ότι υπάρχει τουλάχιστον η πλειοψηφία των κύριων κόμβων και ένας υποτελής κόμβος για κάθε μη προσπελάσιμο κύριο κόμβο, το σύμπλεγμα είναι ξανά διαθέσιμο μετά από “NODE_TIMEOUT” χρόνο συν μερικά ακόμη δευτερόλεπτα που απαιτούνται για να εκλεγεί ένας υποτελής κόμβος κύριος και να αποτύχει ο κύριος του (Οι ανακάμψεις από τις αποτυχίες εκτελούνται συνήθως σε διάστημα 1 ή 2 δευτερολέπτων). Αυτό σημαίνει ότι το Redis Cluster έχει σχεδιαστεί για να

επιβιώνει από αστοχίες μερικών κόμβων στο σύμπλεγμα, αλλά δεν είναι η κατάλληλη λύση για εφαρμογές που απαιτούν διαθεσιμότητα σε περίπτωση μεγάλων διαμερισμών του δικτύου. Παρόλα αυτά αν πολλοί κύριοι κόμβοι (master nodes) είναι προβληματικοί, τότε το σύμπλεγμα διακόπτει τη λειτουργία του. Χάρη σε μια δυνατότητα του Redis Cluster που ονομάζεται μετανάστευση αντιγράφων, η διαθεσιμότητα του συμπλέγματος βελτιώνεται σε πολλά σενάρια πραγματικού κόσμου από το γεγονός ότι οι υποτελείς κόμβοι μεταναστεύουν σε ορφανούς κύριους κόμβους (οι κύριοι κόμβοι που δεν έχουν πλέον υποτελείς κόμβους). Έτσι σε κάθε επιτυχημένο συμβάν αποτυχίας, το σύμπλεγμα μπορεί να αναδιαμορφώσει τη διάταξη των υποτελών κόμβων, προκειμένου να αντισταθεί καλύτερα στην επόμενη αποτυχία.

Σχετικά με την απόδοση, σε ένα Redis Cluster, οι κόμβοι δεν πραγματοποιούν εντολές μεσολάβησης στον σωστό κόμβο που είναι υπεύθυνος για ένα δεδομένο κλειδί, αλλά αντ 'αυτού ανακατευθύνουν τους πελάτες στους σωστούς κόμβους που εξυπηρετούν ένα δεδομένο τμήμα του χώρου κλειδιών. Τελικά οι πελάτες λαμβάνουν μια ενημερωμένη αναπαράσταση του συμπλέγματος και ποιος κόμβος εξυπηρετεί ποιο υποσύνολο κλειδιών, ώστε κατά τη διάρκεια των κανονικών λειτουργιών οι πελάτες να επικοινωνούν απευθείας με τους σωστούς κόμβους για να στείλουν μια δεδομένη εντολή. Λόγω της χρήσης ασύγχρονης αναπαραγωγής, οι κόμβοι δεν περιμένουν την αναγνώριση εγγραφής άλλων κόμβων (εάν δε ζητείται ρητά χρησιμοποιώντας την εντολή WAIT). Επίσης, επειδή οι εντολές πολλαπλών κλειδιών περιορίζονται μόνο σε κοντινά κλειδιά, τα δεδομένα δεν μετακινούνται ποτέ μεταξύ κόμβων εκτός από την περίπτωση της αναδιαμόρφωσης. Οι κανονικές λειτουργίες αντιμετωπίζονται ακριβώς όπως στην περίπτωση ενός μόνο κόμβου της Redis. Αυτό σημαίνει ότι σε ένα σύμπλεγμα Redis με κύριους κόμβους N η απόδοση παραμένει ίση με το να υπήρχε ένας κόμβος της Redis πολλαπλασμένο με το N εφόσον ο σχεδιασμός της επιτρέπει γραμμική κλιμακωσιμότητα. Ταυτόχρονα, το ερώτημα εκτελείται συνήθως σε ένα ταξίδι μετ 'επιστροφής, δεδομένου ότι οι πελάτες διατηρούν συνήθως μόνιμες συνδέσεις με τους κόμβους, οπότε τα στοιχεία χρόνου καθυστέρησης είναι επίσης ίδια με τη μεμονωμένη περίπτωση κόμβου Redis.

Οι κόμβοι σε ένα σύμπλεγμα επικοινωνούν μεταξύ τους για την ανταλλαγή δεδομένων, τον συγχρονισμό τους, την αναζήτησή δεδομένων και το διαμοιρασμό δεδομένων με τη τεχνική sharding όπως αναφέρθηκε παραπάνω. Για τη τεχνική αυτή δε χρησιμοποιείται συνεπής κατακερματισμός όπως συνήθως, αλλά μια διαφορετική μορφή θρυμματισμού όπου κάθε κλειδί είναι εννοιολογικά μέρος αυτού που ονομάζεται θυρίδα κατακερματισμού (hash slot). Υπάρχουν 16384 θυρίδες κατακερματισμού στο σύμπλεγμα, και για να υπολογίσουμε ποια είναι η θυρίδα κατακερματισμού ενός δεδομένου κλειδιού, παίρνουμε απλά το CRC16 του κλειδιού modulo 16384. Το ποιος κόμβος θα αναλάβει ποιες θυρίδες υπολογίζεται αναλογικά. Αν για παράδειγμα υπάρχουν τρεις κόμβοι στο σύμπλεγμα τότε ο Α κόμβος θα αναλάβει τις θυρίδες 0-5500, ο Β κόμβος τις θυρίδες 5501-11000 και ο Γ κόμβος τις θυρίδες 11001-16383. Κάθε κόμβος σε αυτό το σύμπλεγμα προϋποθέτει δύο ανοιχτές TCP συνδέσεις. Η μία σύνδεση είναι στη θύρα 6379 και χρησιμοποιείται στο να επιτελεί λειτουργίες και εντολές και άλλη μία θύρα (συνήθως προσθέτει 10000 στη

προηγούμενη πόρτα για να πάρει τον ζητούμενο αριθμό) η οποία χρησιμοποιείται για την επικοινωνία μεταξύ των κόμβων που υπάρχουν στο ίδιο σύμπλεγμα.

Οι κόμβοι στο Redis Cluster ανταλλάσσουν συνεχώς πακέτα ping και pong (το πακέτο pong το απαντάει η redis όταν δέχεται ένα πακέτο ping). Αυτά τα δύο είδη πακέτων έχουν την ίδια δομή και και τα δύο φέρουν σημαντικές πληροφορίες διαμόρφωσης. Η μόνη πραγματική διαφορά είναι το πεδίο τύπου μηνύματος. Το άθροισμα των πακέτων ping και pong αναφέρονται ως πακέτα καρδιακού παλμού. Συνήθως οι κόμβοι στέλνουν πακέτα ping που θα ενεργοποιήσουν τους δέκτες να απαντήσουν με πακέτα pong. Ωστόσο, αυτό δεν είναι απαραίτητα αλήθεια. Είναι δυνατό για τους κόμβους να στέλνουν απλώς πακέτα pong για να στέλνουν πληροφορίες σε άλλους κόμβους σχετικά με τη διαμόρφωσή τους, χωρίς να περιμένουν απάντηση. Αυτό είναι χρήσιμο, για παράδειγμα, για τη μετάδοση μιας νέας διαμόρφωσης το συντομότερο δυνατό. Συνήθως ένας κόμβος θα κάνει ping μερικούς τυχαίους κόμβους κάθε δευτερόλεπτο έτσι ώστε ο συνολικός αριθμός των πακέτων ping που αποστέλλονται (και τα πακέτα pong που λαμβάνονται) από κάθε κόμβο να είναι ένα σταθερό ποσό και να είναι ανεξάρτητο από τον αριθμό των κόμβων του συμπλέγματος. Ωστόσο, κάθε κόμβος φροντίζει να κάνει ping σε κάθε άλλο κόμβο που δεν έχει στείλει ping ή δεν έχει λάβει pong για περισσότερο από το μισό χρόνο "NODE_TIMEOUT" . Πριν παρέλθει ο χρόνος "NODE_TIMEOUT" , οι κόμβοι προσπαθούν επίσης να επανασυνδέσουν τον σύνδεσμο TCP με έναν άλλο κόμβο για να διαβεβαιώσουν ότι οι κόμβοι δεν πιστεύεται ότι δεν είναι προσβάσιμοι μόνο και μόνο επειδή υπάρχει πρόβλημα στην τρέχουσα σύνδεση TCP. Ο αριθμός των μηνυμάτων που ανταλλάσσονται μπορεί να είναι αρκετά μεγάλος εάν το "NODE_TIMEOUT" έχει οριστεί σε μια μικρή τιμή και ο αριθμός των κόμβων (N) είναι πολύ μεγάλος, καθώς κάθε κόμβος θα προσπαθήσει να κάνει ping σε κάθε άλλο κόμβο για τον οποίο δεν έχουν νέες πληροφορίες κάθε μισό του χρόνου "NODE_TIMEOUT". Για παράδειγμα, σε ένα σύμπλεγμα 100 κόμβων με χρονικό όριο κόμβου 60 δευτερόλεπτα, κάθε κόμβος θα προσπαθεί να στέλνει 99 pings κάθε 30 δευτερόλεπτα, με συνολικό ποσό pings 3,3 ανά δευτερόλεπτο. Πολλαπλασιαζόμενος με 100 κόμβους, αυτό είναι 330 pings ανά δευτερόλεπτο στο συνολικό σύμπλεγμα. Υπάρχουν τρόποι μείωσης του αριθμού των μηνυμάτων, ωστόσο δεν έχουν αναφερθεί προβλήματα με το εύρος ζώνης που χρησιμοποιείται αυτήν τη στιγμή για την ανίχνευση αστοχιών σε ένα Redis Cluster, οπότε προς το παρόν χρησιμοποιείται ο προφανής και άμεσος σχεδιασμός. Ακόμη και στο παραπάνω παράδειγμα, τα 330 πακέτα ανά δευτερόλεπτο που ανταλλάσσονται κατανέμονται ομοιόμορφα σε 100 διαφορετικούς κόμβους, επομένως η κίνηση που λαμβάνει κάθε κόμβος είναι αποδεκτή.

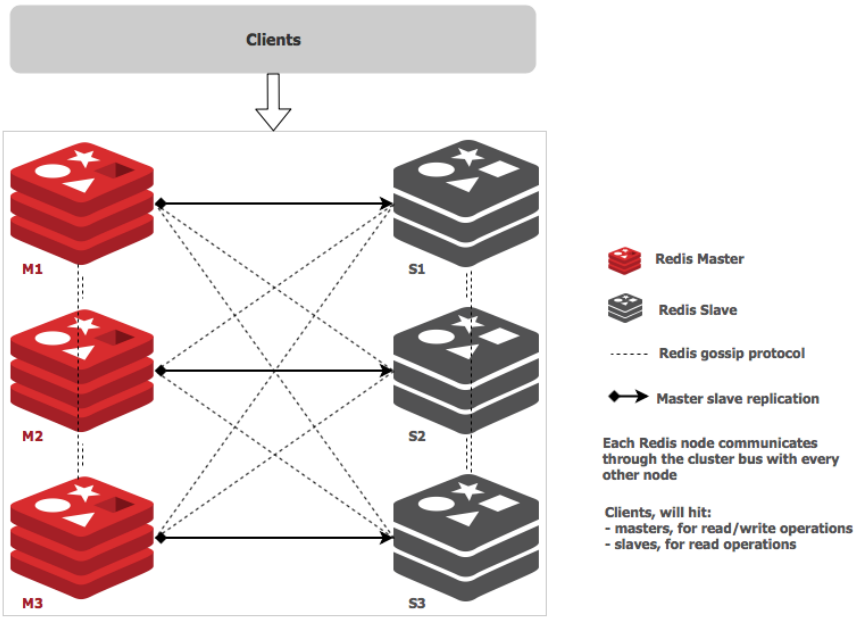
Το Redis Cluster χρησιμοποιεί μια ιδέα παρόμοια με τον όρο του αλγορίθμου Raft[24], το "term". Στο Redis Cluster ο αντίστοιχος αυτός όρος ονομάζεται "epoch", και χρησιμοποιείται για να δώσει σταδιακή εκδοχή σε συμβάντα. Όταν πολλοί κόμβοι παρέχουν αντικρουόμενες πληροφορίες, είναι δυνατό για έναν άλλο κόμβο να κατανοήσει ποια κατάσταση είναι η πιο ενημερωμένη. Η μεταβλητή "currentEpoch" είναι ένας μη προσημασμένος αριθμός 64 bit. Κατά τη δημιουργία κόμβων, κάθε κόμβος του Redis Cluster, τόσο οι κύριοι κόμβοι όσο και οι υποτελείς, ορίζουν το "currentEpoch" σε 0. Κάθε φορά που λαμβάνεται ένα πακέτο από έναν άλλο κόμβο, εάν το "epoch" του αποστολέα (μέρος της κεφαλίδας μηνυμάτων διαύλου συμπλέγματος) είναι μεγαλύτερο από το

“epoch” του τοπικού κόμβου, το “currentEpoch” ενημερώνεται στο “epoch” του αποστολέα. Λόγω αυτών των σημασιολογικών, τελικά όλοι οι κόμβοι θα συμφωνήσουν με τη μεγαλύτερη μεταβλητή “epoch” στο σύμπλεγμα.

Η μεταβλητή αυτή (“currentEpoch”) αναφέρθηκε επειδή θα μας βοηθήσει να κατανοήσουμε τη διαδικασία εκλογών ώστε ένας υποτελής κόμβου να προαχθεί σε κύριο. Η εκλογή και η προώθηση των υποτελών κόμβων γίνεται από τους υποτελείς κόμβους, με τη βοήθεια κύριων κόμβων που ψηφίζουν για την προώθηση του υποτελούς. Η εκλογή συμβαίνει όταν ένας κύριος κόμβος βρίσκεται σε κατάσταση αποτυχίας (FAIL state) υπό την προϋπόθεση ότι τουλάχιστον ενός από τους υποτελείς του κόμβους έχει τις δυνατότητες για να γίνει κύριος. Προκειμένου ένας υποτελής κόμβος να προαχθεί, πρέπει να ξεκινήσει εκλογές και να τις κερδίσει. Όλοι οι υποτελείς κόμβοι μπορούν να ξεκινήσουν μια εκλογή εάν ο κύριος κόμβος βρίσκεται σε κατάσταση αποτυχίας, ωστόσο μόνο ένας θα κερδίσει τις εκλογές και θα προαχθεί σε κύριο κόμβο. Ένας υποτελής κόμβος ξεκινά μια εκλογή όταν πληρούνται οι ακόλουθες προϋποθέσεις:

- Ο κύριος κόμβος του υποτελούς βρίσκεται σε κατάσταση αποτυχίας.
- Ο κύριος κόμβος εξυπηρετούσε κάποιο slot.
- Ο σύνδεσμος αναπαραγωγής υποτελών κόμβων αποσυνδέθηκε από τον κύριο κόμβο για όχι περισσότερο από ένα δεδομένο χρονικό διάστημα, προκειμένου να διασφαλιστεί ότι τα δεδομένα του προωθούμενου υποτελούς κόμβου είναι αρκετά πρόσφατα. Αυτός ο χρόνος μπορεί να ρυθμιστεί από το χρήστη.

Προκειμένου να εκλεγεί, το πρώτο βήμα για έναν υποτελή κόμβο είναι να αυξήσει το μετρητή του “currentEpoch” και να ζητήσει ψήφους από τους υπόλοιπους κύριος κόμβους. Αυτές οι ψήφοι ζητούνται από τον υποτελή κόμβο μεταδίδοντας ένα πακέτο “FAILOVER_AUTH_REQUEST” σε κάθε κύριο κόμβο του συμπλέγματος. Στη συνέχεια, περιμένει για μέγιστο χρόνο δύο φορές επί το “NODE_TIMEOUT” για να φτάσουν οι απαντήσεις (αλλά πάντα για τουλάχιστον 2 δευτερόλεπτα). Μόλις ένας κύριος κόμβος ψηφίσει έναν υποτελή, απαντώντας θετικά με ένα “FAILOVER_AUTH_ACK”, δεν μπορεί πλέον να ψηφίσει έναν άλλο υποτελή του ίδιου κύριου κόμβου για περίοδο NODE_TIMEOUT * 2. Σε αυτήν την περίοδο δεν θα μπορεί να απαντήσει σε άλλα αιτήματα εξουσιοδότησης για τον ίδιο κύριο κόμβο. Αυτό δεν χρησιμοποιείται για την εγγύηση της ασφάλειας, αλλά είναι χρήσιμο για την αποφυγή εκλογής πολλών υποτελών κόμβων (ακόμα και με διαφορετικό configEpoch) περίπου την ίδια ώρα, κάτι που συνήθως δεν είναι επιθυμητό. Ένας υποτελής απορρίπτει τυχόν απαντήσεις AUTH_ACK με ένα epoch που είναι μικρότερο από το currentEpoch τη στιγμή που στάλθηκε το αίτημα ψήφου. Αυτό διασφαλίζει ότι δεν προσμετρούνται οι ψήφοι που προορίζονται για προηγούμενες εκλογές. Μόλις ο υποτελής λάβει ACK από την πλειοψηφία των κύριων κόμβων, κερδίζει τις εκλογές. Διαφορετικά, εάν η πλειοψηφία δεν επιτευχθεί εντός της περιόδου NODE_TIMEOUT *2 (αλλά πάντα τουλάχιστον 2 δευτερόλεπτα), η εκλογή ματαιώνεται και μια νέα θα δοκιμαστεί ξανά μετά από χρόνο NODE_TIMEOUT * 4 (και πάντα τουλάχιστον 4 δευτερόλεπτα).



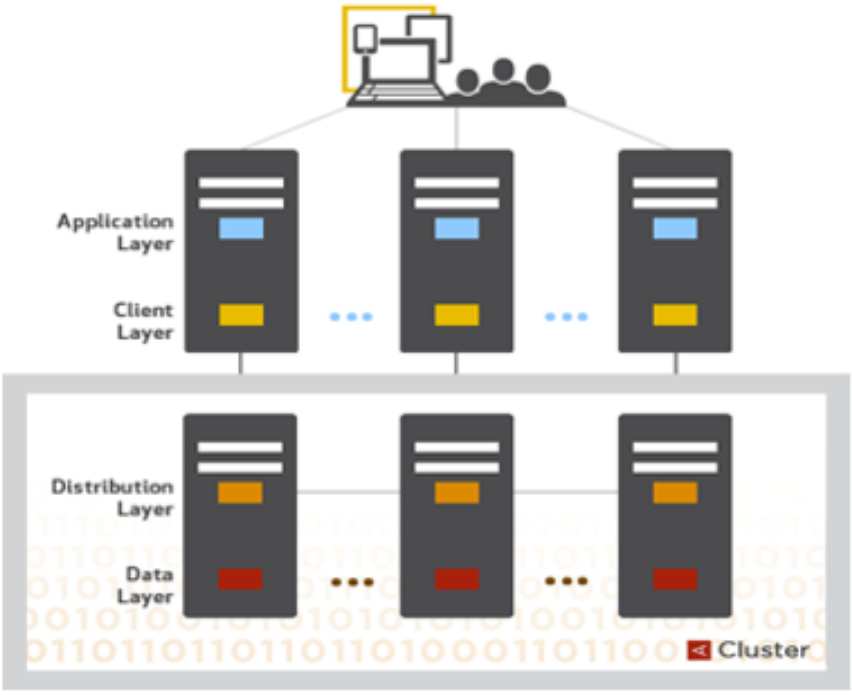
Εικόνα 1: Αρχιτεκτονική ενός συμπλέγματος της Redis. Πηγή: https://i.stack.imgur.com/5W4UZ.png

3.2 Aerospike

Η Aerospike[25] είναι μια μη σχεσιακή (NoSQL) βάση δεδομένων ανοιχτού κώδικα η οποία έχει σχεδιαστεί για να είναι αποδοτική σε μνήμες τύπου flash και να έχει τη δυνατότητα να αποθηκεύει τα δεδομένα προσωρινά στη μνήμη (ram). Αρχικά η Aerospike ήταν γνωστή με το όνομα Citrusleaf 2.0. Τον Αύγουστο του 2012, η εταιρία - η οποία παρέχει τη βάση δεδομένων της από το 2010 - μετονομάστηκε τόσο σε εταιρικό επίπεδο όσο και στο όνομα του λογισμικού που παρείχε, στο όνομα Aerospike. Το όνομα Aerospike προέρχεται από τον κινητήρα aerospike, έναν τύπο πυραύλου ακροφυσίου που μπορεί να διατηρήσει την απόδοση εξόδου του σε μεγάλο εύρος υψομέτρων και ο λόγος που επιλέχθηκε το όνομα αυτό είναι για να αναδείξει την ικανότητα του λογισμικού να κλιμακώνεται. Το 2012, η Aerospike απέκτησε την βάση δεδομένων AlchemyDB και ενσωμάτωσε τις λειτουργίες τους με αποτέλεσμα να εισαχθεί και ένα σχεσιακό σύστημα διαχείρισης δεδομένων. Στις 24 Ιουνίου του 2014 η Aerospike έγινε ανοιχτού κώδικα κάτω από την άδεια AGPL 3.0 για τον διακομιστή βάσης δεδομένων Aerospike και την έκδοση Apache Licence Version 2.0 για το κιτ ανάπτυξης λογισμικού πελατών της Aerospike.

Η Aerospike είναι γραμμένη στη γλώσσα προγραμματισμού C. Είναι μια κατανεμημένη βάση δεδομένων. Η αρχιτεκτονική της έχει τρεις βασικούς στόχους[7]. Να δημιουργήσει μια ευέλικτη και επεκτάσιμη πλατφόρμα για εφαρμογές διαδικτύου, να παρέχει την ευρωστία και την αξιοπιστία (όπως στο ACID) που αναμένεται από τις παραδοσιακές βάσεις δεδομένων και να παρέχει λειτουργική αποδοτικότητα με ελάχιστη εμπλοκή του χρήστη. Με αυτούς τους τρεις στόχους η αρχιτεκτονική της Aerospike αποτελείται από τρία επίπεδα.

- Το επίπεδο του Πελάτη (Client Layer): Αυτό το επίπεδο επίγνωσης του συμπλέγματος (cluster-aware) περιλαμβάνει βιβλιοθήκες πελατών ανοιχτού κώδικα, οι οποίες εφαρμόζουν το API της Aerospike, επιβλέπουν τους κόμβους του συμπλεγματος και γνωρίζουν ανά πάσα στιγμή που βρίσκονται τα δεδομένα στο σύμπλεγμα.
- Το επίπεδο Συμπλέγματος και Διαμοιρασμού Δεδομένων (Clustering and Data Distribution Layer): Αυτό το επίπεδο διαχειρίζεται την επικοινωνία του συμπλέγματος αυτοματοποιεί τη διαδικασία επανεκκίνησης σε περίπτωση αποτυχίας, την αναπαραγωγή, την αναπαραγωγή μεταξύ διαφόρων κέντρων δεδομένων (XDR) και την έξυπνη επανεξισορρόπηση και μετεγκατάσταση δεδομένων.
- Το επίπεδο Αποθήκευσης Δεδομένων (Data Storage Layer): Αυτό το επίπεδο αποθηκεύει αξιόπιστα τα δεδομένα στη δυναμική μνήμη DRAM και στη μνήμη Flash για γρήγορη ανάκτηση δεδομένων. Η Aerospike είναι ένα μια βάση δεδομένων τύπου κλειδιού-τιμής με ένα μοντέλο δεδομένων χωρίς κάποιο συγκεκριμένο σχηματισμό (schemaless). Τα δεδομένα ρέουν σε διάφορα κοντέινερ, που ονομάζονται χώροι ονομάτων (namespaces), που είναι σημασιολογικά παρόμοια με βάσεις δεδομένων σε ένα σύστημα RDBMS. Μέσα σε ένα χώρου ονομάτων, τα δεδομένα υποδιαιρούνται σε σύνολα (πίνακες σε RDBMS) και εγγραφές (σειρές σε RDMBS). Κάθε



Εικόνα 2:Αρχιτεκτονική της Aerospike. Πηγή: https://www.aerospike.com/docs/architecture/assets/as-architecture.png

εγγραφή έχει ένα δεικτοδοτούμενο κλειδί το οποίο είναι μοναδικό στο σύνολο και έναν η περισσότερους κάδους (στήλες σε RDBMS) που διατηρούν τιμές που σχετίζονται με την εγγραφή.

Παρακάτω θα αναλύσουμε τους βασικούς τύπους δεδομένων που μπορούν να αποθηκευτούν στους κάδους της Aerospike και πως υλοποιούνται πραγματικά. Η Aerospike χωρίζει τους τύπους δεδομένων της ως βασικούς τύπους δεδομένων και ως σύνθετους τύπους δεδομένων. Οι βασικοί τύποι δεδομένων είναι οι:

- Ακέραιοι (Integers): Οι Ακέραιοι είναι αριθμοί 64 bit. Κάθε ακέραιος απαιτεί 8 bytes μνήμης για την αποθήκευση του. Για τα σημεία επαφής στη βάση δεδομένων όπου οι Ακέραιοι πρέπει να λάβουν μια προσήμανση, όπως στην περίπτωση όπου χρησιμοποιούνται για δευτερεύον δεικτοδότηση, τότε η τιμή ενός Ακεραίου μπορεί να πάρει τιμή από -2^63 εώς και 2^63-1. Στους Ακέραιους μπορεί να εφαρμοστεί η λειτουργία της ατομικής πρόσθεσης για να υλοποιηθούν μετρητές. Η δευτερεύουσα δεικτοδότηση υποστηρίζονται στο συγκεκριμένο τύπο δεδομένων, τόσο για τα ερωτήματα ισότητας όσο και για τα ερωτήματα εύρους.
- Συμβολοσειρές (Strings): Οι Συμβολοσειρές είναι αφηρημένες συστοιχίες από bytes. Δε δεσμεύονται με κωδικοποίηση, που σημαίνει ότι έχουν τη δυνατότητα να υποστηρίξουν οποιαδήποτε κωδικοποίηση επιθυμεί ο χρήστης. Για σημεία επαφής με τη βάση δεδομένων όπου οι Συμβολοσειρές πρέπει να έχουν μια κωδικοποίηση, τότε χρησιμοποιείται η κωδικοποίηση UTF-8. Οι διάφορες βιβλιοθήκες των πελατών μπορούν να επιβάλουν τη δική τους κωδικοποίηση. Η συμβατότητα μεταξύ των γλωσσών εξαρτάται από την εφαρμογή. Οι συμβολοσειρές έχουν όριο μεγέθους 128 KB. Επίσης υποστηρίζουν τις ατομικές λειτουργίες της προσθήκης (prepend) και

της προσάρτησης (append). Ακόμα η δευτερεύουσα δεικτοδότηση υποστηρίζεται στο συγκεκριμένο τύπο δεδομένων για ερωτήματα που αφορούν ισότητα.

- Bytes: Τα Bytes είναι πίνακες δεδομένου μεγέθους που περιέχουν bytes. Οποιαδήποτε δυαδικά δεδομένα οποιουδήποτε τύπου μπορούν να αποθηκευτούν στα Bytes. Επίσης τα Bytes είναι ένας τύπος δεδομένων ο οποίος στο τέλος του δεν περιέχει τον χαρακτήρα '\o' (non null terminated).
- Double: Τα Doubles είναι είναι ένας τύπος δεδομένων ο οποίος είναι ουσιαστικά ένας αριθμός 64 bit διπλής ακρίβειας. Στα Doubles μπορεί να εφαρμοστεί η λειτουργία της ατομικής πρόσθεσης για να υλοποιηθούν μετρητές

Οι σύνθετοι τύποι δεδομένων υποστηρίζουν ατομικές λειτουργίες για την επεξεργασία των πεδίων τους, όπως επίσης και λειτουργίες για εφαρμογή ερωτημάτων στα δεδομένα τους. Οι τύποι αυτοί είναι:

- Λίστες (Lists): Οι λίστες στην Aerospike μπορούν να περιέχουν έναν από τους βασικούς τύπους δεδομένων που περιγράψαμε προηγουμένως η μπορούν να περιέχουν και άλλες λίστες η χάρτες που θα περιγράψουμε παρακάτω. Οι λειτουργίες πάνω στις λίστες είναι βελτιστοποιημένες ώστε να χειρίζονται τις λίστες απευθείας στο διακομιστή της Aerospike. Κάθε λίστα έχει δείκτες οι οποίοι αντιπροσωπεύουν τη θέση του κάθε δεδομένου, την τιμή η οποία είναι πραγματική τιμή του δεδομένου και το βαθμό ο οποίος αντιπροσωπεύει τη σειρά τιμών των στοιχείων της λίστας, δηλαδή ο μικρότερος αριθμός στη λίστα έχει βαθμό μηδέν. Οι δείκτες μπορούν να πάρουν και αρνητικές τιμές με το μείον ένα να είναι το τελευταίο στοιχείο της λίστας. Επίσης και οι βαθμοί μπορούν να πάρουν αρνητικές τιμές με το μείον ένα να είναι το μεγαλύτερο στοιχείο στη λίστα. Τα στοιχεία σε μια λίστα μπορούν να προσπελαστούν και με τους τρεις παραπάνω τρόπους. Οι λίστες χωρίζονται σε δύο κατηγορίες. Τις ταξινομημένες λίστες και τις μη ταξινομημένες λίστες. Οι ταξινομημένες λίστες διατηρούν το βαθμό κατάταξης τους και κάθε φορά που εισάγεται ένα νέο στοιχείο η λίστα αυτοταξινομείται. Σε μια ταξινομημένη λίστα, στοιχεία τα οποία είναι μικτού τύπου ταξινομούνται αρχικά με τον τύπο τους και έπειτα με τη τιμή τους. Η σειρά με την οποία ταξινομούνται οι τύποι είναι: κενό, BOOLEAN, Ακέραιος, Συμβολοσειρά, Λίστα, Χάρτης, BYTES, DOUBLE, GEOJSON και INF. Η δεύτερη κατηγορία λιστών είναι οι μη ταξινομημένες λίστες. Οι μη ταξινομημένες λίστες διατηρούν τη σειρά εισαγωγής των στοιχείων κατα τη διάρκεια εισαγωγής νέων στοιχείων στη λίστα. Σε μία μη ταξινομημένη λίστα η ανάκτηση δεδομένων με βάση το βαθμό ακολουθεί την παραπάνω προτεραιότητα που περιγράφηκε. Οι λίστες περιορίζονται όσον αφορά το μέγεθος τους από τη μεταβλητή write-block-size. Επίσης οι λίστες δεν υποστηρίζονται από συναρτήσεις γραμμένες από το χρήστη στη προγραμματιστική γλώσσα Lua (Lua UDFs).
- Χάρτες (Maps): Οι χάρτες στην Aerospike μπορούν να περιέχουν έναν από τους βασικούς τύπους δεδομένων που περιγράψαμε προηγουμένως η μπορούν να περιέχουν και άλλες λίστες η χάρτες. Οι λειτουργίες στους χάρτες είναι βελτιστοποιημένη ώστε να μπορούν να χειριστούν ζεύγη κλειδιού-τιμής απευθείας στο διακομιστή της Aerospike. Όπως και οι λίστες, οι χάρτες έχουν δείκτες οι οποίοι αντιπροσωπεύουν τη θέση του κάθε δεδομένου, την τιμή η οποία είναι

πραγματική τιμή του δεδομένου και το βαθμό ο οποίος αντιπροσωπεύει τη σειρά τιμών των στοιχείων του χάρτη. Τα στοιχεία σε ένα χάρτη αποθηκεύονται ανάλογα με το τύπο ταξινόμησης ενός χάρτη. Οι μη ταξινομημενοι χάρτες δεν έχουν συγκεκριμένο τρόπο ταξινόμησης, ενώ οι K-ταξινομημένοι χάρτες αποθηκεύουν δεδομένα με βάση τη σειρά των κλειδιών. Οι KV-ταξινομημένοι χάρτες είναι επίσης K-ταξινομημένοι χάρτες, αλλά έχουν ένα παραπάνω δείκτη ο οποίος είναι ο συνδυασμός τιμής-σειράς ανάλογα με την διαμόρφωση του χώρου ονομάτων. Τα δεδομένα σε ένα χάρτη μπορούν επίσης να προσπελαστούν με βάση τη σειρά, είτε με βάση τους δείκτες, είτε με βάση το βαθμό. Οι χάρτες περιορίζονται όσον αφορά το μέγεθος τους από τη μεταβλητή write-block-size. Επίσης οι χάρτες δεν υποστηρίζονται από συναρτήσεις γραμμένες από το χρήστη στη προγραμματιστικη γλώσσα Lua (Lua UDFs).

- GeoJSON: Το GeoJSON είναι ένας τύπος δεδομένων όπου ουσιαστικά υλοποιείται με συμβολοσειρές οι οποίες έχουν τη μορφή JSON και χρησιμοποιούνται για να κωδικοποιούν γεωχωρικά δεδομένα.
- HyperLogLog και Πιθανοτικά Δεδομένα (HyperLogLog and Probabilistic Data): Το HyperLogLog είναι ένας πιθανοτικός τύπος δεδομένων που παρέχει λειτουργίες μαθηματικών συνόλων όπως προσθήκη, μέτρηση, διασταύρωση και ένωση. Καθώς ο αριθμός των στοιχείων σε ένα σύνολο δεδομένων αυξάνεται εξαιρετικά, το HyperLogLog είναι αποδοτικό τόσο στο χώρο όσο και στη κεντρική μονάδα επεξεργασίας (CPU) ανεξάρτητα από το μέγεθος του λογικού συνόλου δεδομένων. Παρέχει μια γρήγορη, κατά προσέγγιση μέτρηση του αριθμού των στοιχείων της διασταύρωσης ή ένωσης πολλαπλών συνόλων. Αυτά τα δεδομένα αποτελούν τη βάση άντλησης διαφόρων πιθανοτήτων για κάθε είδους εφαρμογή.

Αναλύοντας λοιπόν όλους τους βασικούς τύπους δεδομένων που χρησιμοποιεί η Aerospike θα προχωρήσουμε στην ανάλυση του ευρετηριασμού (indexing) που χρησιμοποιεί καθώς θα μας βοηθήσει στο να κατανοήσουμε τη συμπεριφορά της στα διάφορα τεστ που θα της κάνουμε αργότερα. Η Aerospike έχει τη δυνατότητα για πρωτεύον και δευτερεύον ευρετήρια (primary and secondary indexing). Τα πρωτεύον είναι ένα μείγμα κατανεμημένης τεχνολογίας πίνακα κατακερματισμού με κατανεμημένη δομή δέντρου σε κάθε διακομιστή. Ολόκληρος ο χώρος κλειδιών στο χώρο ονομάτων (namespace) διαχωρίζεται μέσω μιας ισχυρής συνάρτησης κατακερματισμού σε διαμερισμούς. Συνολικά 4096 κατατμήσεις κατανέμονται εξίσου σε κόμβους του συμπλέγματος. Η Aerospike χρησιμοποιεί μια μορφή κόκκινων-μαύρων δέντρων (red-black trees) που ονομάζονται sprigs [26]. Υπάρχουν δύο λόγοι που χρησιμοποιούνται τα sprigs και όχι τα απλά κόκκινα-μαύρα δέντρα. Αρχικά η διάσχιση ενός τέτοιου δέντρου με μεγάλο αριθμό δεδομένων είναι πολύ χρονοβόρα και μπορεί να είναι η πιο αργή λειτουργία σε μια δοσοληψία. Και δεύτερον ενώ οι προσπελάσεις στο δέντρο δε δημιουργούν πρόβλημα κλειδώματος, οι λειτουργίες δημιουργίας και διαγραφής δεδομένων δημιουργούν. Οπότε σε αυτές τις διαδικασίες όλο το σύστημα περιμένει να ολοκληρωθούν ώστε να συνεχιστεί η λειτουργία του προγράμματος με αποτέλεσμα να καθυστερούν την όλη εκτέλεση. Γι' αυτό δημιουργήθηκαν τα sprigs που είναι ουσιαστικά η διαχώριση ενός δέντρου ευρετηρίου ενός διαμερισμού, σε πολλά δευτερεύοντα δέντρα (sprigs). Έτσι το βάθος/μέγεθος ενός δέντρου μειώνεται με αποτέλεσμα την ταχύτερη

αναζήτησή και προσπέλαση των στοιχείων του δέντρου, όπως επίσης και την υποστήριξη της παρουσίας πολλαπλών δέντρων, άρα και μείωση ζευγών κλειδώματος ανά διαμέρισμα, με κάθε κλείδωμα (locks) να εφαρμόζεται σε ένα ή περισσότερα δευτερεύοντα δέντρα, για τη μείωση των συγκρούσεων (conflicts) κλειδώματος.

Το πρωτεύον ευρετήριο βρίσκεται στο 20ο byte κατακερματισμού και ονομάζεται σύνοψη (digest) του καθορισμένου πρωτεύοντος ευρετηρίου. Ενώ αυτό επεκτείνει το μέγεθος κλειδιού ορισμένων εγγραφών (για παράδειγμα, ένα ακέραιο κλειδί που είναι μόνο 8-bytes), είναι προτιμότερο καθώς η λειτουργία κώδικα είναι προβλέψιμη ανεξάρτητα από το μέγεθος ή την κατανομή του κλειδιού εισαγωγής. Όταν ένας διακομιστής αποτύχει, τα ευρετήρια σε άλλον διακομιστή είναι άμεσα διαθέσιμα. Εάν ο αποτυχημένος διακομιστής παραμείνει εκτός λειτουργίας, τα δεδομένα αρχίζουν να εξισορροπούνται εκ νέου και τα αναπαραγόμενα ευρετήρια δημιουργούνται σε νέους κόμβους. Το κύριο ευρετήριο προέρχεται από τα ίδια τα δεδομένα και μπορεί να ξαναχτιστεί από αυτά τα δεδομένα, ανάλογα με τη ρύθμιση παραμέτρων για γρήγορη επανεκκίνηση. Για γρήγορες αναβαθμίσεις του συμπλέγματος με ελάχιστο χρόνο διακοπής λειτουργίας, η Aerospike διαθέτει τη δυνατότητα γρήγορης επανεκκίνησης. Η γρήγορη επανεκκίνηση εκχωρεί τη μνήμη ευρετηρίου από ένα τμήμα κοινόχρηστης μνήμης των Linux. Για προγραμματισμένους τερματισμούς λειτουργίας και επανεκκινήσεις (για παράδειγμα, για αναβάθμιση), κατά την επανεκκίνηση ο διακομιστής συνδέεται εκ νέου στο κοινόχρηστο τμήμα μνήμης και ενεργοποιεί τα κύρια ευρετήρια χωρίς σάρωση δεδομένων του χώρου αποθήκευσης.

Εκτός από πρωτεύον ευρετήρια η Aerospike υποστηρίζει και δευτερεύον ευρετήρια. Τα δευτερεύοντα ευρετήρια βρίσκονται σε ένα μη πρωτεύον κλειδί, το οποίο επιτρέπει τη μοντελοποίηση σχέσεων απο ένα σε πολλά. Τα δευτερεύοντα ευρετήρια καθορίζονται διαφορετικά σε κάθε κάδο (όπως με τις στήλες RDBMS). Αυτό επιτρέπει αποτελεσματικές ενημερώσεις και ελαχιστοποιεί το ποσό των πόρων που απαιτούνται για την αποθήκευση των ευρετηρίων. Τα δευτερεύοντα ευρετήρια αποθηκεύονται στη DRAM για αν είναι πιο άμεσα διαθέσιμα. Ακόμη, είναι φτιαγμένα μέσα σε κάθε κόμβο του συμπλέγματος και βρίσκονται μαζί με τα πρωτεύον ευρετήρια και η κάθε εισαγωγή στο δευτερεύον ευρετήριο περιέχει μόνο αναφορές σε εγγραφές τοπικές του κόμβου στον οποίο βρίσκεται. Ακόμη, ένα δευτερεύον ευρετήριο περιέχει δείκτες τόσο σε πρωτεύον δεδομένα τα οποία υπάρχουν στους κόμβους όσο και σε αντίγραφα των δεδομένων αυτών. Η Aerospike παρακολουθεί ποια ευρετήρια δημιουργούνται σε μια παγκόσμια συντηρημένη δομή δεδομένων με το σύστημα μεταδεδομένων (SMD). Για να δημιουργηθεί ένα δευτερεύον ευρετήριο, καθορίζεται ένας χώρος ονομάτων, ένα σετ, ένας κάδος, ένας τύπος κοντέινερ (λίστα, χάρτης) και ένας τύπος δεδομένων (ακέραιος, συμβολοσειρά, λίστα, χάρτης κ.ο.κ.). Μετά την επιβεβαίωση από το SMD, κάθε κόμβος δημιουργεί το δευτερεύον ευρετήριο σε λειτουργία εγγραφής και ξεκινά τη σάρωση όλων των δεδομένων και την εισαγωγή καταχωρήσεων στο δευτερεύον ευρετήριο. Στις διαγραφές δεδομένων τα δεδομένα δεν διαβάζονται από το δίσκο για να διαγράψουν την καταχώρηση από το δευτερεύον ευρετήριο. Αυτό έχει σαν αποτέλεσμα την αποφυγή της υπερβολικής επιβάρυνσης του υποσυστήματος εισόδου/εξόδου. Οι υπόλοιπες καταχωρήσεις στο δευτερεύον ευρετήριο διαγράφονται από ένα νήμα το οποίο εκτελείται στο παρασκήνιο, το οποίο

ξυπνά σε τακτά χρονικά διαστήματα και εκτελεί αυτή την εκκαθάριση. Αυτός ο Συλλέκτης απορριμμάτων έχει σχεδιαστεί για να είναι μη παρεμβατικός. Δημιουργεί μια λίστα καταχωρήσεων για κατάργηση σε μικρές παρτίδες και στη συνέχεια τις διαγράφει αργά από το ευρετήριο. Αυτό απαιτεί παροχή μνήμης για το δευτερεύον ευρετήριο σε συστήματα με υψηλά ποσοστά λήξης και εκδίωξης δεδομένων.

Όπως και στη Redis, επειδή στα πλαίσια της αξιολόγησης μας θα χρησιμοποιήσουμε τη λειτουργία συμπλέγματος (Cluster Mode) που προσφέρει η Aerospike, κρίνεται σκόπιμο να αναλυθεί. Η Aerospike χρησιμοποιεί την αρχιτεκτονική Shared-Nothing. Αυτό σημαίνει ότι κάθε κόμβος του συμπλέγματος είναι ίδιος, όλοι οι κόμβοι είναι ομότιμοι (peers) και ότι δεν υπάρχει κανένα συγκεκριμένο σημείο αποτυχίας. Χρησιμοποιώντας τον αλγόριθμο Smart Partitions η Aerospike κατανέμει τα δεδομένα ομοιόμορφα σε όλους τους κόμβους του συμπλέγματος. Η κατανομή διαμερισμών στην Aerospike έχει τα ακόλουθα χαρακτηριστικά:

- Η Aerospike χρησιμοποιεί μια μέθοδο τυχαίου κατακερματισμού για να διασφαλίσει ότι οι διαμερισμοί κατανέμονται ομοιόμορφα στους κόμβους συμπλέγματος. Δεν υπάρχει ανάγκη για χειροκίνητο θρυμματισμό.
- Όλοι οι κόμβοι στο σύμπλεγμα είναι ομότιμοι - δεν υπάρχει κανένας κύριος κόμβος βάσης δεδομένων που μπορεί να αποτύχει και να σταματήσει τη λειτουργία ολόκληρης τη βάσης δεδομένων.
- Όταν προστεθούν ή αφαιρεθούν κόμβοι, θα σχηματιστεί ένα νέο σύμπλεγμα και οι κόμβοι θα συντονιστούν για να κατανείμουν ομοιόμορφα τους διαμερισμούς μεταξύ τους. Το σύμπλεγμα θα επανεξισορροπηθεί αυτόματα.

Επειδή τα δεδομένα κατανέμονται ομοιόμορφα (και τυχαία) στους κόμβους συμπλέγματος, δεν υπάρχουν "καυτά" σημεία (hot spots) ή σημεία συμφόρησης (bottlenecks) όπου ένας κόμβος χειρίζεται σημαντικά περισσότερα αιτήματα από οποιονδήποτε άλλο κόμβο. Η τυχαία εκχώρηση δεδομένων διασφαλίζει ισορροπημένο φορτίο στους διακομιστές. Για λόγους αξιοπιστίας, η Aerospike αναπαράγει τους διαμερισμούς της σε έναν ή περισσότερους κόμβους. Αυτό ρυθμίζεται παραμετροποιώντας τη μεταβλητή "replication-factor". Όταν το replication-factor είναι ίσο με ένα τότε δε δημιουργούνται ρέπλικες των δεδομένων. Όταν είναι από $n > 1$ τότε τα δεδομένα αυτά αναπαράγονται n φορές. Ένας κόμβος γίνεται ο κύριος δεδομένων για αναγνώσεις και εγγραφές για έναν διαμερισμό, ενώ άλλοι κόμβοι αποθηκεύουν τις ρέπλικες των διαμερισμών αυτών.

Ο παράγοντας αναπαραγωγής αυτός ωστόσο, δεν μπορεί να υπερβεί τον αριθμό των κόμβων στο σύμπλεγμα. Περισσότερα αντίγραφα ισοδυναμούν με καλύτερη αξιοπιστία, αλλά δημιουργούν υψηλότερη απαίτηση στο σύμπλεγμα καθώς τα αιτήματα εγγραφής πρέπει να μεταφέρονται σε όλα τα αντίγραφα. Οι περισσότερες εφαρμογές χρησιμοποιούν συντελεστή αναπαραγωγής 2 (ένα κύριο αντίγραφο και ένα απλό αντίγραφο). Η σύγχρονη αναπαραγωγή παρέχει υψηλότερο επίπεδο ορθότητας σε περίπτωση που δεν υπάρχουν σφάλματα δικτύου. Μια συναλλαγή εγγραφής διαδίδεται σε όλα τα αντίγραφα πριν από τη δέσμευση των δεδομένων και την επιστροφή των αποτελεσμάτων στον πελάτη. Σε σπάνιες περιπτώσεις κατά τη διάρκεια της

αναδιάρθρωσης του συμπλέγματος, όταν ο Aerospike Smart Client ενδέχεται να έχει στείλει το αίτημα σε λάθος κόμβο, επειδή δεν είναι ενημερωμένος για τις τελευταίες αλλαγές, τότε το Aerospike Smart Cluster μεσολαβεί με διαφάνεια ώστε το αίτημα να φτάσει στο σωστό κόμβο. Όταν ένα σύμπλεγμα αναρρώνει από ένα διαμερισμό, μπορεί να υπάρξουν εγγραφές που έχουν αποθηκευτεί σε περίοδο συγκρούσεων (conflicts) σε διαφορετικούς διαμερισμούς. Σε αυτήν την περίπτωση, η Aerospike εφαρμόζει μια ευριστική για την πιο πιθανή έκδοση, η οποία επιλύει τυχόν συγκρούσεις μεταξύ διαφορετικών αντιγράφων των δεδομένων. Από προεπιλογή, επιλέγεται η έκδοση με τον μεγαλύτερο αριθμό αλλαγών (υψηλότερος αριθμός γενεών), αν και μπορεί να επιλεγεί η έκδοση με τον πιο πρόσφατα τροποποιημένο χρόνο. Η σωστή επιλογή θα καθοριστεί από το μοντέλο δεδομένων.

Ο μηχανισμός εξισορρόπησης δεδομένων της Aerospike διασφαλίζει ότι ο όγκος των ερωτημάτων κατανέμεται ομοιόμορφα σε όλους τους κόμβους του συμπλέγματος και είναι επίμονος κατά την αποτυχία του κόμβου. Το σύστημα αυτό είναι συνεχώς διαθέσιμο. Η εξισορρόπηση δεν επηρεάζει τη συμπεριφορά του συμπλέγματος. Οι αλγόριθμοι συναλλαγών που είναι ενσωματωμένοι στο σύστημα διανομής δεδομένων διασφαλίζουν ότι υπάρχει μία συναίνεση ψήφου για το συντονισμό μιας αλλαγής στο σύμπλεγμα. Η ψηφοφορία ανά αλλαγή συμπλέγματος, αντί ανά συναλλαγή, παρέχει υψηλότερη απόδοση διατηρώντας παράλληλα την απλότητα της κοινής χρήσης ανάμεσα στους κόμβους. Η Aerospike επιτρέπει επιλογές διαμόρφωσης για να καθοριστεί το πόσο γρήγορα προχωρά η εξισορρόπηση. Η προσωρινή επιβράδυνση των συναλλαγών θεραπεύει το σύμπλεγμα πιο γρήγορα. Εάν πρέπει να διατηρηθούν η ταχύτητα και ο όγκος των συναλλαγών, το σύμπλεγμα εξισορροπείται πιο αργά. Κατά τη διάρκεια της εξισορρόπησης, η Aerospike δεν διατηρεί πλήρεις παράγοντες αναπαραγωγής όλων των διαμερισμών. Ορισμένα διαμετακομιστικά διαμερίσματα γίνονται προσωρινά μονά αντίγραφα (single replicas), για να παρέχουν τη μέγιστη διαθεσιμότητα μνήμης και αποθήκευσης καθώς το σύμπλεγμα εξισορροπείται.

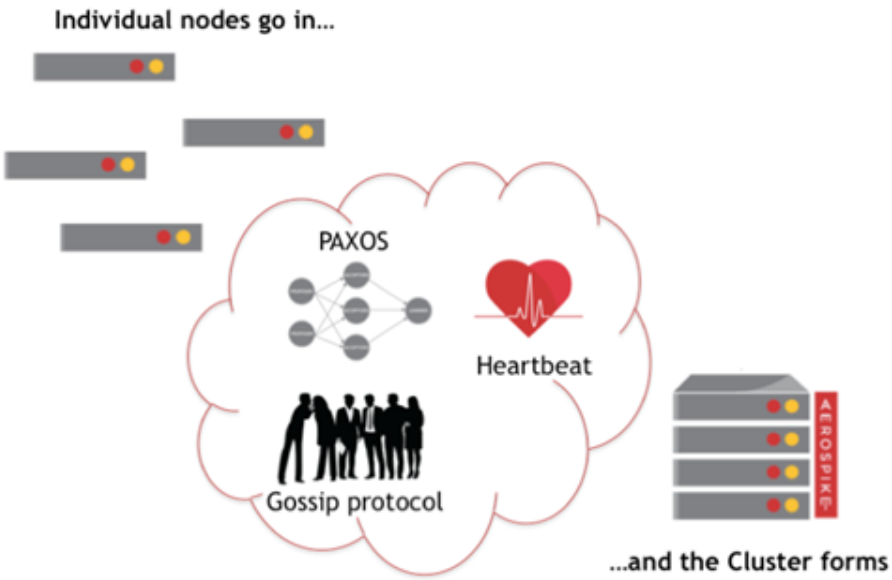
Σε περίπτωση υπερχείλισης του αποθηκευτικού χώρου, το όριο διακοπής εγγραφών της Aerospike εμποδίζει νέες εγγραφές. Η εγγραφή αντιγράφων και μετεγκατάστάσεων (migrations) , καθώς και αναγνώσεις των εγγραφών, συνεχίζουν κανονικά τη λειτουργία τους . Έτσι, ακόμη και πέρα από τη μέγιστη χωρητικότητα, η βάση δεδομένων δεν σταματά να χειρίζεται αιτήματα.

Η Aerospike μπορεί να διαμορφωθεί είτε ως Available και Partition-tolerant (AP) είτε ως Consistent και Partition-tolerant (CP) όπου τα AP και CP προέρχονται από το θεώρημα CAP. Αυτό σημαίνει ότι η Aerospike μπορεί να είναι είτε διαθέσιμη και ανθεκτική στους διαμερισμούς, είτε συνεπής και ανθεκτική στους διαμερισμούς. Στην περίπτωση AP η Aerospike χρησιμοποιεί την αναπαραγωγή δεδομένων ανάμεσα στους κόμβους του συστήματος. Η αυτόματη αναπαραγωγή δεδομένων δίνει στο σύστημα το πλεονέκτημα της υψηλής απόδοσης και της διαθεσιμότητας, αλλά στο κόστος της αυξημένης καθυστέρησης των συναλλαγών. Κατά τη διάρκεια καλών λειτουργικών συνθηκών δικτύου, το αυξημένο κόστος καθυστέρησης είναι αρκετά χαμηλό, αλλά μπορεί να είναι τροχοπέδη κατά την αναδιάρθρωση του συμπλέγματος ή κατά τη διάρκεια βλαβών στο δίκτυο, όταν όλα τα αντίγραφα ενδέχεται να μην είναι συγχρονισμένα. Όσον αφορά τη περίπτωση CP η Aerospike εγγυάται ότι όλες οι λειτουργίες εγγραφής σε ένα αρχείο θα εφαρμοστούν σε μια

συγκεκριμένη σειρά (διαδοχικά), και οι εγγραφές δεν θα αναδιαταχθούν ούτε θα παραλειφθούν. Συγκεκριμένα, οι λειτουργίες εγγραφής που αναγνωρίζονται ως δεσμευμένες θα έχουν συμβεί και θα υπάρχουν στο χρονοδιάγραμμα συναλλαγών σε αντίθεση με άλλες λειτουργίες εγγραφές στο ίδιο αρχείο. Αυτή η εγγύηση ισχύει ακόμη και σε περίπτωση βλάβης, διακοπής λειτουργίας και κατατμήσεων του δικτύου. Η ισχυρή εγγύηση συνέπειας της Aerospike είναι μόνο ανά αρχείο. Η εγγραφή ή η ενημέρωση κάθε αρχείου θα είναι ατομική και απομονωμένη και η προτεραιότητα εγγυάται χρησιμοποιώντας ένα υβριδικό ρολόι.

Η εγγύηση αξιοπιστίας της Aerospike ξεκινά με τον εντοπισμό αστοχιών στα συμπλέγματα. Σε περίπτωση αποτυχίας ενός κόμβου του συμπλέγματος ή αποτυχίας δικτύου, η Aerospike ανακτά γρήγορα και μεταρρυθμίζει το σύμπλεγμα. Για τη δημιουργία ενός συμπλέγματος, πρώτα πρέπει να εγκατασταθεί ένας κόμβος και μετά γίνεται προσθήκη όσων κόμβων χρειάζεται ανάλογα με την ανάγκη. Οι κόμβοι συμπλέγματος παρακολουθούν ο ένας τον άλλο χρησιμοποιώντας τη λειτουργία του καρδιακού παλμού (heartbeat). Υπάρχουν δύο υποστηριζόμενες λειτουργίες καρδιακού παλμού. Η Multicast η οποία χρησιμοποιεί το πρωτόκολλο UDP και η Mesh η οποία χρησιμοποιεί το πρωτόκολλο TCP. Χρησιμοποιώντας έναν καρδιακό παλμό, οι κόμβοι μπορούν να συντονιστούν. Οι κόμβοι είναι ομότιμοι. Δεν υπάρχει κύριος κόμβος σε αυτή τη λειτουργία. Όλοι οι κόμβοι παρακολουθούν τους άλλους κόμβους του συμπλέγματος. Κατά τη διαχείριση κόμβων, όλοι οι κόμβοι συμπλέγματος εντοπίζουν αλλαγές χρησιμοποιώντας τον μηχανισμό καρδιακού παλμού.

Η Aerospike προσδιορίζει ενα σύμπλεγμα χρησιμοποιώντας τις παρακάτω μεθόδους. Το Multicast χρησιμοποιεί την IP:PORT για να στείλει το μήνυμα του καρδιακού παλμού. Το Mesh χρησιμοποιεί τη διεύθυνση ενός διακομιστή της Aerospike για να ενταχθεί στο σύμπλεγμα. Μόλις η Aerospike εντοπίσει αλλαγή στο σύμπλεγμα, οι άλλοι κόμβοι χρησιμοποιούν μια διαδικασία ψηφοφορίας κουτσομπολιού που βασίζεται στο πρωτόκολλο Paxos για να προσδιορίσουν ποιοι κόμβοι σχηματίζουν το σύμπλεγμα. Ο αλγόριθμος Aerospike Smart Partitions εκχωρεί εκ νέου αυτόματα τις κατατμήσεις αυτές και εξισορροπεί το σύμπλεγμα. Ο ντετερμινιστικός αλγόριθμος κατακερματισμού χαρτογραφεί πάντα μια εγγραφή στον ίδιο διαμερισμό. Τα δεδομένα παραμένουν στον ίδιο διαμερισμό για ολόκληρη τη ζωή τους, αλλά οι διαμερισμοί μπορούν να μετακινηθούν από τον ένα διακομιστή στον άλλο. Η Aerospike χρησιμοποιεί έναν αλγόριθμο κουτσομπολιού που βασίζεται στο πρωτόκολλο Paxos για να εξασφαλίσει συμφωνία για ένα ελάχιστο ποσό κρίσιμης κοινής κατάστασης. Το πιο κρίσιμο μέρος αυτής της κοινής κατάστασης είναι η λίστα των κόμβων που συμμετέχουν στο σύμπλεγμα. Κάθε φορά που ένας κόμβος φτάνει ή αναχωρεί, ο αλγόριθμος εκτελείται για να εξασφαλίσει συμφωνία. Αυτή η διαδικασία διαρκεί ένα κλάσμα του δευτερολέπτου. Μετά από αυτήν τη φάση, κάθε κόμβος συμφωνεί τόσο στους συμμετέχοντες όσο και στη σειρά τους στο σύμπλεγμα. Χρησιμοποιώντας αυτές τις πληροφορίες, υπολογίζεται ο κύριος κόμβος για οποιαδήποτε συναλλαγή καθώς και οι κόμβοι ρέπλικα. Δεδομένου ότι οι βασικές πληροφορίες για οποιαδήποτε συναλλαγή υπολογίζονται, οι συναλλαγές μπορούν να είναι απλούστερες και να χρησιμοποιούν αποδεδειγμένους αλγόριθμους βάσης δεδομένων. Αυτό έχει ως αποτέλεσμα ελάχιστο λανθάνοντα χρόνο, καθώς εμπλέκεται μόνο ένα ελάχιστο υποσύνολο κόμβων.



Εικόνα 3 :Πως δημιουργείται ένα σύμπλεγμα στην Aerospike. Πηγή:https://aerospike.com/docs/architecture/assets/clustering.png

Η λειτουργία κατανεμημένης επεξεργασίας που προσφέρει η Aerospike διασφαλίζει την αξιοπιστία των δεδομένων. Άλλες ανταγωνιστικές βάσεις δεδομένων έχουν υψηλή απόδοση από έναν διακομιστή με μεγάλο SSD. Η Aerospike παρέχει και κάθετη κλιμακωσιμότητα, δηλαδή αρκετά εκατομμύρια συναλλαγές ανά δευτερόλεπτο ανά διακομιστή με αποθηκευτικό χώρο τύπου Flash, και δεκάδες εκατομμύρια με αποθηκευτικό χώρο τύπου DRAM. Με το σύμπλεγμα, μια βάση δεδομένων μπορεί να παρέχει τόσο οριζόντια κλιμακωσιμότητα όσο και υψηλότερη διαθεσιμότητα, καθώς τα δεδομένα αποθηκεύονται σε πολλούς διακομιστές. Εάν ένας διακομιστής αποτύχει, η πρόσβαση στη βάση δεδομένων σταματά. Η Aerospike εκτελείται σε λιγότερους διακομιστές από άλλες βάσεις δεδομένων, γεγονός που διατηρεί το κόστος χαμηλό. Η Aerospike μπορεί επίσης να διευκολύνει στο σχεδιασμό ανάπτυξης προκειμένου να επιτύχει την ελάχιστη δυνατή αδράνεια από όσο το δυνατόν λιγότερους διακομιστές. Η αρχιτεκτονική της Aerospike, όπου οι κόμβοι αυτοδιαχειρίζονται και συντονίζονται για να διασφαλίσουν την αξιοπιστία, διευκολύνει ακόμα την επέκταση καθώς αυξάνονται οι απαιτήσεις των συμπλεγμάτων. Ο ενσωματωμένος έξυπνος πελάτης της Aerospike (ο οποίος περιλαμβάνεται σε οποιαδήποτε εφαρμογή που χρησιμοποιεί το API της Aerospike), καθιστά δυνατό ότι η εφαρμογή μπορεί να αγνοήσει τη διαχείριση κόμβων και να επιτρέψει στον πελάτη να χειριστεί την επικοινωνία ανάμεσα στους κόμβους του συμπλέγματος. Η βάση δεδομένων Aerospike παρέχει εργαλεία παρακολούθησης για την παρακολούθηση της χωρητικότητας, των σημείων συμφόρησης, της διάγνωσης βλαβών υλικού και ούτω καθεξής. Τέλος τα εργαλεία παρακολούθησης της βάσης δεδομένων Aerospike επιτρέπουν την αξιολόγηση και την αναγνώριση των σημείων συμφόρησης που μειώνουν τη χωρητικότητα απόδοσης της βάσης δεδομένων, καθιστώντας τα αιτήματα αργά.

3.3 ArangoDB

Η ArangoDB [27] είναι ένα δωρεάν και ανοιχτού κώδικα εγγενές σύστημα βάσεων δεδομένων πολλαπλών μοντέλων που αναπτύχθηκε από την ArangoDB GmbH. Το σύστημα βάσης δεδομένων υποστηρίζει τρία μοντέλα δεδομένων (κλειδί / τιμή, έγγραφα, γραφήματα) με έναν πυρήνα βάσης δεδομένων και μια ενοποιημένη γλώσσα ερωτήματος AQL (ArangoDB Query Language). Η γλώσσα ερωτημάτων είναι δηλωτική και επιτρέπει τον συνδυασμό διαφορετικών προτύπων πρόσβασης δεδομένων σε ένα μόνο ερώτημα. Η ArangoDB είναι ένα σύστημα βάσης δεδομένων NoSQL, αλλά η γλώσσα AQL που παρέχει είναι αρκετά παρόμοια με πολλούς τρόπους με τη σύνταξη SQL ερωτημάτων. Η ArangoDB ξεκίνησε το 2011 και κυκλοφόρησε αρχικά με το όνομα AvocadoDB, όπου και πήρε το λογότυπό της, αλλά άλλαξε σε ArangoDB το 2012. Η βάση αυτή εντάσσεται στη κατηγορία των καθολικών βάσεων δεδομένων, αλλά οι δημιουργοί της το αναφέρουν ως "εγγενή βάση πολλαπλών μοντέλων" για να υποδείξουν ότι έχει σχεδιαστεί ειδικά για να επιτρέπουν τα δεδομένα κλειδιού / τιμής, εγγράφου και γραφήματος να μπορούν να αποθηκεύονται μαζί και όλα να έχουν τη δυνατότητα να υποβληθούν σε ερωτήματα με κοινή γλώσσα την AQL.

Η ArangoDB έχει αναπτυχθεί με τις γλώσσες προγραμματισμού C++ και JavaScript, έχει όλες τις ιδιότητες μιας ACID (atomicity, consistency, isolation, durability) βάσης δεδομένων και μπορεί να επεκταθεί τόσο οριζόντια όσο και κατακόρυφα. Η βάση αυτή χωρίζει τους τύπους δεδομένων της ως βασικούς τύπους δεδομένων και ως σύνθετους τύπους δεδομένων. Οι βασικοί τύποι δεδομένων είναι οι [8] :

- Κενή Τιμή (Null Value): Μια Κενή Τιμή μπορεί να χρησιμοποιηθεί για την αναπαράσταση μιας κενής ή απούσας τιμής. Διαφέρει από την αριθμητική τιμή μηδέν (null ! = 0) και άλλες ψευδείς τιμές (λογικό false, συμβολοσειρά μηδενικού μήκους, κενός πίνακας ή αντικείμενο). Είναι επίσης γνωστό ως nil ή None σε άλλες γλώσσες. Το σύστημα ενδέχεται να επιστρέψει Κενή Τιμή εάν δεν υπάρχει τιμή, για παράδειγμα εάν καλεστεί μια συνάρτηση με μη υποστηριζόμενες τιμές ως ορίσματα ή εάν προσπελαστεί ένα χαρακτηριστικό που δεν υπάρχει.
- Λογικός Τύπος Δεδομένων (Boolean data type): Ο Λογικός Τύπος Δεδομένων έχει δύο πιθανές τιμές, αληθές και λάθος. Αντιπροσωπεύουν τις δύο τιμές αλήθειας στη λογική και τα μαθηματικά.
- Αριθμοί (Numeric Literals): Οι αριθμοί μπορεί να είναι ακέραιοι ή να είναι πραγματικές τιμές δηλαδή αριθμοί κινητής υποδιαστολής. Μπορούν προαιρετικά να υπογραφούν με τα σύμβολα + ή - προκειμένου να υποδηλώσουν το πρόσημο του αριθμού. Ένα δεκαδικό σημείο χρησιμοποιείται ως διαχωριστικό για το προαιρετικό κλασματικό μέρος και συμβολίζεται με την τελεία “.”. Υποστηρίζεται επίσης η επιστημονική σημειογραφία (E-notation και e-notation). Όλες οι αριθμητικές τιμές αντιμετωπίζονται ως τιμές διπλής ακρίβειας 64-bit εσωτερικά. Η εσωτερική μορφή που χρησιμοποιείται είναι το IEEE 754.
- Συμβολοσειρές (String Literals): Οι συμβολοσειρές πρέπει να περικλείονται σε μονά ή διπλά εισαγωγικά. Εάν ο χρησιμοποιούμενος χαρακτήρας προσφοράς πρέπει να χρησιμοποιηθεί ο ίδιος

μέσα στην κυριολεκτική συμβολοσειρά, πρέπει να “ξεφύγει” (escaped) τότε χρησιμοποιείται το σύμβολο backslash. Μια κυριολεκτική χρησιμοποίηση του συμβόλου backslash πρέπει και αυτή να “ξεφύγει” χρησιμοποιώντας ένα άλλο backslash. Όλες οι συμβολοσειρές πρέπει να είναι κωδικοποιημένες με κώδικα UTF-8. Προς το παρόν δεν είναι δυνατή η χρήση αυθαίρετων δυαδικών δεδομένων εάν δεν είναι κωδικοποιημένα σε UTF-8. Μια λύση για τη χρήση δυαδικών δεδομένων είναι η κωδικοποίηση των δεδομένων χρησιμοποιώντας το Base64 ή άλλους αλγόριθμους στην πλευρά της εφαρμογής πριν την αποθήκευση και την αποκωδικοποίησή τους από την πλευρά της εφαρμογής μετά την ανάκτηση. Το Base64 είναι μια ομάδα σχημάτων κωδικοποίησης δυαδικού κειμένου που αντιπροσωπεύουν δυαδικά δεδομένα σε μορφή συμβολοσειράς ASCII μεταφράζοντάς τα σε παράσταση radix-64.

Πέρα από τους βασικού τύπους η ArangoDB προσφέρει και σύνθετους τύπους. Αυτοί είναι:

- Πίνακες / Λίστες (Arrays / Lists): Ο πρώτος υποστηριζόμενος σύνθετος τύπος είναι ο τύπος του πίνακα. Οι πίνακες είναι ουσιαστικά ακολουθίες ανώνυμων τιμών. Μεμονωμένα στοιχεία πίνακα μπορούν να προσπελαστούν με βάση τις θέσεις τους. Η σειρά των στοιχείων σε έναν πίνακα είναι σημαντική. Μια δήλωση πίνακα ξεκινά με ένα αριστερό τετράγωνο αγκύλη “[“ και τελειώνει με ένα δεξί τετράγωνο αγκύλη “]”. Η δήλωση περιέχει μηδέν, μία ή περισσότερες εκφράσεις, διαχωρισμένες μεταξύ τους με το σύμβολο κόμμα “,”. Το κενό διάστημα γύρω από τα στοιχεία αγνοείται στη δήλωση, έτσι ώστε οι αλλαγές γραμμής, οι καρτέλες διακοπής (tab stops) και τα κενά μπορούν να χρησιμοποιηθούν για τη μορφοποίησή του. Στην ευκολότερη περίπτωση, ένας πίνακας είναι άδειος και έτσι η δήλωση του μοιάζει με το “[]”. Τα στοιχεία ενός πίνακα μπορούν να περιέχουν κάθε τύπο δεδομένων όπως αριθμούς, λογικές τιμές, συμβολοσειρές ταυτόχρονα. Επίσης μπορούν να περιέχουν και άλλους εμφωλευμένους πίνακες. Το τελευταίο κόμμα “,” μετά την εισαγωγή ενός στοιχείου έχει αρχίσει να επιτρέπεται μετά την έκδοση 3.7.0 και ένας πίνακας έτσι έχει τη μορφή [a, b, c,]. Οι μεμονωμένες τιμές ενός πίνακα μπορούν αργότερα να προσπελαστούν από τις θέσεις τους χρησιμοποιώντας το εργαλείο εκτίμησης “[]”. Η θέση του προσπελάσιμου στοιχείου πρέπει να είναι αριθμητική τιμή. Οι θέσεις ξεκινούν από το 0. Είναι επίσης δυνατό να χρησιμοποιηθούν αρνητικές τιμές ευρετηρίου για την προσπέλαση των τιμών του πίνακα ξεκινώντας από το τέλος του. Αυτό είναι βολικό εάν το μήκος του πίνακα είναι άγνωστο και απαιτείται πρόσβαση σε στοιχεία στο τέλος του πίνακα. Για τις Λίστες ισχύον ακριβώς τα ίδια καθώς στην ArangoDB οι Λίστες και οι Πίνακες αποτελούν την ίδια δομή δεδομένων.
- Αντικείμενα / Έγγραφα (Objects / Documents): Ο άλλος τύπος σύνθετου υποστηριζόμενου τύπου είναι ο τύπος αντικειμένου ή διαφορετικά έγγραφο. Τα αντικείμενα είναι μια σύνθεση από μηδέν έως πολλά χαρακτηριστικά. Κάθε χαρακτηριστικό είναι ένα ζεύγος ονόματος-τιμής. Τα χαρακτηριστικά των αντικειμένων μπορούν να προσπελαστούν μεμονωμένα από τα ονόματά τους. Αυτός ο τύπος δεδομένων είναι επίσης γνωστός ως λεξικό (dictionary), χάρτης (map), συσχετιστικός πίνακας (associative array) και άλλα ονόματα. Οι δηλώσεις αντικειμένων ξεκινούν με μια αριστερή σγουρή αγκύλη “{“ και τελειώνουν με μια δεξιά σγουρή αγκύλη “}”. Ένα αντικείμενο περιέχει μηδενικές έως πολλές δηλώσεις χαρακτηριστικών, διαχωρισμένες μεταξύ

τους με το σύμβολο “,”. Το κενό διάστημα γύρω από τα στοιχεία αγνοείται στη δήλωση, έτσι ώστε οι αλλαγές γραμμής, οι καρτέλες διακοπής (tab stops) και τα κενά μπορούν να χρησιμοποιηθούν για τη μορφοποίηση. Στην απλούστερη περίπτωση, ένα αντικείμενο είναι κενό. Η δήλωσή του θα είναι τότε η “{}”. Κάθε χαρακτηριστικό σε ένα αντικείμενο είναι ένα ζεύγος ονόματος / τιμής. Το όνομα και η τιμή ενός χαρακτηριστικού διαχωρίζονται χρησιμοποιώντας το σύμβολο άνω και κάτω τελείας “:”. Το όνομα είναι πάντα μια συμβολοσειρά, ενώ η τιμή μπορεί να είναι οποιουδήποτε τύπου, συμπεριλαμβανομένων των υπο-αντικειμένων. Το όνομα του χαρακτηριστικού είναι υποχρεωτικό - δεν μπορεί να υπάρχουν ανώνυμες τιμές σε ένα αντικείμενο. Το όνομα μπορεί να καθοριστεί ως συμβολοσειρά χωρίς εισαγωγικά, με μονά εισαγωγικά ή με διπλά εισαγωγικά. Πρέπει να αναφέρεται εάν περιέχει κενό διάστημα, ακολουθίες διαφυγής (escape sequences) ή χαρακτήρες διαφορετικούς από τα γράμματα ASCII (a-z, A-Z), τα ψηφία (0-9), τις κάτω παύλες (_) και τα σύμβολα δολαρίου ($). Ο πρώτος χαρακτήρας πρέπει πάντα να είναι γράμμα, υπογράμμιση ή σύμβολο δολαρίου. Εάν χρησιμοποιηθεί μια προσημασμένη λέξη (FOR, RETURN,…) σαν όνομα χαρακτηριστικού τότε πρέπει να μπει ανάμεσα από μονά εισαγωγικά, είτε διπλά εισαγωγικά, είτα μονα ticks “ ´ ” είτε ανάστροφα ticks “ ` ” (backticks). Το τελευταίο κόμμα “,” μετά την εισαγωγή ενός στοιχείου έχει αρχίσει να επιτρέπεται μετά την έκδοση 3.7.0 και ένας πίνακας έτσι έχει τη μορφή { "a" : 1 , "b" : 2, "a" : 3, }. Τα ονόματα χαρακτηριστικών μπορούν επίσης να υπολογιστούν χρησιμοποιώντας δυναμικές εκφράσεις. Για να αποσαφηνιστούν τα κανονικά ονόματα χαρακτηριστικών από τις εκφράσεις ονομάτων χαρακτηριστικών, τα υπολογισμένα ονόματα χαρακτηριστικών πρέπει να περικλείονται σε αγκύλες […]. Οποιαδήποτε έγκυρη έκφραση μπορεί να χρησιμοποιηθεί ως τιμή χαρακτηριστικού. Αυτό σημαίνει επίσης ότι τα ένθετα αντικείμενα μπορούν να χρησιμοποιηθούν ως τιμές χαρακτηριστικών. Τα μεμονωμένα χαρακτηριστικά αντικειμένων μπορούν να προσπελαστούν από τα ονόματά τους χρησιμοποιώντας την τελεία “.” σαν λειτουργία εκτίμησης ως εξής: “u.address” όπου address είναι ένα χαρακτηριστικό του αντικειμένου “u” . Τα μεμονωμένα χαρακτηριστικά αντικειμένων μπορούν επίσης να προσπελαστούν από τα ονόματά τους χρησιμοποιώντας χρησιμοποιώντας την τετραγωνισμένη αγκύλη, όπως για παράδειγμα u["address"].

Πριν προχωρήσουμε στην ανάλυση των ευρετηρίων έχει νόημα να αναφέρουμε ότι στο κάτω μέρος του συστήματος της βάσης δεδομένων της ArangoDB βρίσκεται η μηχανή αποθήκευσης των δεδομένων. Η μηχανή αποθήκευσης είναι υπεύθυνη για τη διατήρηση των εγγράφων στο δίσκο, τη διατήρηση αντιγράφων στη μνήμη, παρέχοντας ευρετήρια και προσωρινή μνήμη για την επιτάχυνση των ερωτημάτων.

Μέχρι την έκδοση 3.1, η ArangoDB υποστηρίζει μόνο αρχεία που έχουν αντιστοιχιστεί στη μνήμη (memory-mapped files η αλλιώς MMFiles) ως τη μοναδική μηχανή αποθήκευσης. Στην έκδοση 3.2, άρχισε να υποστηρίζει μηχανές αποθήκευσης με δυνατότητα σύνδεσης και έτσι προστέθηκε μια δεύτερη μηχανή με βάση τη RocksDB [28] που δημιουργήθηκε από το Facebook. Τα MMFiles παρέμειναν η προεπιλεγμένη μηχανή για την έκδοση 3.3, αλλά στην 3.4 η RocksDB έγινε η νέα προεπιλεγμένη μηχανή. Στην έκδοση 3.6 η ArangoDB σταμάτησε την υποστήριξη των MMFiles και στην έκδοση 3.7.0 καταργήθηκαν εντελώς.

Όπως και σε άλλα συστήματα βάσεων δεδομένων, τα ευρετήρια μπορούν να χρησιμοποιηθούν στην ArangoDB για να επιταχύνουν τα ερωτήματα ανάκτησης δεδομένων. Η ρύθμιση των ευρετηρίων με τον σωστό τρόπο είναι απαραίτητη για την καλή απόδοση του κάθε ερωτήματος. Η ArangoDB παρέχει αρκετούς διαφορετικούς τύπους ευρετηρίου, οι οποίοι έχουν διαφορές ανάλογα με τη μηχανή αποθήκευσης που χρησιμοποιείται. Κάθε ευρετήριο συνοδεύεται και από μια μοναδική τιμή (index handle) η οποία προσδιορίζει μοναδικά ένα ευρετήριο στη βάση δεδομένων. Είναι μια συμβολοσειρά και αποτελείται από ένα όνομα συλλογής και ένα αναγνωριστικό ευρετηρίου διαχωρισμένο με τον χαρακτήρα “ / ”. Εάν το ευρετήριο είναι μοναδικά δηλωμένο, τότε η πρόσβαση στα ευρετηριασμένα χαρακτηριστικά είναι γρήγορη. Η απόδοση υποβαθμίζεται εάν τα ευρετηριασμένα χαρακτηριστικά περιέχουν πολύ λίγες διαφορετικές τιμές. Οι τύποι ευρετηρίου που υποστηρίζει η ArangoDB είναι οι παρακάτω:

- Πρωτεύον Ευρετήριο (Primary Index): Κάθε συλλογή που δημιουργείται στην ArangoDB θα έχει τουλάχιστον το λεγόμενο Πρωτεύον Ευρετήριο. Το ευρετήριο αυτό θα δημιουργηθεί αυτόματα κατά τη δημιουργία μιας συλλογής και δεν μπορεί να αφαιρεθεί ή να αλλάξει, ούτε μπορεί να δημιουργηθεί ρητά. Η κύρια ευθύνη αυτού του ευρετηρίου είναι η ευρετηρίαση των τιμών του χαρακτηριστικού “_key” (κλειδιού) της συλλογής και η διασφάλιση ότι τα κλειδιά εγγράφου στη συλλογή είναι πραγματικά μοναδικά. Το Πρωτεύον Ευρετήριο χρησιμοποιείται για αναζητήσεις με βάση το χαρακτηριστικό “_key” ή το χαρακτηριστικό “_id ” της συλλογής, αλλά μόνο για ερωτήματα αναζήτησης ισότητας. Επίσης χρησιμοποιείται για λειτουργίες που ενημερώνουν, αντικαθιστούν ή καταργούν έγγραφα με βάση το πρωταρχικό τους κλειδί. Δεν θα χρησιμοποιηθεί για ερωτήματα εύρους ή εργασίες ταξινόμησης. Η εσωτερική αναπαράσταση του πρωτεύοντος ευρετηρίου είναι συγκεκριμένη ανάλογα με τη μηχανή αποθήκευσης που θα χρησιμοποιηθεί. Για τη μηχανή αποθήκευσης MMFiles, το Πρωτεύον Ευρετήριο είναι ένα ευρετήριο που αποθηκεύεται στη μνήμη. Οι αναζητήσεις στο Πρωτεύον Ευρετήριο έχουν πολυπλοκότητα O(1) για τη μηχανή MMFiles. Για τη μηχανή αποθήκευσης RocksDB, το Πρωτεύον Ευρετήριο είναι ένα ταξινομημένο ευρετήριο στο δίσκο εσωτερικά, οπότε θα έχει χειρότερη απόδοση από το O(1).
- Ευρετήριο Άκρων (Edge Index): Κάθε συλλογή ακμών που δημιουργείται στην ArangoDB θα έχει επίσης Ευρετήριο Άκρων αυτόματα. Το Ευρετήριο Άκρων είναι υπεύθυνο για την ευρετηρίαση των τιμών “_from” και “_to”, προκειμένου να υποστηρίξει τη γρήγορη αναζήτηση των συνδεδεμένων εξερχόμενων ή εισερχόμενων ακμών για τη διάσχιση των γραφημάτων. Τα Ευρετήρια Άκρων μιας συλλογής δεν μπορούν να αφαιρεθούν ή να αλλάξουν, ούτε μπορούν να δημιουργηθούν ρητά. Τα ευρετήρια άκρων δεν είναι μοναδικά, που σημαίνει ότι μπορεί να υπάρχουν πολλαπλές άκρες συνδεδεμένες στο ίδιο έγγραφο από προεπιλογή. Το ευρετήριο αυτό μπορεί να χρησιμοποιηθεί από ερωτήματα που αναζητούν είτε με τον όρο “_from” είτε με τον όρο “_to”. Και πάλι, αυτός ο τύπος ευρετηρίου μπορεί να χρησιμοποιηθεί μόνο για αναζήτηση ισότητας, αλλά όχι για ερωτήματα εύρους. Και πάλι, η εσωτερική αναπαράσταση ενός ευρετηρίου άκρου εξαρτάται από τη μηχανή αποθήκευσης που χρησιμοποιείται. Για τη μηχανή αποθήκευσης MMFiles, το Ευρετήριο Άκρων είναι ένα ευρετήριο στη μνήμη. Οι αναζητήσεις στο ευρετήριο της μηχανής MMFiles είναι επομένως πολύ αποδοτικές. Για τη μηχανή αποθήκευσης RocksDB, το

Ευρετήριο Άκρων είναι ένα ταξινομημένο ευρετήριο δίσκου, αλλά θα έχει μία αυτόματη κατακερματισμένη προσωρινή μνήμη (hash cache) μπροστά του που θα αποθηκεύει τα έγγραφα που είναι συνδεδεμένα σε κάθε άκρη στη μνήμη. Θα πρέπει, επομένως, να είναι σχετικά αποτελεσματικό μετά την αρχική "προθέρμανση" της προσωρινής μνήμης (warmup cache). Με τη μηχανή RocksDB, το Ευρετήριο Άκρων μπορεί επίσης να χρησιμοποιηθεί για ταξινόμηση με βάση το "_from" ή με βάση το "_to" για συγκεκριμένα είδη ερωτημάτων.

- Ευρετήριο Κατακερματισμού (Hash Index): Σε αντίθεση με τα ευρετήρια που έχουμε δει μέχρι τώρα, αυτός ο τύπος ευρετηρίου μπορεί να δημιουργηθεί κατ 'απαίτηση από τους τελικούς χρήστες. Ένα Ευρετήριο Κατακερματισμού μπορεί να δημιουργηθεί με βάση ένα μόνο χαρακτηριστικό ή με βάση πολλά χαρακτηριστικά ταυτόχρονα, ανάλογα με τη χρήση. Η δημιουργία ευρετηρίου σε ένα μοναδικό χαρακτηριστικό δεν είναι απαραίτητη: τότε το ευρετήριο θα υποστηρίζει μόνο ερωτήματα που αναζητούν αυτό το συγκεκριμένο χαρακτηριστικό. Ωστόσο, όταν ένα ευρετήριο καλύπτει πολλά χαρακτηριστικά (το λεγόμενο συνδυασμένο ευρετήριο η αλλιώς Combined Index), οι κανόνες γίνονται λίγο πιο περίπλοκοι. Και πάλι, οι κανόνες είναι ειδικά σχεδιασμένοι ανάλογα με τη χρήση της κάθε μηχανή αποθήκευσης. Για τη μηχανή MMFiles, ένα Ευρετήριο Κατακερματισμού είναι ένα πραγματικό ευρετήριο κατακερματισμού στη μνήμη, το οποίο ξαναχτίζεται στη μνήμη από τα δεδομένα του δίσκου όταν φορτώνεται μια συλλογή. Δεδομένου ότι είναι ένα ευρετήριο κατακερματισμού, κάθε ερώτημα πρέπει να κάνει αναζήτηση ισότητας για όλα τα χαρακτηριστικά ευρετηρίου για να κάνει χρήση του ευρετηρίου. Οι αναζητήσεις εύρους δεν υποστηρίζονται για Ευρετήριο Κατακερματισμού MMFiles, ούτε μπορεί να χρησιμοποιηθεί για ταξινόμηση. Κατά τη δημιουργία ενός ευρετηρίου σε πολλά χαρακτηριστικά, ένα ερώτημα πρέπει να κάνει αναζήτηση ισότητας σε όλα τα χαρακτηριστικά αυτού ευρετηρίου προκειμένου να χρησιμοποιηθεί αυτό το ευρετήριο. Με τη μηχανή RocksDB, ένα Ευρετήριο Κατακερματισμού εσωτερικά είναι και πάλι ένα ταξινομημένο ευρετήριο στο δίσκο. Το όνομα "Ευρετήριο Κατακερματισμού" ("Hash Index") διατηρήθηκε εδώ κυρίως για λόγους συμβατότητας. Ένα Ευρετήριο Κατακερματισμού στο RocksDB θα χρησιμοποιηθεί για ερωτήματα που αναζητούν ισότητα στα χαρακτηριστικά ευρετηρίου, εφόσον όλα τα χαρακτηριστικά ευρετηρίου καλύπτονται από το ερώτημα είτε ένα αριστερό πρόθεμα των χαρακτηριστικών ευρετηρίου καλύπτεται από αυτό το ερώτημα. Το τελευταίο χαρακτηριστικό ευρετηρίου που χρησιμοποιήθηκε στο αριστερό πρόθεμα μπορεί επίσης να χρησιμοποιηθεί για αναζήτηση εύρους. Ένα Ευρετήριο Κατακερματισμού μπορεί επίσης να χρησιμοποιηθεί και για τη ταξινόμηση των χαρακτηριστικών ευρετηρίου με τη μηχανή RocksDB, υπό την προϋπόθεση ότι ο όρος SORT χρησιμοποιεί την ίδια κατεύθυνση (είτε όλα σε αύξουσα σειρά είτε όλα σε φθίνουσα σειρά) για όλα τα χαρακτηριστικά ευρετηρίου. Και πάλι, το ευρετήριο θα υποστηρίξει την ταξινόμηση στο αριστερότερο πρόθεμα των χαρακτηριστικών ευρετηρίου. Τα Ευρετήρια Κατακερματισμού που καθορίζονται από τον χρήστη μπορεί να είναι μοναδικά ή και μη μοναδικά. Η δήλωση μοναδικού ευρετηρίου θα του δώσει ένα μικρό πλεονέκτημα στις ευρετικές λειτουργίες του ερωτηματολογίου. Εάν ένα ευρετήριο δεν μπορεί να δηλωθεί μοναδικό, το εργαλείο βελτιστοποίησης θα χρησιμοποιήσει μια μέση εκτίμηση επιλεκτικότητας για να

καθορίσει πόσα έγγραφα θα επιστραφούν κατά μέσο όρο από το ευρετήριο. Αυτή η εκτίμηση επιλεκτικότητας θα χρησιμοποιηθεί από το εργαλείο βελτιστοποίησης ερωτημάτων εφόσον υπάρχουν πολλά ευρετήρια. Ένα Ευρετήριο Κατακερματισμού που καθορίζεται από τον χρήστη μπορεί προαιρετικά να κηρυχθεί αραιό. Ένα αραιό ευρετήριο αγνοεί όλα τα έγγραφα για τα οποία τουλάχιστον ένα από τα χαρακτηριστικά ευρετηρίου δεν υπάρχει ή έχει τιμή κενού (null). Σε αυτήν την περίπτωση, ένα τέτοιο έγγραφο δεν θα συμπεριληφθεί στο ευρετήριο. Αυτό θα διατηρήσει το μέγεθος του ευρετηρίου μικρότερο σε περίπτωση που υπάρχουν πολλά έγγραφα, με τα χαρακτηριστικά ευρετηρίου είτε να μην υπάρχουν είτε να είναι κενά (null). Ωστόσο, όταν ένα ευρετήριο δηλώνεται αραιό, το εργαλείο βελτιστοποίησης ενδέχεται να μην το θεωρεί χρησιμοποιήσιμο σε όλες τις περιπτώσεις. Η δήλωση ενός αραιού ευρετηρίου το καθιστά μη ασφαλές στη χρήση σε πολλές περιπτώσεις από την οπτική γωνία της βελτιστοποίησης ερωτήματος, επομένως αυτή η επιλογή πρέπει να χρησιμοποιείται με προσοχή.

- Ευρετήριο Skiplist (Skiplist Index): Το ευρετήριο Skiplist είναι ένα ταξινομημένο ευρετήριο που μπορεί να χρησιμοποιηθεί για την εύρεση μεμονωμένων εγγράφων ή σειρές εγγράφων. Στη μηχανή MMFiles, το ευρετήριο Skiplist χρησιμοποιεί στην πραγματικότητα μια δομή δεδομένων που ονομάζεται skiplist και την αποθηκεύει στη μνήμη. Η skiplist είναι μια πιθανοτική δομή δεδομένων που επιτρέπει πολυπλοκότητα αναζήτησης O(logn) καθώς και πολυπλοκότητα εισαγωγής O(logn) εντός μιας ταξινομημένης ακολουθίας n στοιχείων. Επειδή η skiplist είναι μια δομή ταξινομημένων δεδομένων το Ευρετήριο Skiplist είναι ένας τύπος ευρετηρίου γενικού σκοπού που μπορεί να χρησιμοποιηθεί για την υποστήριξη πολλών διαφορετικών τύπων ερωτημάτων όπως αναζήτηση ισότητας, σάρωση εύρους και ταξινόμηση. Στη μηχανή RocksDB, το Ευρετήριο Skiplist μοιράζεται την ίδια εφαρμογή με το Ευρετήριο Κατακερματισμού, οπότε όλα όσα ισχύουν για το Ευρετήριο Κατακερματισμού που βασίζεται στη μηχανή RocksDB ισχύουν και για το Ευρετήριο Skiplist που βασίζεται στη μηχανή RocksDB. Και πάλι, το όνομα "Skiplist" διατηρήθηκε μόνο για λόγους συμβατότητας. Ανεξάρτητα από την υποκείμενη μηχανή αποθήκευσης, το Ευρετήριο Skiplist χρησιμοποιείται για ερωτήματα που κάνουν αναζήτηση ισότητας στα χαρακτηριστικά ευρετηρίου, είτε όλα τα χαρακτηριστικά του ευρετηρίου καλύπτονται από το ερώτημα είτε ένα αριστερό πρόθεμα των χαρακτηριστικών του ευρετηρίου καλύπτεται από το ερώτημα. Το τελευταίο χαρακτηριστικό ευρετηρίου που χρησιμοποιήθηκε στο αριστερό πρόθεμα μπορεί επίσης να χρησιμοποιηθεί και για αναζήτηση εύρους. Επίσης μπορεί να χρησιμοποιηθεί για ταξινόμηση στα χαρακτηριστικά ευρετηρίου, υπό την προϋπόθεση ότι ο όρος SORT χρησιμοποιεί την ίδια κατεύθυνση (είτε όλα σε αύξουσα σειρα είτε όλα σε φθίνουσα σειρά) για όλα τα χαρακτηριστικά ευρετηρίου. Και πάλι, το ευρετήριο θα υποστηρίξει την ταξινόμηση στο αριστερότερο πρόθεμα των χαρακτηριστικών ευρετηρίου.
- Ευρετήριο Πλήρους Κειμένου (Fulltext Index): Το Ευρετήριο Πλήρους Κειμένου στην ArangoDB μπορεί να χρησιμοποιηθεί για να χωρίσει ένα χαρακτηριστικό κειμένου σε μεμονωμένες λέξεις, οι οποίες στη συνέχεια θα εισαχθούν όλες στο ευρετήριο για το έγγραφο που τις περιέχει. Επίσης μπορεί να χρησιμοποιηθεί για την αναζήτηση εγγράφων με βάση μεμονωμένες λέξεις ή

συνδυασμό λέξεων. Το Ευρετήριο Πλήρους Κειμένου χρησιμοποιείται μόνο σε ερωτήματα που χρησιμοποιούν τη συνάρτηση AQL FULLTEXT.

- Ανθεκτικό Ευρετήριο (Persistent Index): Το Ανθεκτικό Ευρετήριο είναι ένα ταξινομημένο ευρετήριο του οποίου το κύριο χαρακτηριστικό είναι η μονιμότητά του. Οι καταχωρήσεις ευρετηρίου γράφονται στο δίσκο όταν αποθηκεύονται ή ενημερώνονται έγγραφα. Αυτό σημαίνει ότι οι καταχωρήσεις ευρετηρίου δεν χρειάζεται να ξαναχτιστούν από τα δεδομένα συλλογής κατά την επανεκκίνηση του διακομιστή ή την αρχική φόρτωση της συλλογής. Έτσι, η χρήση Ανθεκτικών Ευρετηρίων μπορεί να μειώσει τους χρόνους φόρτωσης συλλογής. Αυτός ο τύπος ευρετηρίου μπορεί να χρησιμοποιηθεί μόνο για δευτερεύοντα ευρετήρια αυτή τη στιγμή. Αυτό σημαίνει ότι προς το παρόν δεν μπορεί να γίνει το μόνο ευρετήριο για μια συλλογή, επειδή θα υπάρχει πάντα το Πρωτεύον Ευρετήριο στη μνήμη για τη συλλογή επιπλέον, και ενδεχομένως περισσότερα ευρετήρια. Η εφαρμογή του ευρετηρίου αυτού χρησιμοποιεί τη μηχανή RocksDB και παρέχει λογαριθμική πολυπλοκότητα για εργασίες εισαγωγής, ενημέρωσης και κατάργησης. Καθώς το Ανθεκτικό Ευρετήριο δεν είναι ευρετήριο στη μνήμη, δεν αποθηκεύει δείκτες στο Πρωτεύον Ευρετήριο, όπως κάνουν όλοι τα ευρετήρια που αποθηκεύονται στη μνήμη, αλλά αποθηκεύει το πρωτεύον κλειδί ενός εγγράφου. Για την ανάκτηση ενός εγγράφου μέσω ενός Ανθεκτικού Ευρετηρίου μέσω μιας αναζήτησης αξίας ευρετηρίου, θα υπάρξει μια επιπλέον αναζήτηση σε O(1) στο Πρωτεύον Ευρετήριο για την ανάκτηση του πραγματικού εγγράφου. Καθώς το Ανθεκτικό Ευρετήριο έχει ταξινομηθεί, μπορεί να χρησιμοποιηθεί για αναζητήσεις σημείων, ερωτήματα εύρους και λειτουργίες ταξινόμησης, αλλά μόνο εάν όλα τα χαρακτηριστικά ευρετηρίου παρέχονται σε ένα ερώτημα ή εάν το αριστερότερο πρόθεμα των χαρακτηριστικών ευρετηρίου είναι καθορισμένο.
- TTL(time-to-live) Ευρετήριο (TTL Index): Ο τύπος ευρετηρίου TTL που παρέχεται από την ArangoDB μπορεί να χρησιμοποιηθεί για την αυτόματη κατάργηση εγγράφων που έχουν λήξει σε μια συλλογή. Ένα Ευρετήριο TTL δημιουργείται ρυθμίζοντας την μεταβλητή “ expireAfter ” και επιλέγοντας ένα χαρακτηριστικό εγγράφου που περιέχει την ημερομηνία και ώρα δημιουργίας των εγγράφων. Τα έγγραφα λήγουν μετά από “ expireAfter ” δευτερόλεπτα από την ώρα δημιουργίας τους. Η ώρα δημιουργίας ορίζεται είτε ως αριθμητική χρονική σήμανση (Unix timestamp) είτε ως συμβολοσειρά ημερομηνίας σε μορφή YYYY-MM-DDTHH:MM:SS, προαιρετικά με χιλιοστά του δευτερολέπτου μετά από ένα δεκαδικό σημείο στη μορφή YYYY-MM-DDTHH:MM:SS.MMM και μια προαιρετική μετατόπιση ζώνης ώρας. Όλες οι συμβολοσειρές ημερομηνιών χωρίς μετατόπιση ζώνης ώρας θα ερμηνευθούν ως ημερομηνίες UTC.
- Γεωγραφικό Ευρετήριο (Geo Index): Οι χρήστες μπορούν να δημιουργήσουν πρόσθετα γεωγραφικά ευρετήρια σε ένα ή περισσότερα χαρακτηριστικά στις συλλογές. Ένα Γεωγραφικό Ευρετήριο χρησιμοποιείται για την εύρεση θέσεων στην επιφάνεια της γης γρήγορα. Το Γεωγραφικό Ευρετήριο αποθηκεύει δισδιάστατες συντεταγμένες. Μπορεί να δημιουργηθεί είτε σε δύο ξεχωριστά χαρακτηριστικά εγγράφου (γεωγραφικό πλάτος και γεωγραφικό μήκος) είτε σε ένα χαρακτηριστικό πίνακα που περιέχει τόσο το γεωγραφικό πλάτος όσο και το γεωγραφικό μήκος. Το γεωγραφικό πλάτος και μήκος πρέπει να είναι αριθμητικές τιμές. Το Γεωγραφικό Ευρετήριο

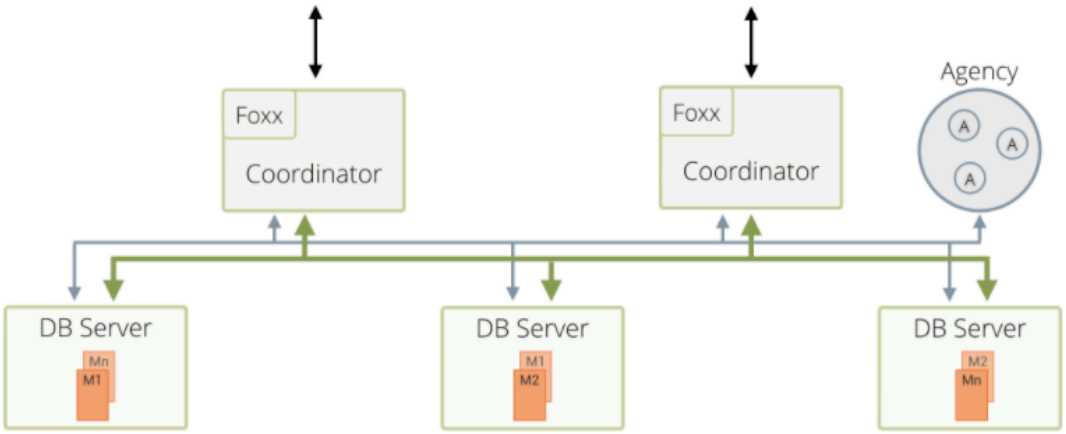
παρέχει λειτουργίες για την εύρεση εγγράφων με συντεταγμένες πλησιέστερες σε μια δεδομένη συντεταγμένη σύγκρισης όπως επίσης και για την εύρεση εγγράφων με συντεταγμένες που βρίσκονται εντός καθορισμένης ακτίνας γύρω από μια συντεταγμένη σύγκρισης. Επίσης χρησιμοποιείται και μέσω ειδικών AQL συναρτήσεων και εφαρμόζεται όταν στα AQL ερωτήματα χρησιμοποιείται η λέξη κλειδί SORT ή FILTER με τη συνάρτηση απόστασης. Διαφορετικά, δεν θα χρησιμοποιηθεί για άλλους τύπους ερωτημάτων ή συνθηκών.

Όπως και στις άλλες δύο βάσεις που αναλύθηκαν, επειδή στα πλαίσια της αξιολόγησης μας θα χρησιμοποιήσουμε τη λειτουργία συμπλέγματος (Cluster Mode) που προσφέρει η ArangoDB, κρίνεται σκόπιμο να αναλυθεί. Η αρχιτεκτονική Cluster της ArangoDB είναι ένα μοντέλο πρωτεύον/πρωτεύον κόμβου(master/master) CP χωρίς κανένα σημείο αποτυχίας. Ο όρος “ CP ” προέρχεται από το θεώρημα CAP (Consistency-Availability-Partition Tolerance) [29]. Αυτό σημαίνει ότι εν παρουσία διαμερίσματος δικτύου, η βάση δεδομένων προτιμά την εσωτερική συνέπεια από τη διαθεσιμότητα. Το μοντέλο “πρωτεύον/πρωτεύον” κόμβου σημαίνει ότι οι πελάτες μπορούν να στείλουν τα αιτήματά τους σε έναν αυθαίρετο κόμβο και να έχουν την ίδια προβολή στη βάση δεδομένων ανεξάρτητα. “Κανένα σημείο αποτυχίας” σημαίνει ότι το σύμπλεγμα μπορεί να συνεχίσει να εξυπηρετεί αιτήματα, ακόμη και όταν ένας κόμβος αποτύχει εντελώς. Με αυτόν τον τρόπο, η ArangoDB έχει σχεδιαστεί ως κατανεμημένη βάση δεδομένων πολλαπλών μοντέλων. Ένα σύμπλεγμα της ArangoDB αποτελείται από μια σειρά κόμβων της ArangoDB που μιλούν μεταξύ τους μέσω του δικτύου. Έχουν διαφορετικούς ρόλους, οι οποίοι θα εξηγηθούν λεπτομερώς παρακάτω. Η τρέχουσα διαμόρφωση του συμπλέγματος διατηρείται στο λεγόμενο “Agency”, το οποίο είναι μία πολύ διαθέσιμη και ανθεκτική αποθήκη δεδομένων τύπου κλειδιού/τιμής που βασίζεται σε ένα μονό αριθμό κόμβων ArangoDB που εκτελούν το πρωτόκολλο Raft Consensus [24]. Το Raft είναι ένας αλγόριθμος συναίνεσης που έχει σχεδιαστεί για να είναι κατανοητός. Είναι ισοδύναμο με το πρωτόκολλο Paxos σε ανοχή σφαλμάτων και απόδοση. Η διαφορά είναι ότι αποσυντίθεται σε σχετικά ανεξάρτητα υποπροβλήματα και αντιμετωπίζει καθαρά όλα τα σημαντικά κομμάτια που απαιτούνται για την ανάπτυξη πρακτικών συστημάτων.

Οι διάφοροι αυτοί κόμβοι μπορούν να πάρουν διαφορετικούς ρόλους μέσα στο σύμπλεγμα. Αυτοί είναι οι:

- Μέσα (Agents): Ένα ή περισσότερα Μέσα σχηματίζουν το Agency σε ένα σύμπλεγμα της ArangoDB. Το Agency είναι το κεντρικό μέρος για την αποθήκευση της διαμόρφωσης σε ένα σύμπλεγμα. Εκτελεί εκλογές ηγέτη και παρέχει άλλες υπηρεσίες συγχρονισμού για ολόκληρο το σύμπλεγμα. Χωρίς το Agency, κανένα από τα άλλα στοιχεία δεν μπορεί να λειτουργήσει. Αν και γενικά αόρατο στο εξωτερικό, το Agency είναι η καρδιά του συμπλέγματος. Ως εκ τούτου, η ανοχή σφαλμάτων πρέπει φυσικά να υπάρχει σαν κύριο χαρακτηριστικό του. Για να επιτευχθεί αυτό τα Μέσα χρησιμοποιούν τον αλγόριθμο Raft Consensus. Ο αλγόριθμος εγγυάται επισήμως τη διαχείριση ρυθμίσεων χωρίς συγκρούσεις εντός του συμπλέγματος της ArangoDB. Στον πυρήνα του, το Agency διαχειρίζεται ένα μεγάλο δέντρο διαμορφώσεων. Υποστηρίζει συναλλαγές ανάγνωσης και εγγραφής σε αυτό το δέντρο και άλλοι διακομιστές μπορούν να εγγραφούν σε HTTP callbacks για όλες τις αλλαγές στο δέντρο.

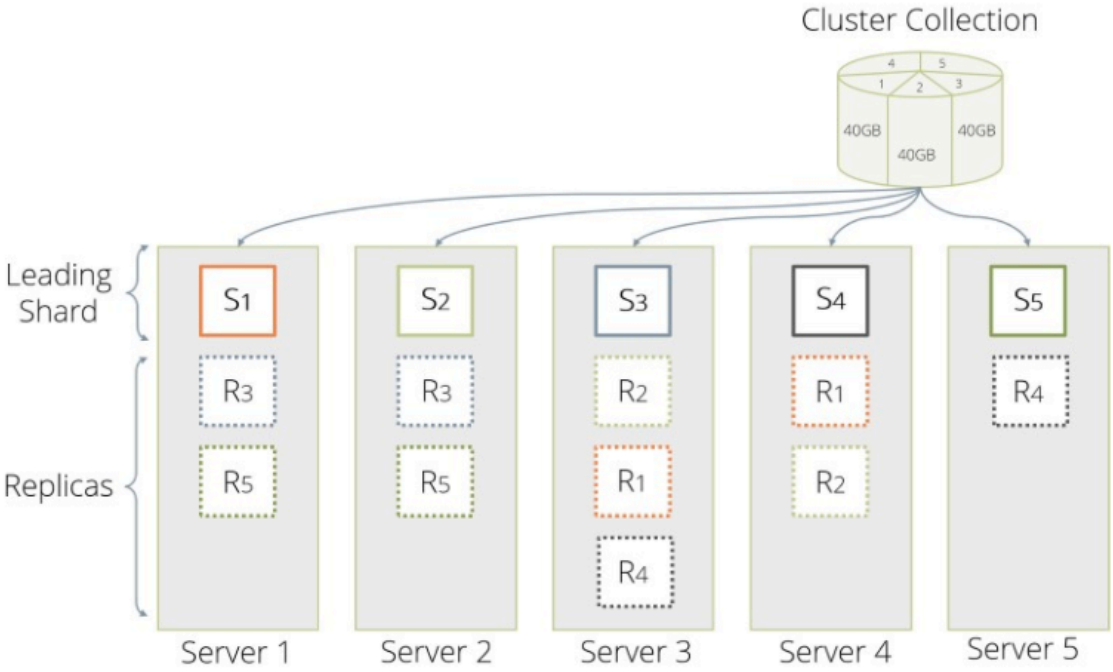
- Συντονιστές (Coordinators): Οι Συντονιστές πρέπει να είναι προσβάσιμοι εξωτερικά. Αυτοί είναι οι οποίοι επικοινωνούν με τους πελάτες. Συντονίζουν τις εργασίες του συμπλέγματος, όπως την εκτέλεση ερωτημάτων και την εκτέλεση υπηρεσιών Foxx. Οι υπηρεσίες Foxx αποτελούνται από κώδικα JavaScript που εκτελούνται στην μηχανή JavaScript V8 η οποία είναι ενσωματωμένη στην ArangoDB. Οι Συντονιστές ξέρουν που αποθηκεύονται τα δεδομένα και θα βελτιστοποιήσουν το που να εκτελεστούν τα ερωτήματα που παρέχονται από τον χρήστη ή τμήματα αυτών. Οι Συντονιστές δε διαθέτουν κατάσταση (stateless) και επομένως μπορούν εύκολα να κλείσουν και να επανεκκινηθούν ανάλογα με την περίπτωση.
- Διακομιστές DB (DB-Servers): Οι Διακομιστές DB είναι αυτοί που περιέχουν τα δεδομένα. Περιέχουν θραύσματα δεδομένων και χρησιμοποιώντας συγχρονισμό αναπαραγωγής, ένας Διακομιστής DB μπορεί να είναι ηγέτης (leader) ή οπαδός (follower) για ένα θραύσμα. Οι λειτουργίες εγγραφής εφαρμόζονται πρώτα στον ηγέτη και στη συνέχεια αναπαράγονται συγχρόνως σε όλους τους ακόλουθους. Τα θραύσματα δεν πρέπει να προσπελάσονται εξωτερικά αλλά έμμεσα μέσω των Συντονιστών. Μπορούν επίσης να εκτελέσουν ερωτήματα εν μέρει ή στο σύνολό τους όταν τους ζητηθεί από έναν Συντονιστή.



Εικόνα 4: Αρχιτεκτονική ενός συμπλέγματος της ArangoDB. Πηγή: https://www.arangodb.com/wp-content/uploads/2016/01/cluster_topology-1.png

Χρησιμοποιώντας τους ρόλους που περιγράφησαν παραπάνω, ένα σύμπλεγμα της ArangoDB μπορεί να διανείμει δεδομένα σε θραύσματα (sharding) σε πολλούς Διακομιστές DB. Ο θρυμματισμός αυτός επιτρέπει τη χρήση πολλαπλών μηχανών για την εκτέλεση ενός συμπλέγματος της ArangoDB που μαζί αποτελούν μια μοναδική βάση δεδομένων. Αυτό επιτρέπει την αποθήκευση πολύ περισσότερων δεδομένων, καθώς η ArangoDB διανέμει τα δεδομένα αυτά αυτόματα στους

διαφορετικούς διακομιστές. Έτσι η διακίνηση δεδομένων (data throughput) αυξάνεται ραγδαία, επειδή πλέον το φορτίο μπορεί να διανεμηθεί σε πολλά μηχανήματα.



Εικόνα 5: Παρουσίαση της διαδικασίας θρυμματισμού δεδομένων στην ArangoDB. Πηγή: https://www.arangodb.com/docs/stable/images/cluster_sharding.jpg

Εξωτερικά, αυτή η διαδικασία είναι απόλυτα διαφανής. Μια εφαρμογή μπορεί να μιλήσει σε οποιονδήποτε Συντονιστή και θα καταλάβει αυτόματα που είναι αποθηκευμένα τα δεδομένα (περίπτωση ανάγνωσης) ή επρόκειτο να αποθηκευτούν (περίπτωση γραφής). Οι πληροφορίες σχετικά με τα θραύσματα κοινοποιούνται σε όλους τους Συντονιστές που χρησιμοποιούν το Agency. Τα θραύσματα διαμορφώνονται ανά συλλογή, έτσι ώστε πολλά θραύσματα δεδομένων να αποτελούν τη συλλογή στο σύνολό της. Για να προσδιοριστεί σε ποια θραύσματα θα αποθηκευτούν ποια δεδομένα, η ArangoDB εκτελεί κατακερματισμό μεταξύ των τιμών. Από προεπιλογή, αυτός ο κατακερματισμός δημιουργείται χρησιμοποιώντας την τιμή της μεταβλητής “document_key”.

3.4 Αναμενόμενα Αποτελέσματα

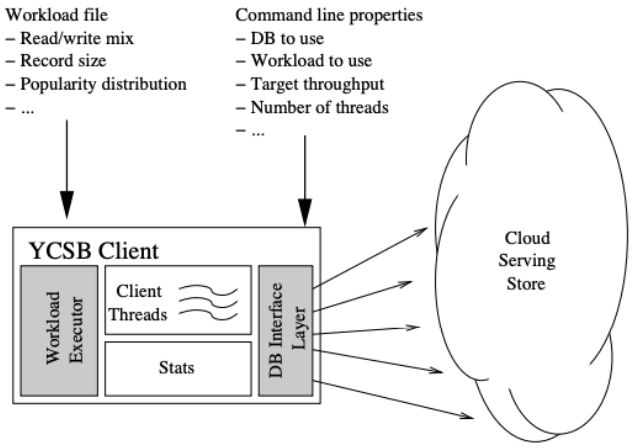
Ύστερα από τη διεξοδική αυτή ανάλυση των παραπάνω βάσεων δεδομένων μπορούμε πλέον να εξάγουμε τις αναμενόμενες συμπεριφορές που περιμένουμε από κάθε βάση. Αρχικά για την Redis αναμένουμε να είναι η πιο γρήγορα βάση από τις υπόλοιπες σε όλους τους φόρτους εργασίας. Αυτό γιατί η Redis έχει φτιαχτεί καθαρά για ταχύτητα (cache) και ώντας καθαρά βάση δεδομένων που λειτουργεί κυρίως με τη Ram. Τη δεύτερη θέση από άποψη αποδόσεων περιμένουμε να την έχει η Aerospike. Η Aerospike έχει δημιουργηθεί και αυτή για να προσφέρει μεγάλες επιδόσεις. Έχει δοθεί όμως περισσότερη σημασία στο να δουλεύει στενά με το σκληρό δίσκο (μόνο SSD) και να προσφέρει κορυφαίες αποδόσεις, παρά στο να χρησιμοποιείται καθαρά σαν μνήμη cache. Ακόμα έχει φτιαχτεί ώστε να είναι πολύ απλή για τον χρήστη και να μη χρειάζεται σχεδόν καθόλου παραμετροποίηση και αυτό πολλές φορές έχει σαν συνέπεια την ελάχιστη μείωση της απόδοσης σε σχέση με τις υπόλοιπες βάσεις δεδομένων (Redis) που θέλουν εκτενέστερη παραμετροποίηση. Τέλος έχουμε την ArangoDB η οποία περιμένουμε να έχει τη χειρότερη απόδοση σε σχέση με τις υπόλοιπες δύο. Αυτό γιατι η βάση αυτή δεν έχει σχεδιαστεί για να προσφέρει επιδόσεις, αλλά για να προσφέρει ευκολία και ελαστικότητα στο χρήστη παρέχοντας πολλά μοντέλα βάσεων δεδομέων. Αρχικά χρησιμοποιεί μια έτοιμη μηχανή βάσης δεδομένων (RocksDB) και αυτό γνωρίζουμε ότι προσθέτει ένα ακόμα επίπεδο πολυπλοκότητας (wrapper), οπότε και καθυστέρησης σε σχέση με τις άλλες δύο βάσεις δεδομένων που έχουν υλοποιήσεις τις δικές τους μηχανές βάσεων δεδομένων. Ακόμα το πρωτεύον μοντέλο της βάσης αυτής δεν είναι το μοντέλο Κλειδιού -Τιμής μόνο, αλλά υποστηρίζει συνολικά τέσσερα πρωτεύοντα μοντέλα και αυτό έχει ως συνέπεια την απώλεια απόδοσης. Επίσης ολόκληρη πλατφόρμα παρακολούθησης (monitoring) προκειμένου ο χρήστης να μπορεί κάθε στιγμή να παρακολουθεί το σύστημά του που και αυτό καταναλώνει πόρους του συστήματος με αποτέλεσμα τη μείωση της απόδοσης.

4. Μεθοδολογία Πειραμάτων

4.1 Benchmarking Suite - YCSB

Η σύγκριση των διαφόρων συστημάτων βάσεων δεδομένων είναι μια διαδικασία η οποία δεν είναι απλή. Υπάρχουν πάρα πολλές παράμετροι, διαφορετικοί για κάθε βάση δεδομένων, που πρέπει να μορφοποιηθούν ώστε να η βάση να είναι όσο το δυνατόν βέλτιστη προκειμένου τα αποτελέσματα να ανταποκρίνονται στις πραγματικές δυνατότητες του κάθε συστήματος. Επίσης οι λειτουργίες ανάμεσα στις βάσεις δεδομενων διαφέρουν και είναι δύσκολο να συγκριθούν αντικειμενικά. Έχοντας αυτό ως σκοπό αποφασίζουμε να χρησιμοποιήσουμε το εργαλείο YCSB το οποίο έχει δημιουργηθεί από τους Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan και Russell Sears κάτω από την αιγίδα της Yahoo! για να απλουστεύσει ακριβώς αυτή τη δυσκολία παρέχοντας μια πλατφόρμα η οποία υποστηρίζει πολλά συστήματα βάσεων δεδομένων. Αυτό που θα χρησιμοποιήσουμε εμείς είναι ουσιαστικά το YCSB Client, για την εκτέλεση των τεστ που παρέχει το YCSB. Ένας βασικός στόχος σχεδιασμού του εργαλείου είναι η επεκτασιμότητα, έτσι ώστε να μπορεί να χρησιμοποιηθεί για τη συγκριτική αξιολόγηση νέων συστημάτων βάσης δεδομένων cloud και για την ανάπτυξη νέων φόρτων εργασίας. Αυτό το εργαλείο είναι επίσης διαθέσιμο με άδεια χρήσης ανοιχτού κώδικα, έτσι ώστε άλλοι χρήστες να μπορούν να χρησιμοποιούν και να επεκτείνουν το εργαλείο και να συνεισφέρουν νέα φορτία εργασίας και νέες διεπαφές βάσεων δεδομένων.

Το YCSB Client είναι ένα πρόγραμμα γραμμένο στη προγραμματιστική γλώσσα Java για τη δημιουργία των δεδομένων που θα φορτωθούν στη βάση δεδομένων και για τη δημιουργία των λειτουργιών που αποτελούν τον φόρτο εργασίας. Η αρχιτεκτονική του φαίνεται στην παρακάτω εικόνα. Η βασική λειτουργία είναι ότι ο εκτελεστής φόρτου εργασίας οδηγεί πολλά νήματα πελατών. Κάθε νήμα εκτελεί μια διαδοχική σειρά λειτουργιών πραγματοποιώντας κλήσεις στο επίπεδο διεπαφής βάσης δεδομένων, τόσο για τη φόρτωση της βάσης δεδομένων (φάση φόρτωσης) όσο και για την εκτέλεση του φόρτου εργασίας (η φάση δοσοληψιών). Τα νήματα επιταχύνουν τον ρυθμό με τον οποίο δημιουργούν αιτήματα, έτσι ώστε να μπορεί να ελεγχθεί άμεσα το προσφερόμενο φορτίο έναντι της βάσης δεδομένων. Τα νήματα μετρούν επίσης το χρόνο καθυστέρησης εκτέλεσης των εντολών και επιτυγχάνουν και αναφέρουν αυτές τις μετρήσεις στη μονάδα στατιστικών στοιχείων.



Εικόνα 6: Αρχιτεκτονική του YCSB Client Πηγή: [10]

Ο Client παίρνει μια σειρά ιδιοτήτων (ζεύγη ονόματος / τιμής) που ορίζουν τη λειτουργία του. Κατά συνθήκη, διαιρούμε αυτές τις ιδιότητες σε δύο ομάδες:

- Ιδιότητες φόρτου εργασίας (Workload properties): Είναι οι ιδιότητες που ορίζουν το φόρτο εργασίας, ανεξάρτητα από μια δεδομένη βάση δεδομένων ή πειραματική εκτέλεση. Για παράδειγμα, ο συνδυασμός ανάγνωσης / εγγραφής της βάσης δεδομένων, η διανομή προς χρήση, το μέγεθος και ο αριθμός των πεδίων σε μια εγγραφή.
- Ιδιότητες χρόνου εκτέλεσης (Runtime properties): Ιδιότητες συγκεκριμένες για ένα συγκεκριμένο πείραμα. Για παράδειγμα, το επίπεδο διεπαφής βάσης δεδομένων που θα χρησιμοποιηθεί (π.χ. Redis, Aerospike κ.λπ.), τις ιδιότητες που χρησιμοποιούνται για την προετοιμασία αυτού του επιπέδου (όπως τα ονόματα κεντρικών υπολογιστών της υπηρεσίας βάσης δεδομένων), τον αριθμό των νημάτων πελατών κ.λπ.

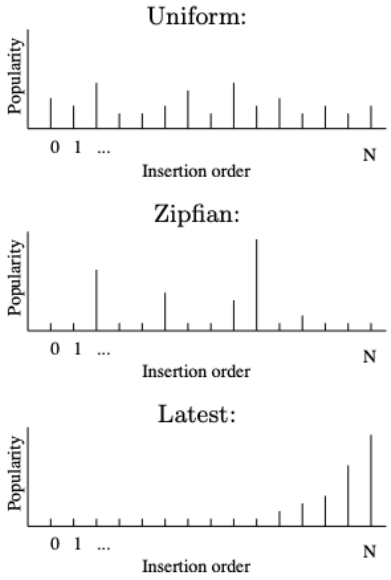
Έτσι, μπορεί να υπάρχουν αρχεία φόρτου εργασίας που παραμένουν στατικά και χρησιμοποιούνται για την αξιολόγηση μιας ποικιλίας βάσεων δεδομένων. Σε αντίθεση, οι ιδιότητες χρόνου εκτέλεσης, ενώ ενδέχεται επίσης να αποθηκευτούν σε αρχεία ιδιοτήτων, θα διαφέρουν από πείραμα σε πείραμα, καθώς η βάση δεδομένων, η απόδοση στόχου κλπ αλλάζουν. Πρωταρχικός στόχος του YCSB είναι η δυνατότητα επέκτασής του. Τα σκιασμένα κουτιά στο σχήμα 1 δείχνουν τα κομμάτια που μπορούν εύκολα να αντικατασταθούν. Το Workload Executor περιέχει κώδικα για την εκτέλεση τόσο των φορτίων όσο και των φάσεων συναλλαγής του φόρτου εργασίας. Το πακέτο YCSB περιλαμβάνει κάποιους βασικούς φόρτους εργασίας οι οποίοι ονομάζονται CoreWorkloads, τα οποία θα αναλύσουμε στη συνέχεια. Οι χρήστες του YCSB επίσης μπορούν να ορίσουν νέα πακέτα με δύο τρόπους. Το πιο απλό είναι να οριστεί ένα σύνολο φορτίων εργασίας που χρησιμοποιούν τα CoreWorkloads αλλά καθορίζουν διαφορετικές παραμέτρους φόρτου εργασίας. Αυτό επιτρέπει στους χρήστες να μεταβάλλουν διάφορους άξονες του βασικού πακέτου: ποια λειτουργία να εκτελεστεί, η ανατροπή στη δημοτικότητα των εγγραφών και το μέγεθος και τον αριθμό των εγγραφών. Η δεύτερη προσέγγιση είναι να οριστεί μια νέα κατηγορία φόρτου εργασίας (π.χ.,

γράφοντας Java) και σχετικές παραμέτρους. Αυτή η προσέγγιση επιτρέπει την εισαγωγή πιο περίπλοκων λειτουργιών και την εξερεύνηση διαφορετικών αντισταθμίσεων από ότι το βασικό πακέτο. αλλά απαιτεί μεγαλύτερη προσπάθεια σε σύγκριση με την προηγούμενη προσέγγιση. Το Επίπεδο Διεπαφής Βάσης Δεδομένων μεταφράζει απλά αιτήματα (όπως λειτουργίες read()) από τα νήματα του πελάτη σε κλήσεις ενάντια στη βάση δεδομένων. Οι κλάσεις επιπέδου διεπαφής φόρτου εργασίας και διεπαφής βάσης δεδομένων που θα χρησιμοποιηθούν για ένα πείραμα καθορίζονται ως ιδιότητες και αυτές οι κλάσεις φορτώνονται δυναμικά κατά την εκκίνηση του προγράμματος. Φυσικά, ως πακέτο ανοιχτού κώδικα, οποιοδήποτε μπορεί να αντικατασταθεί η τάξη στο εργαλείο YCSB, αλλά το Workload Executor και το Database Interface Layer μπορούν να αντικατασταθούν πιο εύκολα.

4.2 Περιγραφή μεθοδολογίας

Πριν περάσουμε στην ανάλυση των φόρτων εργασίας θα αναλύσουμε αρχικά τους τρόπους που χρησιμοποιεί το YCSB για να δημιουργήσει αυτά τα δεδομένα [10]. Τα δεδομένα δημιουργούνται χρησιμοποιώντας μία απο τις παρακάτω μεθόδους τυχαίων κατανομών.

- Ομοιόμορφη Κατανομή (Uniform): Επιλογή ενός στοιχείου ομοιόμορφα τυχαία. Για παράδειγμα, κατά την επιλογή μιας εγγραφής, όλες οι εγγραφές στη βάση δεδομένων είναι πιθανό να επιλεγούν.
- Zipfian: Επιλογή ενός στοιχείου σύμφωνα με τη κατανομή Zipfian. Για παράδειγμα, κατά την επιλογή μιας εγγραφής, ορισμένες εγγραφές θα είναι εξαιρετικά δημοφιλείς (ο επικεφαλής της διανομής), ενώ οι περισσότερες εγγραφές θα είναι μη δημοφιλείς (η ουρά).
- Πιο Πρόσφατη Κατανομή (Latest): Όπως η κατανομή Zipfian με τη μόνη διαφορά ότι τα στοιχεία που εισάχθηκαν πιο πρόσφατα βρίσκονται στην αρχή της κατανομής.
- Πολυωνυμική Κατανομή (Multinomial): Οι πιθανότητες για κάθε στοιχείο μπορούν να καθοριστούν. Για παράδειγμα, ενδέχεται να εκχωρήσουμε μια πιθανότητα 0,95 στη λειτουργία ανάγνωσης, μια πιθανότητα 0,05 στη λειτουργία ενημέρωσης και μια πιθανότητα 0 για σάρωση και εισαγωγή. Το παραπάνω αποτέλεσμα θα ήταν ένας φόρτος εργασίας ο οποίος θα είχε πολλές αναγνώσεις στοιχείων.



Εικόνα 7: Κατανομές πιθανότητας.Οι οριζόντιοι άξονες αντιπροσωπεύουν τη σειρά εισαγωγής των στοιχείων και οι κάθετοι άξονες αντιπροσωπεύουν την πιθανότητα επιλογής τους. Πηγή: [10]

Μια βασική διαφορά μεταξύ της Πιο Πρόσφατης και της Zipfian κατανομής είναι η συμπεριφορά τους όταν εισάγονται νέα αντικείμενα. Σύμφωνα με την Πιο Πρόσφατη κατανομή, το

αντικείμενο που εισήχθη πρόσφατα γίνεται το πιο δημοφιλές, ενώ τα προηγούμενα δημοφιλή στοιχεία γίνονται λιγότερο δημοφιλή. Κάτω από τη κατανομή Zipfian, τα στοιχεία διατηρούν τη δημοτικότητά τους, ακόμη και όταν εισάγονται νέα αντικείμενα, ανεξάρτητα από το αν το νέο στοιχείο που έχει εισαχθεί είναι δημοφιλές. Η Πιο Πρόσφατη κατανομή προορίζεται για τη μοντελοποίηση εφαρμογών όπου η πρόσφατη εισαγωγή έχει σημασία όπως για παράδειγμα, σε διάφορα blogs και σε διάφορες ιστοσελίδες ειδήσεων όπου η δημοτικότητα μειώνεται γρήγορα. Σε αντίθεση, το Zipfian μοντέλο κατανομής του οποίου η δημοτικότητα είναι ανεξάρτητη από την εισαγωγή νέου στοιχείου, όπως για παράδειγμα ένας χρήστης ο οποίος έχει εκατομμύρια ακόλουθους στο Instagram είναι πιο δημοφιλής από ένα χρήστη που έκανε εγγραφή πρόσφατα και δεν έχει πολλούς ακολούθους, παρόλο η εγγραφή του έγινε αρκετά χρόνια πριν.

Μια απροσδόκητα περίπλοκη πτυχή της εφαρμογής του εργαλείου YCSB αφορούσε την εφαρμογή των διανομών Zipfian και Latest. Συγκεκριμένα, χρησιμοποιήθηκε ο αλγόριθμος Gray για τη δημιουργία μιας ακολουθίας κατανεμημένης από Zipfian. Ωστόσο, αυτός ο αλγόριθμος έπρεπε να τροποποιηθεί με διάφορους τρόπους για να χρησιμοποιηθεί στο εργαλείο αυτό. Το πρώτο πρόβλημα είναι ότι τα δημοφιλή αντικείμενα συγκεντρώνονται μονομερώς. Συγκεκριμένα, το πιο δημοφιλές αντικείμενο είναι το στοιχείο 0, το δεύτερο πιο δημοφιλές αντικείμενο είναι το στοιχείο 1 και ούτω καθεξής. Για τη κατανομή Zipfian, τα δημοφιλή αντικείμενα πρέπει να διασκορπίζονται σε ολόκληρο το χώρο. Σε πραγματικές εφαρμογές ιστού, το πιο δημοφιλές θέμα χρήστη ή ιστολογίου δεν είναι απαραίτητα το πρώτο λεξικογραφικά αντικείμενο. Για τη διασπορά αντικειμένων στο χώρο, οι δημιουργοί αποφάσισαν να κατακερματίσουν την έξοδο από τη γεννήτρια Gray. Συγκεκριμένα, χρησιμοποιήθηκε η μέθοδος nextItem() για να πάρουν το επόμενο (ακέραιο) στοιχείο και στη συνέχεια δημιουργούν ένα hash αυτής της τιμής για να έχουν το κλειδί που θα χρησιμοποιηθεί. Η επιλογή της συνάρτησης κατακερματισμού είναι κρίσιμη: η ενσωματωμένη συνάρτηση String.hashCode() της Java τείνει να αφήνει τα δημοφιλή αντικείμενα σε ομαδοποίηση. Επιπλέον, μετά από τον κατακερματισμό, οι συγκρούσεις σήμαιναν ότι μόνο το 80 τοις εκατό του χώρου κλειδιών θα είχε δημιουργηθεί στην ακολουθία. Μια προσέγγιση θα ήταν η χρησιμοποιήση τέλειου κατακερματισμού, ο οποίος αποφεύγει τις συγκρούσεις, με μειονέκτημα ότι απαιτείται περισσότερος χρόνος εγκατάστασης για την κατασκευή του τέλειου κατακερματισμού (πολλαπλά λεπτά για εκατοντάδες εκατομμύρια εγγραφές).Η προσέγγιση που ακολούθησαν ήταν να κατασκευάσουμε μια γεννήτρια Zipfian για πολύ μεγαλύτερο χώρο κλειδιών από ό, τι πραγματικά χρειαζόμαστε, έπειτα να εφαρμόσουν τον κατακερματισμό FNV σε κάθε παραγόμενη τιμή και τελικά να πάρουν το mod N (όπου το μέγεθος N ο αριθμός εγγραφών στη βάση δεδομένων). Το αποτέλεσμα ήταν ότι το 99,97% του χώρου κλειδιών δημιουργείται και τα παραγόμενα κλειδιά συνέχισαν να έχουν διανομή Zipfian.

Το δεύτερο ζήτημα αφορούσε την μεταβολή του αριθμού των στοιχείων σε μια κατανομή. Για μερικούς φόρτους εργασίας, εισάγονται νέες εγγραφές στη βάση δεδομένων. Η κατανομή Zipfian θα πρέπει να έχει ως αποτέλεσμα να διατηρούνται δημοφιλείς οι ίδιες εγγραφές, ακόμη και μετά τις εισαγωγές, ενώ στην Πιο Πρόσφατη κατανομή, η δημοτικότητα πρέπει να μετατοπιστεί στα νέα κλειδιά. Για την Πιο Πρόσφατη κατανομή, υπολογίζεται μια νέα κατανομή κατά την εισαγωγή

μιας εγγραφής. Για να υλοποιηθεί αυτό φθηνά, ο αλγόριθμος Gray τροποποιήθηκε ώστε να υπολογίζονται σταδιακά οι σταθερές του. Για το Zipfian, επεκτάθηκε ο αρχικός χώρος κλειδιών στο αναμενόμενο μέγεθος μετά τις εισαγωγές. Εάν ένα σύνολο δεδομένων είχε εγγραφές N και ο φόρτος εργασίας είχε συνολικές λειτουργίες T, με αναμενόμενο κλάσμα I των ενθέτων, τότε η γεννήτρια Zipfian τροποποιείται ώστε να αντλεί από ένα χώρο μεγέθους $N + T \times I + \varepsilon$. Προστέθηκε ένας επιπλέον παράγοντας ε δεδομένου ότι ο πραγματικός αριθμός ενθέτων εξαρτάται από την τυχαία επιλογή των εργασιών κατά τη διάρκεια του φόρτου εργασίας σύμφωνα με μια πολυωνυμική κατανομή. Κατά την εκτέλεση του φόρτου εργασίας, εάν η γεννήτρια παρήγαγε ένα στοιχείο που δεν είχε εισαχθεί ακόμα, αυτή η τιμή παραλείπεται και επιλέγεται ένα άλλο. Έτσι η κατανομή της δημοτικότητας δεν αλλάζει καθώς εισάγονται νέες εγγραφές.

Στα πλαίσια αυτής της διπλωματικής θα χρησιμοποιήσουμε σχεδόν όλους τους βασικούς φόρτους εργασίας που το YCSB προσφέρει, εφόσον κρίνεται ότι μας αρκούν καθώς καλύπτουν πολλά σενάρια, με τη μόνη παραμετροποίηση ότι θα αυξήσουμε τις εγγραφές που θα αποθηκευτούν από 1,000 σε 1,000,000 όπως επίσης και τον αριθμό των ενεργείων πάνω σε αυτές τις εγγραφές από 1,000 σε 1,000,000.

4.2.1 Workload A - Update Heavy Workload

Αυτός ο φόρτος εργασίας περιέχει πολλές λειτουργίες οι οποίες ενημερώνουν τα στοιχεία της βάσης δεδομένων. Το 50% αυτού το φόρτου είναι ενέργειες ανάγνωσης από τη βάση δεδομένων και το υπόλοιπο 50% είναι ενημέρωση των εγγραφών. Το προκαθορισμένο μέγεθος κάθε εγγραφής είναι 1KB. Η κατανομή που χρησιμοποιείται είναι η Zipfian.

4.2.2 Workload B - Read Mostly Benchmark

Αυτός ο φόρτος εργασίας περιέχει πολλές λειτουργίες οι οποίες διαβάζουν τα στοιχεία της βάσης δεδομένων. Το 95% αυτού το φόρτου είναι ενέργειες ανάγνωσης από τη βάση δεδομένων και το υπόλοιπο 5% είναι ενημέρωση των εγγραφών. Το προκαθορισμένο μέγεθος κάθε εγγραφής είναι 1KB. Η κατανομή που χρησιμοποιείται είναι η Zipfian.

4.2.3 Workload C - Read Only Benchmark

Αυτός ο φόρτος εργασίας περιέχει μόνο λειτουργίες οι οποίες διαβάζουν τα στοιχεία της βάσης δεδομένων. Το 100% αυτού το φόρτου είναι ενέργειες ανάγνωσης από τη βάση δεδομένων. Το προκαθορισμένο μέγεθος κάθε εγγραφής είναι 1KB. Η κατανομή που χρησιμοποιείται είναι η Zipfian.

4.2.4 Workload D - Read Latest Benchmark

Αυτός ο φόρτος εργασίας περιέχει πολλές λειτουργίες οι οποίες διαβάζουν τα στοιχεία της βάσης δεδομένων. Το 95% αυτού το φόρτου είναι ενέργειες ανάγνωσης από τη βάση δεδομένων και το υπόλοιπο 5% είναι εισαγωγή εγγραφών. Το προκαθορισμένο μέγεθος κάθε εγγραφής είναι 1KB. Η κατανομή που χρησιμοποιείται είναι η Latest.

4.2.5 Workload F - Read-Modify-Write

Αυτός ο φόρτος εργασίας περιέχει λειτουργίες οι οποίες διαβάζουν τα στοιχεία της βάσης δεδομένων. Το 50% αυτού το φόρτου είναι ενέργειες ανάγνωσης από τη βάση δεδομένων και το υπόλοιπο 50% είναι ανάγνωση-επεξεργασία-αποθήκευση εγγραφών. Το προκαθορισμένο μέγεθος κάθε εγγραφής είναι 1KB. Η κατανομή που χρησιμοποιείται είναι η Zipfian.

5. Υλοποίηση - Πειραματικά Δεδομένα

5.1 Λεπτομέρειες υλοποίησης

5.1.1 Πραγματικό Περιβάλλον - Bare Metal Machine

Για τα benchmarks τα οποία θα εκτελέσουμε στο πραγματικό περιβάλλον θα χρησιμοποιήσουμε το μηχάνημα haci3 το οποίο μας παρέχεται από το εργαστήριο cslab. Το μηχάνημα αυτό έχει τα εξής χαρακτηριστικά:

- Λειτουργικό Σύστημα: Debian 8
- Επεξεργαστής: Intel(R) Xeon(R) CPU E5-2697 v3 @ 2.60GHz με 56 πυρήνες
- Κύρια Μνήμη (RAM) : 198 GB
- Σκληρός Δίσκος: 8 TB HDD

Πρώτου ξεκινήσουμε με τα πειραματικά δεδομένα πρέπει πρώτα να εγκαταστήσουμε όλες τις βάσεις και το YCSB. Αρχικά, πρέπει να ελέγξουμε για τυχόν ενημερώσεις των πακέτων καθώς στα μηχανήματα αυτά έχει μόλις εγκατασταθεί το λειτουργικό. Προκειμένου να ελέγξουμε για τις ενημερώσεις αυτές εκτελούμε την εντολή

```
$ sudo apt-get update
```

Υπάρχουν δύο πιθανά αποτελέσματα σε αυτή τη εντολή. Είτε θα υπάρχουν πακέτα τα οποία πρέπει να ενημερώσουμε είτε δεν υπάρχει κανένα και μπορούμε να προχωρήσουμε με την εγκατάσταση του προγράμματός μας. Στη δική μας περίπτωση επειδή τα μηχανήματα αυτά είναι ακόμα “φρέσκα” θα υπάρχουν πολλά πακέτα τα οποία πρέπει να ενημερωθούν. Επομένως για να ενημερώσουμε αυτά τα πακέτα εκτελούμε την εντολή:

```
$ sudo apt-get upgrade -y
```

Έπειτα από αυτά τα βήματα είμαστε έτοιμοι να εγκαταστήσουμε το εργαλείο YCSB.

Αρχικά διαβάζοντας τα βήματα για την εγκατάσταση του εργαλείου αυτού παρατηρούμε ότι βασίζεται σε μερικά άλλα πακέτα και εργαλεία τα οποία πρέπει να εγκαταστήσουμε. Αυτά είναι η Java και το Maven. Για την Java θα χρησιμοποιήσουμε το πακέτο Open JDK 8. Το εγκαθιστούμε με τις εντολές:

```
$ sudo apt-get update
$ sudo apt-get install openjdk-8-jre -y
```

Προκειμένου να βεβαιωθούμε ότι η Java έχει εγκατασταθεί σωστά εκτελούμε την εντολή

```
$ java -version
```

Το αποτέλεσμα που περιμένουμε να δούμε μοιάζει με το παρακάτω:

```
openjdk version "1.8.0_265"
```

```
OpenJDK Runtime Environment (build 1.8.0_265-8u265-b01-0ubuntu2~16.04-b01)
OpenJDK 64-Bit Server VM (build 25.265-b01, mixed mode)
```

Το YCSB έχει σαν εξάρτηση και το πακέτο Maven 3. Εδώ χρειάζεται προσοχή καθώς αν εγκατασταθεί το Maven 2 το YCSB θα βγάζει διάφορα μηνύματα λάθους (errors) με αποτέλεσμα να μην λειτουργήσει. Το Maven 3 το εγκαθιστούμε με τις εντολές:

```
$ sudo apt-get update
$ sudo apt-get install maven -y
```

Προκειμένου να βεβαιωθούμε ότι η Java έχει εγκατασταθεί σωστά εκτελούμε την εντολή

```
$ mvn --version
```

Το αποτέλεσμα που περιμένουμε να δούμε μοιάζει με το παρακάτω:

```
Maven home: /usr/share/maven
Java version: 1.8.0_265, vendor: Private Build
Java version: 1.8.0_265, vendor: Private Build
Java home: /usr/lib/jvm/java-8-openjdk-amd64/jre
Default locale: en_US, platform encoding: ANSI_X3.4-1968
OS name: "linux", version: "4.4.0-189-generic", arch: "amd64", family:
"unix"
```

Εφόσον εγκαταστήσαμε τα απαραίτητα πακέτα τα οποία χρειάζονται για την εγκατάσταση του YCSB, εγκαθιστούμε το ίδιο με τις παρακάτω εντολές:

```
$ sudo apt-get update
$ curl -O --location https://github.com/brianfrankcooper/YCSB/$ releases/
download/0.17.0/ycsb-0.17.0.tar.gz
$ tar xfvz ycsb-0.17.0.tar.gz
$ cd ycsb-0.17.0
```

Η έκδοση του YCSB που χρησιμοποιούμε είναι η 0.17.0 η οποία είναι η τελευταία σταθερή έκδοση τη χρονική περίοδο που γράφεται αυτή η εργασία. Προκειμένου να βεβαιωθούμε ότι το πρόγραμμα έχει εγκατασταθεί σωστά εκτελούμε την εντολή

```
$ ./bin/ycsb
```

και περιμένουμε να μας εμφανιστεί ένα μήνυμα λάθους και να μας δείχνει τη σωστή χρήση των εντολών του εργαλείου αυτού. Αποφασίζουμε να χρησιμοποιήσουμε όλους τους προεπιλεγμένους φόρτους εργασίας που παρέχει το εργαλείο αυτό (εκτός από το workloade), αλλά με τη διαφορά ότι θα αυξήσουμε τον αριθμό των εγγραφών σε κάθε φόρτο εργασίας και τον αριθμό των λειτουργιών που θα εκτελεστούν πάνω σε αυτά τα δεδομένα από 1,000 σε 1,000,000. Πλοηγούμαστε στο φάκελο ***ycsb-0.17.0/workloads*** και εφαρμόζουμε τις παρακάτω αλλαγές σε όλους τους φόρτους εργασίας που θα χρησιμοποιήσουμε:

```
recordcount=1000 –> recordcount=1000000
```

```
operationcount=1000 –> operationcount=1000000
```

Το επόμενο βήμα είναι να εγκαταστήσουμε τις βάσεις που θα χρησιμοποιήσουμε και να τις διαμορφώσουμε ανάλογα ώστε να επιτύχουμε τις διάφορες συμπεριφορές που θέλουμε από αυτές.

Αρχικά για την Redis χρειαζόμαστε τα πακέτα make, tcl και gcc. Επομένως εκτελούμε τις παρακάτω εντολές:

```
$ sudo apt-get update
$ sudo apt install make -y
$ sudo apt install tcl -y
$ sudo apt install gcc -y
```

Προκειμένου να επαληθεύσουμε ότι τα πακέτα αυτά έχουν εγκατασταθεί σωστά εκτελούμε την εντολή:

```
$ make
```

και σαν αποτέλεσμα περιμένουμε το μήνυμα:

```
make: *** No targets specified and no makefile found.  Stop.
```

Για το gcc εκτελούμε την εντολή:

```
$ gcc –version
```

και ως αποτέλεσμα περιμένουμε ένα μήνυμα της μορφής:

```
gcc (Ubuntu 5.4.0-6ubuntu1~16.04.12) 5.4.0 20160609
Copyright (C) 2015 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR
PURPOSE.
```

Τέλος για το πακέτο tcl δεν μπορούμε άμεσα να προσδιορίσουμε τη σωστή εγκατάστασή του, αλλά όταν εγκαταστήσουμε τη Redis και δοκιμάσουμε να τρέξουμε τα τεστ για να βεβαιωθούμε για τη δική της σωστή εγκατάσταση, εαν το πακέτο tcl έχει εγκατασταθεί σωστά τα τεστς θα ξεκινήσουν να τρέχουν, διαφορετικά όχι. Εφόσον εγκαταστήσουμε αυτά τα πακέτα προχωράμε με την εγκατάσταση της Redis. Από την επίσημη ιστοσελίδα τους ακολουθούμε τις εντολές για την εγκατάσταση της Redis οι οποίες είναι:

```
$ wget http://download.redis.io/redis-stable.tar.gz
$ tar xvzf redis-stable.tar.gz
$ cd redis-stable
$ make
```

Εφόσον τελειώσει η τελευταία εντολή ***make*** ένα προαιρετικό, αλλα σημαντικό, βήμα είναι το τρέξιμο των τεστς. Γι'αυτό πηγαίνουμε στο φάκελο ***redis-stable*** και εκτελούμε την εντολή:

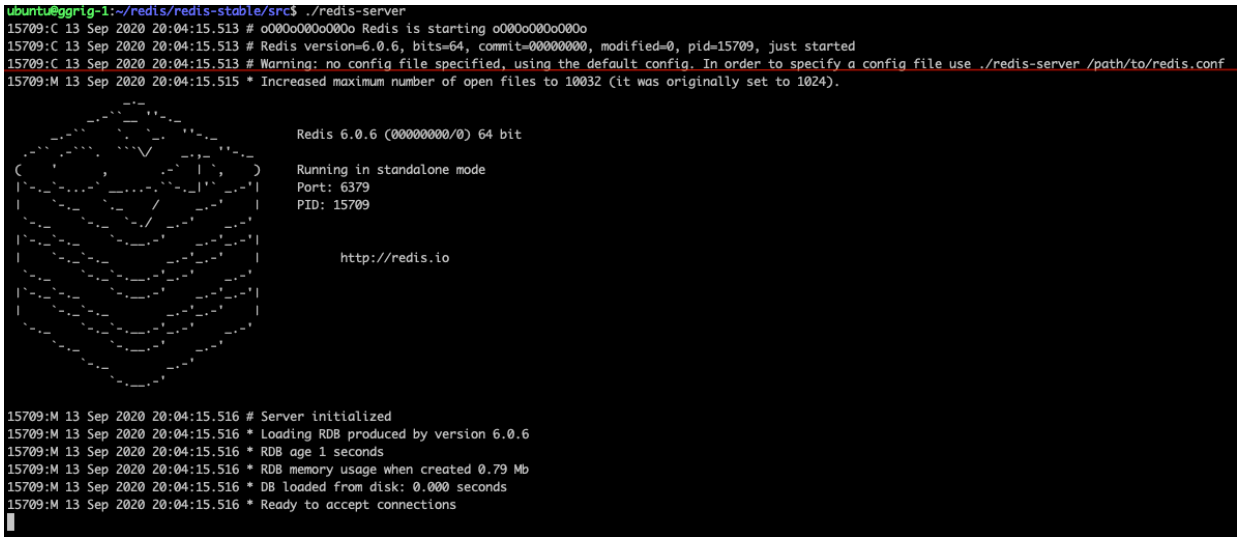
```
$ make test
```

Αν όλα τα τεστ επιτύχουν τότε είμαστε βέβαιοι ότι η βάση αυτή έχει εγκατασταθεί σωστά στο σύστημά μας. Για να δοκιμάσουμε να τρέξουμε την Redis μετακινούμαστε στο φάκελο

redis-stable/src και εκτελούμε την εντολή:

```
$ ./redis-server
```

Το αποτέλεσμα αυτής της εντολής είναι μια οθόνη παρόμοια με την παρακάτω:



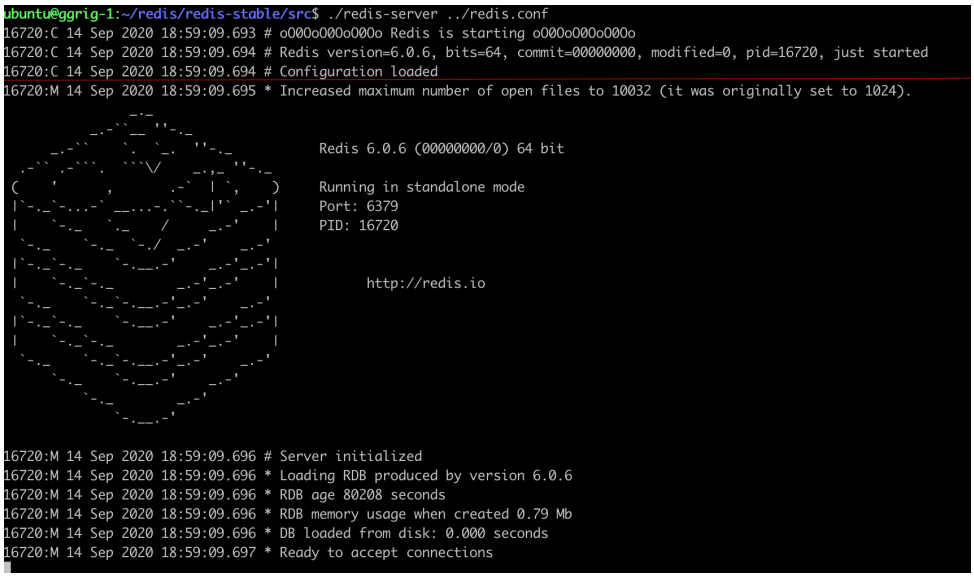
Εικόνα 8: Redis χωρίς configuration file loaded

Παρατηρούμε ότι η βάση μας με την προεπιλεγμένη διαμόρφωση δέχεται αιτήματα στην πόρτα 6379 και τρέχει σε standalone mode που σημαίνει ότι δεν είναι μέρος κάποιου συμπλέγματος. Για να την σταματήσουμε απλά πατάμε ταυτόχρονα τα πλήκτρα ***ctrl + c*** .

Θα παραμετροποιήσουμε τη Redis ανάλογα με τη περίσταση που θέλουμε. Κάθε φορά προκειμένου η Redis να εκτελεστεί βασισμένη στο αρχείο redis.conf όπου θα αλλάξουμε θα πρέπει να τρέχουμε την εντολή μέσα από το φάκελο ***/redis-stable/src*** ως:

```
$ ./redis-server ../redis.conf
```

Οπότε παρατηρώντας την υπογραμμισμένη ειδοποίηση στην παραπάνω και παρακάτω εικόνα παρατηρούμε ότι οι ρυθμίσεις φορτώθηκαν σωστά.



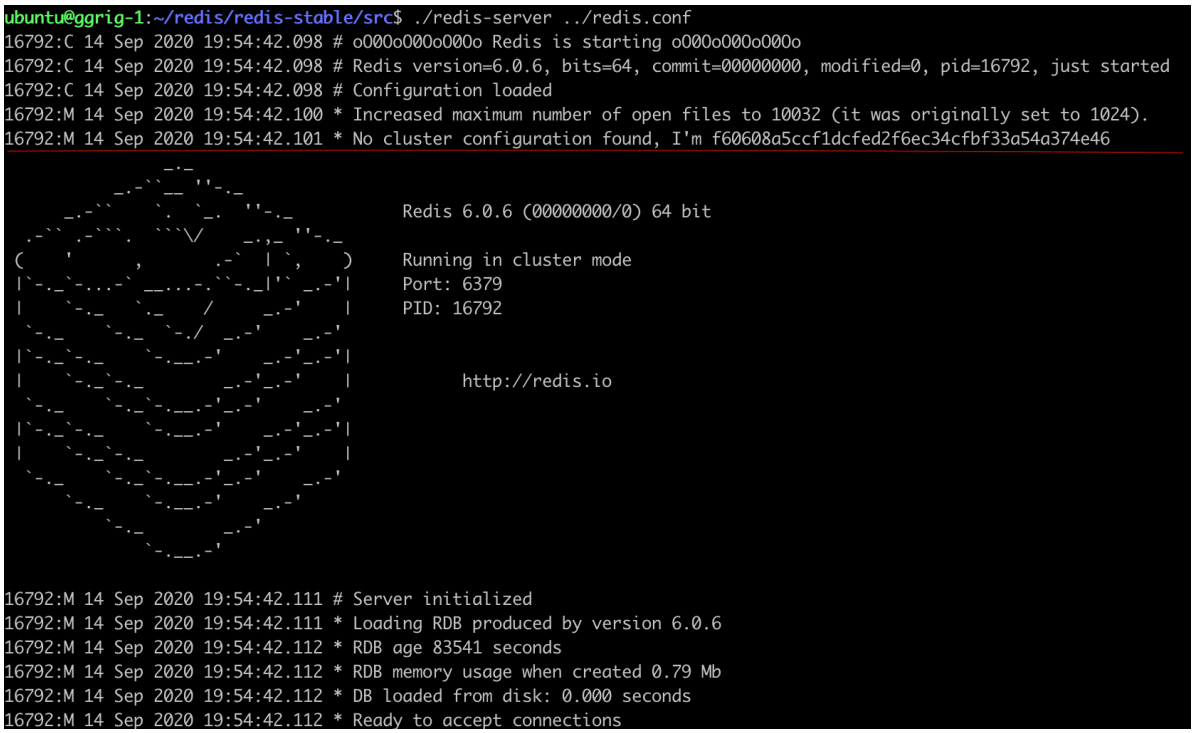
Εικόνα 9: Redis με configuration file loaded

Οι επόμενες αλλαγές αφορούν τις 3 κατηγορίες τεστ που θα κάνουμε στην Redis. Αρχικά θα χρησιμοποιήσουμε τη Redis με τη προεπιλεγμένη παραμετροποίηση δηλαδή σε κατάσταση μονού κόμβου (standalone mode) και αποθηκεύοντας τα δεδομένα στο σκληρό δίσκο. Η επόμενη κατηγορία αφορά πάλι κατάσταση μονού κόμβου αλλά αυτή τη φορά θα απενεργοποιήσουμε την αποθήκευση στο σκληρό δίσκο, οπότε όλα τα δεδομένα θα αποθηκευτούν στη μνήμη ram και σε περίπτωση που σταματήσουμε την Redis τα δεδομένα χάνονται. Προκειμένου να το επιτύχουμε αυτό θα πρέπει να βάλουμε σε σχόλιο τις παρακάτω γραμμές που βρίσκονται στο αρχείο redis.conf που ουσιαστικά απενεργοποιούν τη λειτουργία ***snapshotting***.

```
save 900 1 –> # save 900 1
save 300 10 –> # save 300 10
save 60 10000 –> # save 60 10000
```

Η τελευταία κατηγορία τεστ που θα κάνουμε στην Redis είναι ότι θα την τρέξουμε σε λειτουργία συμπλέγματος χωρίς να αποθηκεύει τα δεδομένα στο σκληρό δίσκο, αλλά να τα κρατάει μόνο στη μνήμη ram. Επομένως μέ όλες τις παραπάνω αλλαγές που έχουμε κάνει, θα πρέπει να βγάλουμε από σχόλιο τις παρακάτω γραμμές που βρίσκονται στο αρχείο redis.conf ώστε να ενεργοποιήσουμε τη λειτουργία συμπλέγματος (cluster mode) της Redis.

```
# cluster-enabled yes –> cluster-enabled yes
# cluster-config-file nodes-6379.conf –> cluster-config-file
nodes-6379.conf
# cluster-node-timeout 15000 –> cluster-node-timeout 15000
```



Εικόνα 10: Redis χωρίς cluster configuration loaded

Μπορούμε να επιβεβαιώσουμε ότι αυτή η λειτουργία ενεργοποιήθηκε παρατηρώντας για την υπογραμμισμένη γραμμή στην παραπάνω εικόνα.

Επειδή όλοι οι κόμβοι των βάσεων θα φορτωθούν στο ίδιο σύστημα θα χρειαστούμε 8 διαφορετικά αρχεία παραμετροποίησης, ένα για κάθε κόμβο που θέλουμε να τρέξουμε. Πλοηγούμαστε στο φάκελο ***/redis-stable.*** Προκειμένου να δημιουργήσουμε αντίγραφα του προεπιλεγμένου αρχείου διαμόρφωσης εκτελούμε την παρακάτω εντολή

```
$ cp redis.conf redisN.conf
```

όπου το N είναι μια μεταβλητή η οποία παίρνει τις τιμές ένα έως εφτά (1-7) προκειμένου να δημιουργήσουμε τελικά οκτώ (8) μηχανήματα. Αρχικά πρέπει να αλλάξουμε μερικές γραμμές σε αυτά τα αρχεία παραμετροποίησης που δημιουργήσαμε. Πρέπει να αλλάξουμε τις θύρες με τις οποίες κάθε κόμβος ανταλλάζει και δέχεται μηνύματα. Οι θύρες που θα χρησιμοποιήσουμε είναι οι 6380-6387. Επομένως πρέπει να αλλάξουμε τις παρακάτω γραμμές για κάθε θύρα αντίστοιχα:

```
port 638x
pidfile /var/run/redis_638x.pid
cluster-config-file nodes-638x.conf
```

Η επόμενη βάση η οποία θα εγκαταστήσουμε είναι η Aerospike. Από την επίσημη ιστοσελίδα παρατηρούμε ότι το Aerospike Server έχει σαν εξάρτηση τη βιβλιοθήκη ***libcurl***. Παρατηρούμε ότι για τη δική μας έκδοση των Ubuntu η οποία είναι η 16.04 χρειαζόμαστε τη βιβλιοθήκη ***libcurl3***. Επομένως τρέχουμε τις παρακάτω εντολές για να την εγκαταστήσουμε.

```
$ sudo apt-get update
$ sudo apt-get install -y libcurl3
```

Και έπειτα για την εγκατάσταση της ίδιας της Aerospike στη δική μας έκδοση των Ubuntu εκτελούμε τις εντολές:

```
$ wget -O aerospike.tgz 'https://www.aerospike.com/download/server/latest/artifact/ubuntu16'
$ tar -xvf aerospike.tgz
```

Ανάλογα με το αν η έκδοση που εγκαταστήσαμε είναι η Community Edition είτε η Enterprise Edition θα έχει δημιουργηθεί ο αντίστοιχος φάκελος. Στη δική μας περίπτωση που εγκαταστήσαμε την Community Edition έχει δημιουργηθεί ο φάκελος ***aerospike-server-community-5.1.0.4-ubuntu16.04*** **.** Μπαίνουμε μέσα σε αυτό τον φάκελο και εκτελούμε την εντολή

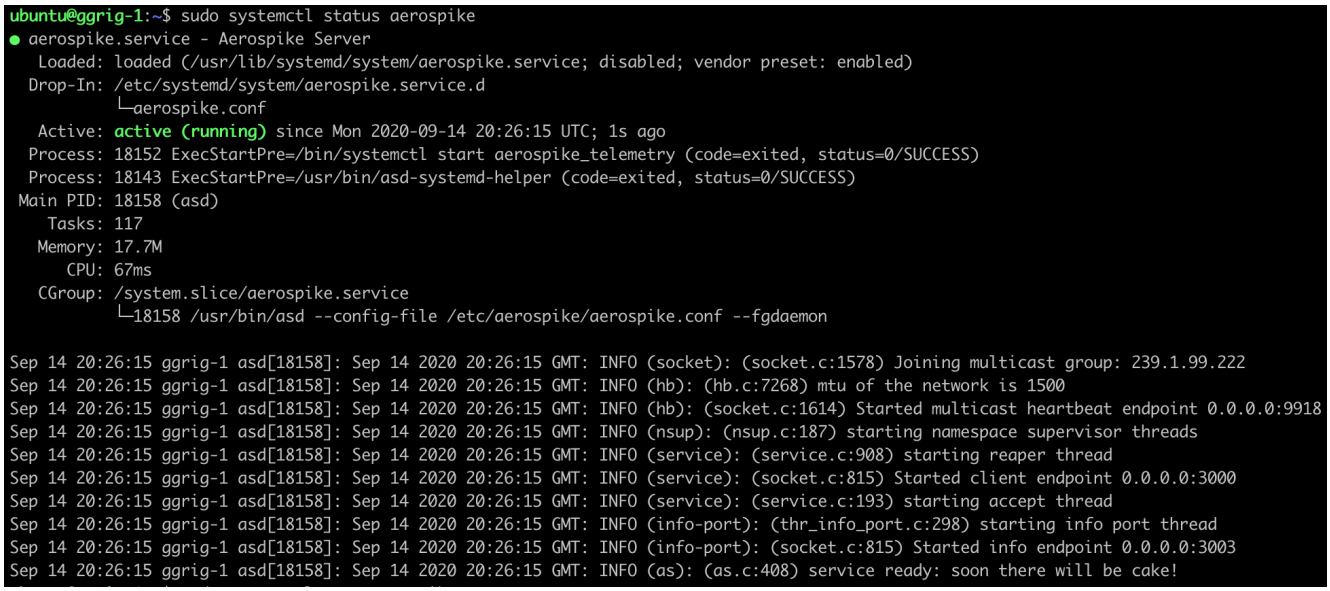
```
$ sudo ./asinstall
```

προκειμένου να εγκαταστήσουμε την Aerospike στο σύστημά μας. Εφόσον τελειώσει η εγκατάσταση μπορούμε να ελέγξουμε τη λειτουργία της Aerospike με τις παρακάτω εντολές:

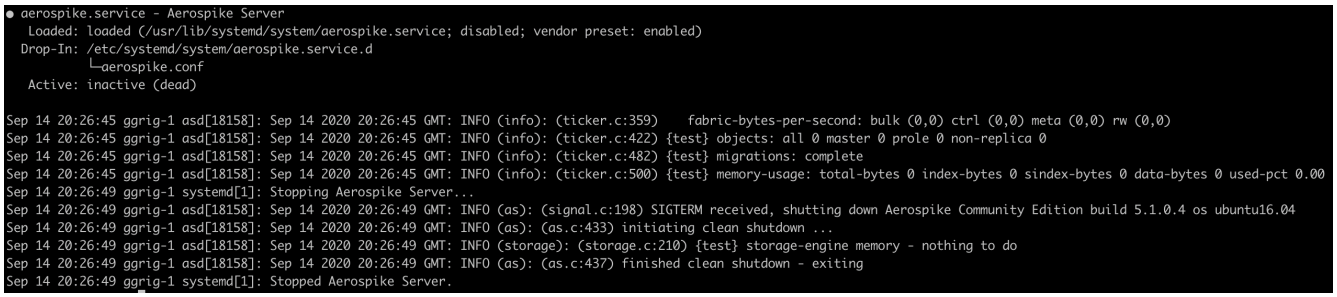
```
$ sudo systemctl start aerospike
$ sudo systemctl status aerospike
$ sudo systemctl stop aerospike
```

Με τη σειρά την οποία αναφέρθηκαν αυτές οι εντολές ξεκινούν την Aerospike σε λειτουργία daemon, εμφανίζουν την κατάσταση της Aerospike και σταματούν την εκτέλεση της Aerospike. Παρακάτω βλέπουμε το αποτέλεσμα της εντολής ***sudo systemctl status aerospike***

αφού ξεκινήσουμε και αφού έπειτα σταματήσουμε την Aerospike.



Εικόνα 11: Aerospike daemon σε εκτέλεση



Εικόνα 12: Aerospike daemon σε αδράνεια

Η Aerospike έχει αρκετά πιο απλή παραμετροποίηση από την Redis. Το αρχείο το οποίο είναι υπεύθυνο για την παραμετροποίηση της Aerospike είναι το ***/etc/aerospike/aerospike.conf*** **.** Ανοίγοντας το αρχείο αυτό παρατηρούμε ότι υπάρχουν δύο namespaces. Το namespace ***test*** και το namespace ***bar***. Διαγράφουμε το namespace ***bar*** καθώς δε χρειαζόμαστε και τα δύο. Η αλλαγή η οποία θα κάνουμε στο namespace ***test*** είναι η παρακάτω:

```
replication-factor 2 –> replication-factor 1
```

Η αλλαγή αυτή ουσιαστικά αποτρέπει την Aerospike να αντιγράψει τα δεδομένα τα οποία έχει ήδη πάνω απο μία φορά με αποτέλεσμα να μη δεσμεύονται παραπάνω πόροι του συστήματος αλλά και υπολογιστική ισχύς. Όμοια με την Redis, επειδή όλοι οι κόμβοι θα εκτελεστούν στο ίδιο μηχάνημα θα πρέπει να δημιουργήσουμε επιπρόσθετα αρχεία παραμετροποίησης, ένα για κάθε κόμβο. Οπότε πλοηγούμαστε στο φάκελο ***aerospike-server/as/etc*** . Προκειμένου να δημιουργήσουμε αντίγραφα του προεπιλεγμένου αρχείου διαμόρφωσης εκτελούμε την παρακάτω εντολή

```
$ cp aerospike_dev.conf aerospike_devN.conf
```

όπου το N είναι μια μεταβλητή η οποία παίρνει τις τιμές ένα έως εφτά (1-7) προκειμένου να δημιουργήσουμε τελικά οκτώ (8) μηχανήματα.

αποτελέσματα των τριών βάσεων στο πραγματικό περιβάλλον.

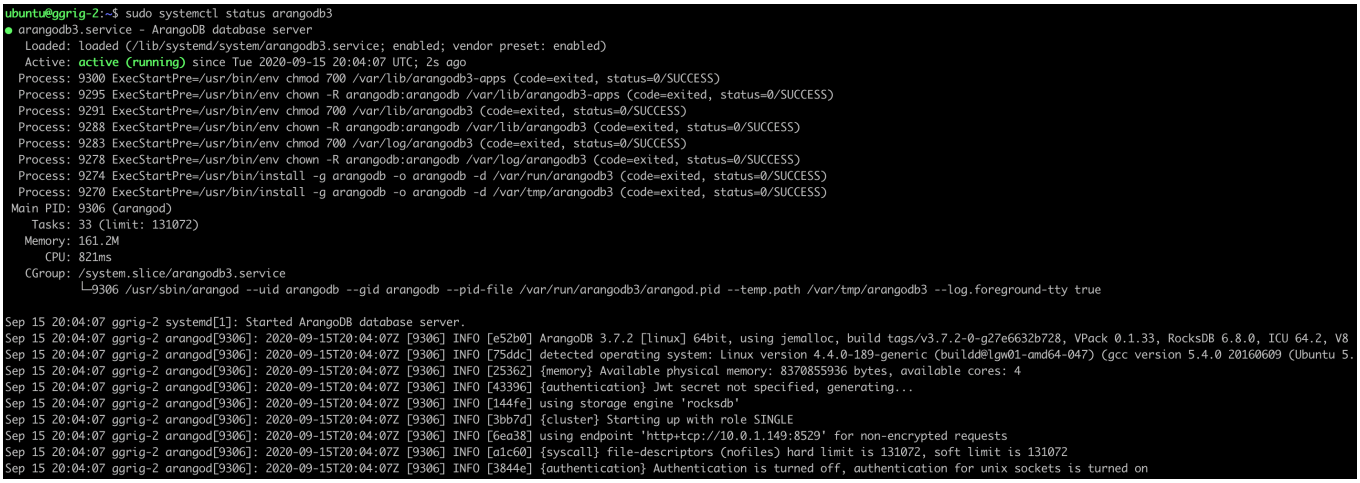
Η τελευταία βάση η οποία θα εξετάσουμε είναι η ArangoDB. Διαβάζοντας τις οδηγίες στην επίσημη ιστοσελίδα της ArangoDB για την εγκατάσταση θα χρειαστεί να εκτελέσουμε τις παρακάτω εντολές:

```
$ curl -OL https://download.arangodb.com/arangodb37/DEBIAN/Release.key
$ sudo apt-key add - < Release.key
$ echo 'deb https://download.arangodb.com/arangodb37/DEBIAN/ /' | sudo tee
/etc/apt/sources.list.d/arangodb.list
$ sudo apt-get install apt-transport-https
$ sudo apt-get update
$ sudo apt-get install arangodb3=3.7.2-1
```

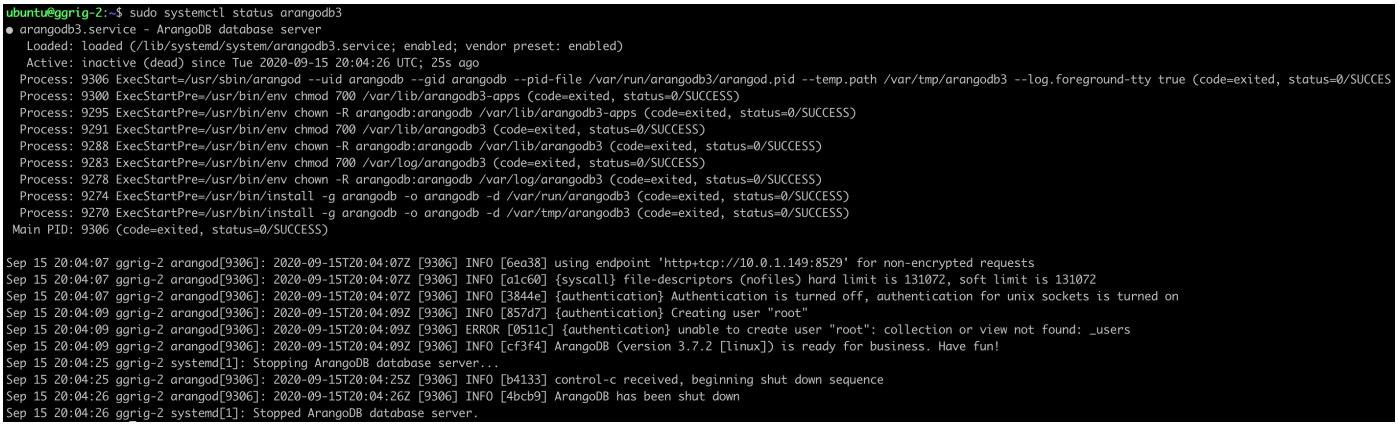
Εφόσον τελειώσει η εγκατάσταση μπορούμε να ελέγξουμε τη λειτουργία της ArangoDB με τις παρακάτω εντολές:

```
$ sudo systemctl start arangodb3
$ sudo systemctl status arangodb3
$ sudo systemctl stop arangodb3
```

Με τη σειρά την οποία αναφέρθηκαν αυτές οι εντολές ξεκινούν την ArangoDB σε λειτουργία daemon, εμφανίζουν την κατάσταση της ArangoDB και σταματούν την εκτέλεση της ArangoDB. Παρακάτω βλέπουμε το αποτέλεσμα της εντολής ***sudo systemctl status arangodb3*** αφού ξεκινήσουμε και αφού έπειτα σταματήσουμε την ArangoDB.



Εικόνα 13: ArangoDB daemon σε εκτέλεση



Εικόνα 14: ArangoDB daemon σε αδράνεια

Σε αυτό το περιβάλλον δε θα παραμετροποιήσουμε την ArangoDB καθώς οι προεπιλεγμένη διαμόρφωση είναι η βέλτιστη.

5.1.2 Εικονικό Περιβάλλον - Cloud Machines

Για τα τεστ στο εικονικό περιβάλλον έχουμε δεσμεύσει 9 εικονικές μηχανές (VMs). Θα εγκαταστήσουμε τις βάσεις δεδομένων στα 8 μηχανήματα με τα πιο καλά χαρακτηριστικα και να εγκαταστήσουμε το YCSB στο 9ο με τα λιγότερο καλά χαρακτηριστικά καθώς οι απαιτήσεις αυτού του προγράμματος δεν είναι τόσο μεγάλες και επίσης θέλουμε να κρατήσουμε συνέπεια ανάμεσα στους κόμβους των βάσεων ώστε ο κάθε κόμβος να είναι ομότιμος με τους υπόλοιπους.Οι πρώτες 8 μηχανές έχουν τα παρακάτω χαρακτηριστικά:

- Λειτουργικό Σύστημα: Ubuntu 16.04 LTS
- Επεξεργαστής: Intel Core 5/7 6ης γενιάς αρχιτεκτονικής Skylake με 4 πυρήνες
- Κύρια Μνήμη (RAM) : 8 GB
- Σκληρός Δίσκος: 60 GB HDD

Η 9η εικονική μηχανή έχει χαρακτηριστικά:

- Λειτουργικό Σύστημα: Ubuntu 16.04 LTS
- Επεξεργαστής: Intel Core 5/7 6ης γενιάς αρχιτεκτονικής Skylake με 4 πυρήνες
- Κύρια Μνήμη (RAM) : 4 GB
- Σκληρός Δίσκος: 60 GB HDD

Για να συνδεθούμε σε ένα τέτοιο μηχάνημα χρησιμοποιούμε το πρωτόκολλο SSH. Ο λόγος που χρησιμοποιούμε αυτό το πρωτόκολλο είναι επειδή χρειαζόμαστε μια ασφαλή σύνδεση με τα μηχανήματα μας πάνω σε ένα μη ασφαλές δίκτυο. Επομένως με την εντολή

```
$ ssh ubuntu@node_ip
```

όπου node_ip είναι η IP του εκάστοτε κόμβου συνδεόμαστε στα μηχανήματα.

Οι παραμετροποίηση των βάσεων είναι παρόμοια με την διαδικασία που περιγράφηκε παραπάνω. Οι μόνες διαφορές για την Redis και την Aerospike είναι ότι δε χρειάζεται να φτιάξουμε τα παραπάνω αρχεία παραμετροποίησης καθώς τώρα έχουμε ένα τέτοιο αρχείο ανά διαφορεικό κόμβο. Ακόμα στη Redis πλοηγούμαστε στο φάκελο ***/redis-stable*** και μέσα στο αρχείο ***redis.conf*** αλλάζουμε τις γραμμές από και σε:

```
bind 127.0.0.1 -> bind 0.0.0.0
protected-mode yes -> protected-mode no
```

ώστε να μπορεί να επικοινωνεί με μηχανήματα με διαφορετικές IP διευθύνσεις. Στην ArangoDB πρέπει να κάνουμε μερικές αλλαγές. Το αρχείο που ευθύνεται για την παραμετροποίηση είναι το ***/etc/arangodb3/arangod.conf*** . Αρχικά η αλλαγή που θα κάνουμε είναι να αλλάξουμε το endpoint το οποίο ευθύνεται για την οποιαδήποτε επικοινωνία της.

```
endpoint = tcp://127.0.0.1:8529 –> endpoint = tcp://node_ip:8529
```

όπου αντί για node_ip αντικαθιστούμε με την ip του μηχανήματος όπου είμαστε συνδεδεμένοι. Η επόμενη γραμμή πρέπει να προστεθεί κάτω από την ενότητα ***[database]*** στο ίδιο αρχείο όπου κάναμε και τις προηγούμενες αλλαγές.

```
cache.size = 2867011686
```

Αυτή η γραμμή ουσιαστικά προσδιορίζει την μνήμη cache της βάσης δεδομένων. Επιλέξαμε αυθαίρετα αυτό το νούμερο, αλλά σίγουρα να είναι μεγαλύτερο από το μέγεθος των δεδομένων που έχουμε σκοπό να προσθέσουμε στη βάση. Οι επόμενες αλλαγές πρέπει να εισαχθούν όλες κάτω από την ενότητα ***[rocksdb]*** στο ίδιο αρχείο όπου κάναμε και τις προηγούμενες αλλαγές.

```
total-write-buffer-size = 0
block-cache-size = 2867011686
```

Με τη σειρά η πρώτη γραμμή επιβάλλει στην ArangoDB να μην περιορίζει τη μνήμη που χρησιμοποιεί η βάση. Η δεύτερη γραμμή προσδιορίζει το μέγιστο μέγεθος των μπλοκ της προσωρινής μνήμης σε bytes. Η προκαθορισμένη τιμή σε ένα σύστημα προσδιορίζεται από τον τύπο ***(system RAM size - 2GiB) * 0.3.*** Στη δική μας περίπτωση όπου έχουμε 8 GB ram στα μηχανήματα αυτά, η μεταβλητή αυτή ισούται με 1,8. Για το λόγο αυτό αποφασίζουμε να τη μεγαλώσουμε σχεδόν στα 3 GB ώστε να είμαστε σίγουροι ότι θα περιέχονται όλα τα δεδομένα και δε θα υπάρχει περίπτωση να γίνει κάποια εκδίωξη δεδομένων της μνήμης (cache eviction).

Το επόμενο βήμα είναι να διαμορφώσουμε κατάλληλα τα μηχανήματα ώστε να μπορούν να προσφέρουν τα καλύτερα δυνατά αποτελέσματα στις βάσεις που θα εγκαταστήσουμε χωρίς περιορισμούς. Η διαδικασία αυτή η οποία θα περιγράψουμε παρακάτω πρέπει να επαναληφθεί για κάθε μία από τις 8 μεγαλύτερες εικονικές μηχανές που έχουμε στη διάθεση μας ώστε όλοι οι κόμβοι να είναι ομότιμοι μεταξύ τους. Αρχικά πρέπει αλλάξουμε μερικές παραμέτρους στο σύστημα μας και στον πυρήνα του συστήματος μας (kernel) ώστε να γίνεται αποτελεσματικότερη εκμετάλλευση των πόρων του συστήματος από τις βάσεις δεδομένων. Η πρώτη αλλαγή που θα κάνουμε είναι να προσθέσουμε τη γραμμή:

```
vm.overcommit_memory = 1
```

στο αρχείο που βρίσκεται στο φάκελο ***/etc/sysctl.conf*** . Ο λόγος που το κάνουμε αυτό είναι ο διαχωρισμός των διεργασιών σε νήματα εκτέλεσης (forks) όπως υλοποιούνται στα σημερινά σύγχρονα συστήματα. Για παράδειγμα η Redis χρησιμοποιεί τα forks προκειμένου να αποθηκεύσει τα δεδομένα στο δίσκο. Αυτό το πετυχαίνει με το να φτιάχνει μια διεργασία-παιδί που είναι ένα ακριβές αντίγραφο του γονέα, δηλαδή της αρχικής διεργασίας. Η διεργασία-παιδί αποθηκεύει στον δίσκο και τελικά τερματίζει. Θεωρητικά, το παιδί θα πρέπει να χρησιμοποιεί τόση μνήμη όσο και ο γονέας που είναι αντίγραφο του, αλλά στην πραγματικότητα χάρη στο σημασιολογικό αντίγραφο-εγγραφής που εφαρμόζεται από τα περισσότερα σύγχρονα λειτουργικά συστήματα, η διαδικασία γονέα και παιδιού θα μοιραστεί τις κοινές σελίδες μνήμης. Μια σελίδα θα αναπαραχθεί μόνο όταν αλλάζει στο παιδί ή στον γονέα. Δεδομένου ότι θεωρητικά όλες οι σελίδες ενδέχεται να αλλάξουν κατά την αποθήκευση της διεργασίας-παιδιού, τα Linux δεν μπορεί εκ των προτέρων να υπολογίσουν πόση μνήμη θα πάρει το παιδί, οπότε αν η ρύθμιση overcommit_memory έχει οριστεί σε μηδέν το fork αυτό θα αποτύχει εκτός αν υπάρχει τόσο ελεύθερη μνήμη RAM όσο απαιτείται ώστε να αναπαραχθούν πραγματικά όλες τις γονικές σελίδες μνήμης. Η ρύθμιση overcommit_memory σε 1 επιβάλλει στα Linux να εκτελέσουν το fork αυτό με πιο οπτιμιστική

κατανομή μνήμης. Η επόμενη αλλαγή που χρειάζεται να κάνουμε είναι να αυξήσουμε την απόδοση του δικτύου. Αυτό το καταφέρνουμε προσθέτοντας τη γραμμή

```
net.core.somaxconn=65535
```

στο αρχείο ***/etc/sysctl.conf*** πριν από τη γραμμή ***exit 0 .*** Αυτό θα μας επιτρέψει να αυξήσουμε τον αριθμό των εισερχόμενων συνδέσεων. Για την τελευταία βελτιστοποίηση του συστήματος αυτού θα χρειαστεί να αλλάξουμε μια παράμετρο του πυρήνα των linux οπότε πρέπει να έχουμε πλήρη δικαιώματα διαχειριστή (root access). Εκτελούμε τις παρακάτω εντολές:

```
$ sudo su
$ echo never > /sys/kernel/mm/transparent_hugepage/enabled
```

Η τελευταία εντολή ουσιαστικά απενεργοποιεί τη λειτουργία ***transparent huge pages*** του πυρήνα των Linux, που επηρεάζουν αρνητικά σε μεγάλο βαθμό τόσο τη χρήση μνήμης όσο και τον χρόνο καθυστέρησης της μνήμης.

5.2 Πειραματικά Αποτελέσματα

5.2.1 Πειραματικά Αποτελέσματα σε Πραγματικό Περιβάλλον - Throughput-Latency

Έχοντας παραμετροποιήσει τα παραπάνω συστήματα θα αρχίσουμε τις συγκρίσεις των τριών βάσεων. Σε κάθε βάση θα εκτελέσουμε τα τεστ για τους διάφορους φόρτους εργασίας που έχουμε αναλύσει σε προηγούμενο κεφάλαιο και για κάθε φόρτο εργασίας θα δοκιμάσουμε διαφορετικό αριθμό νημάτων διεργασιών του εργαλείου YCSB. Η ακριβής διαδικασία που θα ακολουθήσουμε για κάθε βάση είναι η εξής:

Αρχικά φορτώνουμε τα δεδομένα από το εργαλείο YCSB με τις εξής εντολές για κάθε βάση αντίστοιχα:

```
$ cd ycsb-0.17.0/
$ ./bin/ycsb load redis -s -P workloads/workloada -p
“redis.host=127.0.0.1” -p "redis.port=6379" -threads 16 > outputLoad.txt
$ ./bin/ycsb load aerospike -s -P workloads/workloada -p as.host=127.0.0.1
-p as.namespace=test -threads 16 > outputLoad.txt
$ ./bin/ycsb load arangodb -s -P workloads/workloada -threads 16 >
outputLoad.txt
```

Εννοείται ότι τρέχουμε κάθε βάση ανεξάρτητα από τις υπόλοιπες έτσι ώστε κάθε βάση να έχει διαθέσιμους όλους τους υπολογιστικούς πόρους. Εφόσον φορτωθούν τα δεδομένα θα τρέξουμε με διάφορους συνδυασμούς τη κάθε βάση. Για κάθε βάση θα εκτελέσουμε τους φόρτους εργασίας έχοντας 1,3,6 και 8 κόμβους και για κάθε αριθμό κόμβων θα εκτελέσουμε αυτούς τους φόρτους εργασίας με 1,4,8 και 16 νήματα διεργασιών του εργαλείου YCSB. Ο τρόπος που θα εκτελέσουμε τους φόρτους είναι ο εξής:

1. workloada
2. workloadb
3. workloadc
4. workloadf
5. workloadd

Ο λόγος που τρέχουμε το workloadf πριν το workloadd είναι επειδή το workloadd προσθέτει εγγραφές. Για την Redis οι εντολές που τρέχουμε για ένα κόμβο είναι:

```
$ ./bin/ycsb run redis -s -P workloads/workloadx -p “redis.host=127.0.0.1”
-p “redis.port=6379" -p -threads n  > outputRun.txt
```

και για τους 3,6 και 8 κόμβους την εντολή

```
$ ./bin/ycsb run redis -s -P workloads/workloadx -p “redis.host=127.0.0.1"
-p “redis.port=6379" -p “redis.cluster=true” -threads n  > outputRun.txt
```

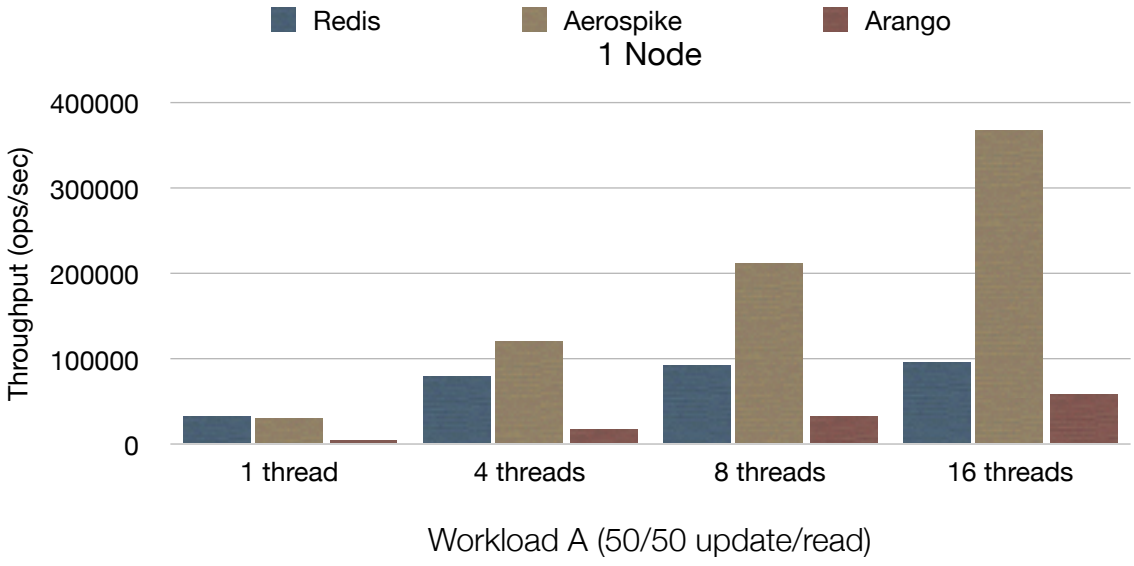
Για την Aerospike οι εντολές που τρέχουμε για οποιοδήποτε αριθμό κόμβων είναι:

```
$ ./bin/ycsb run aerospike -s -P workloads/workloadx -p as.host=node_ip -p
as.namespace=test -threads n > outputRun.txt
```

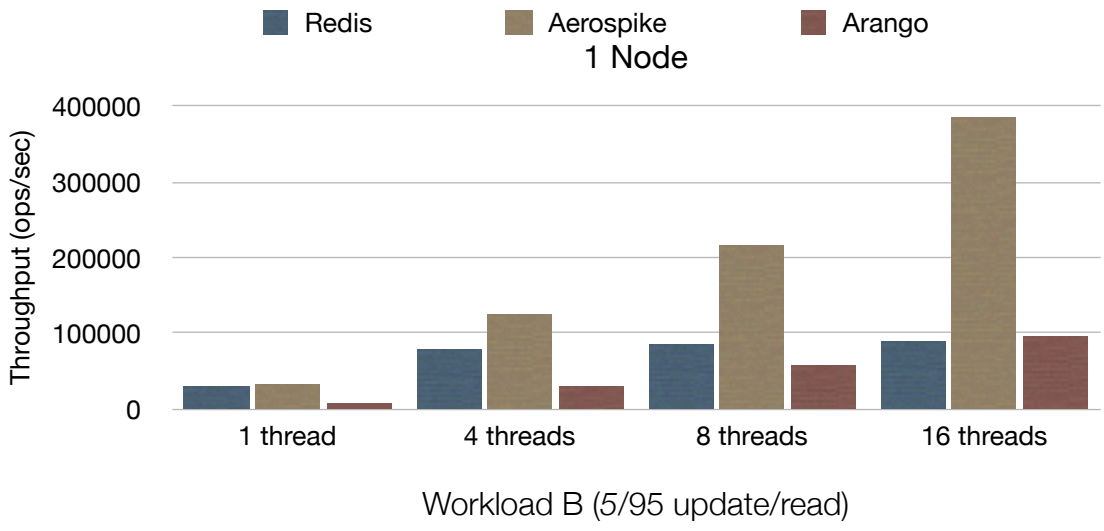
Και τέλος για την ArangoDB οι εντολές που τρέχουμε για οποιοδήποτε αριθμό κόμβων είναι:

```
$ ./bin/ycsb run arangodb -s -P workloads/workloadx -threads n >
outputRun.txt
```

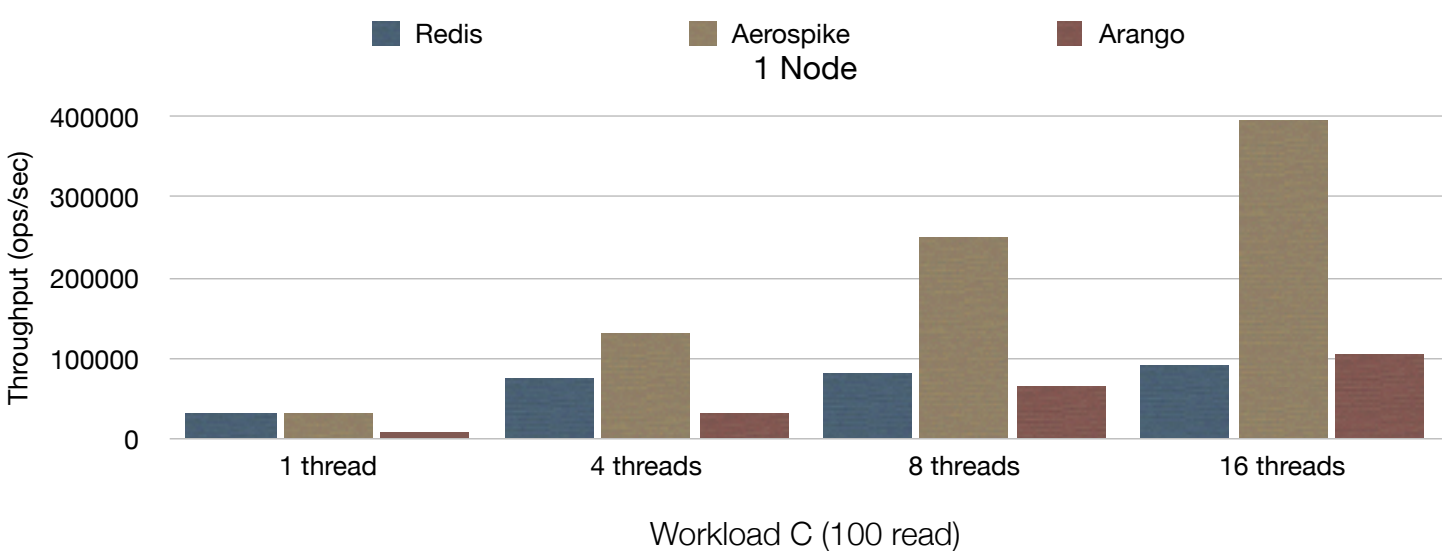
Αρχικά θα παρουσιάσουμε τα αποτελέσματα εκτελώντας κάθε βάση σε λειτουργία μονού κόμβου με τη Redis και την Aerospike να μην αποθηκεύουν τα δεδομένα στο δίσκο. Τα αποτελέσματα που συλλέγουμε για κάθε φόρτο εργασία όσο αφορα τη διεκπεραιωτική ικανότητα (throughput) είναι τα εξής:



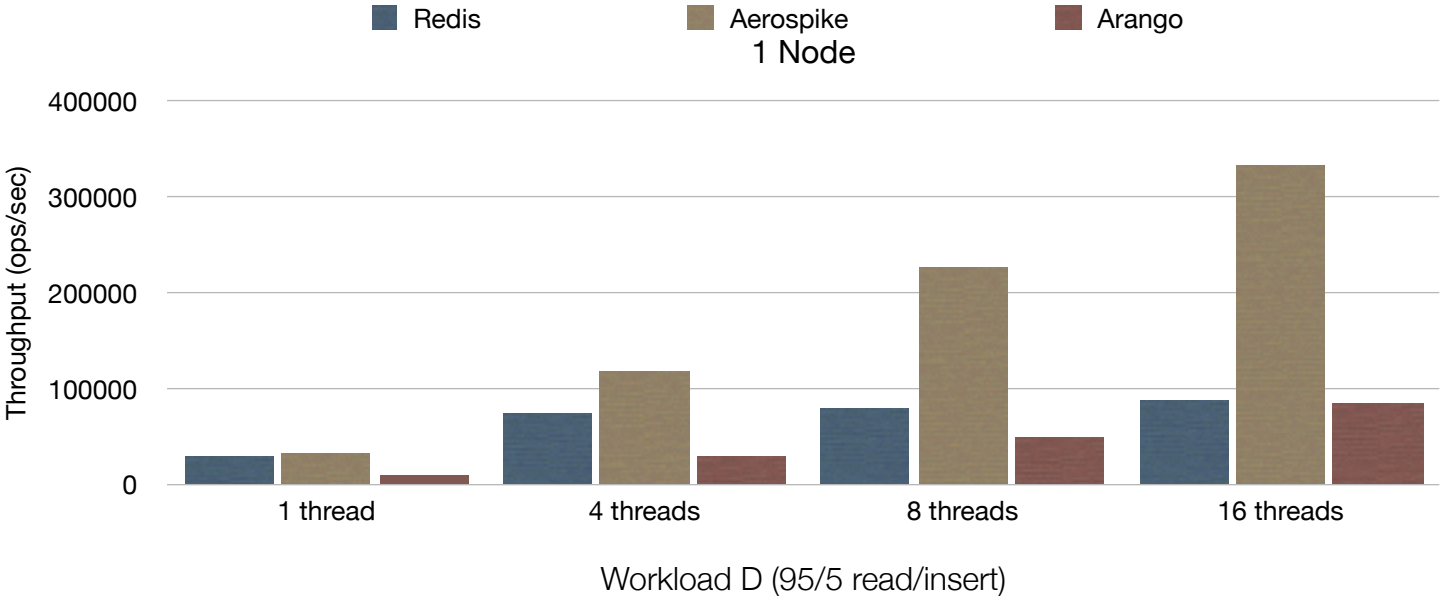
Διάγραμμα 1: Bare Metal Throughput 1 Node WorklaodA



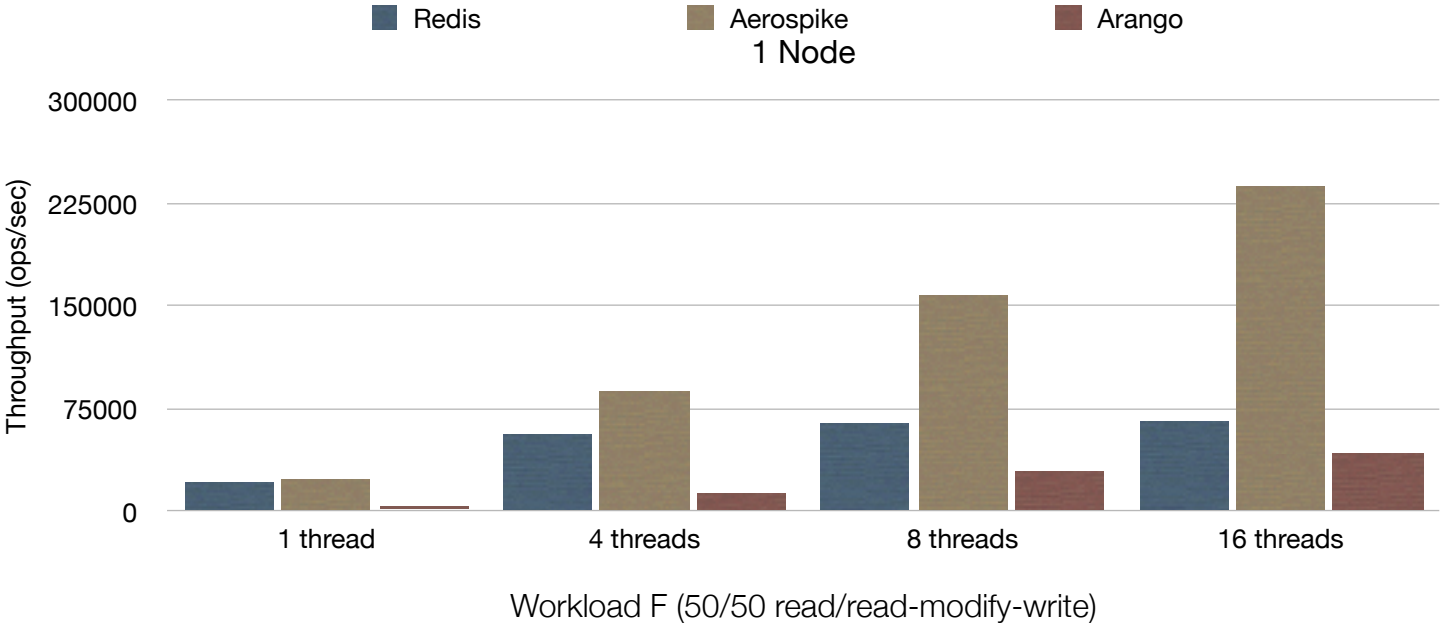
Διάγραμμα 2: Bare Metal Throughput 1 Node WorklaodB



Διάγραμμα 3: Bare Metal Throughput 1 Node WorklaodC

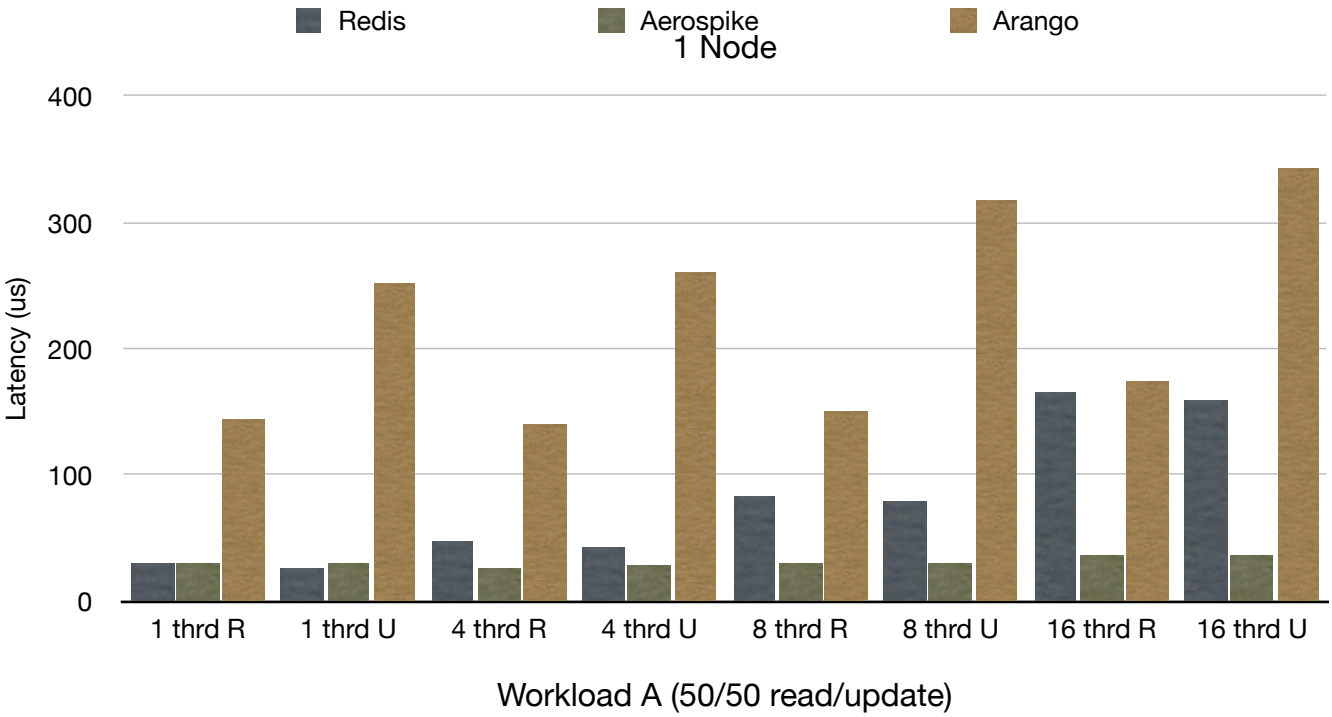


Διάγραμμα 4: Bare Metal Throughput 1 Node WorklaodD

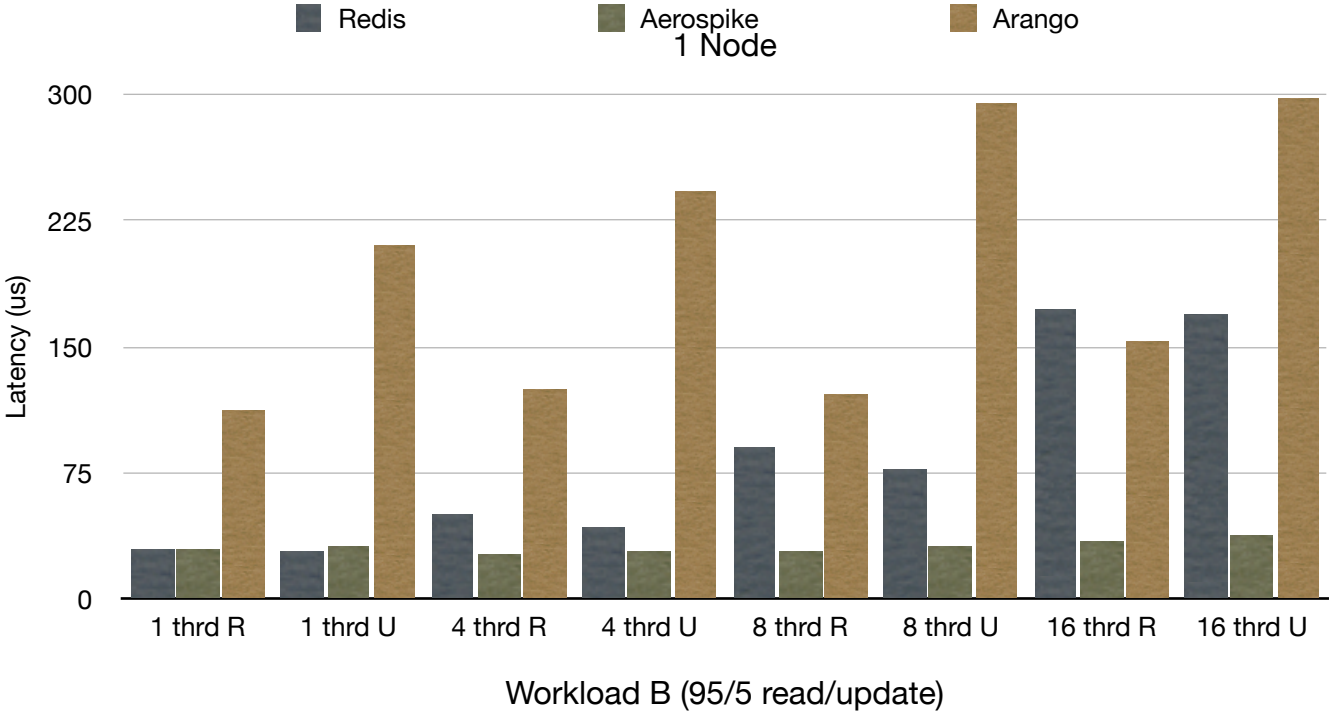


Διάγραμμα 5: Bare Metal Throughput 1 Node WorklaodF

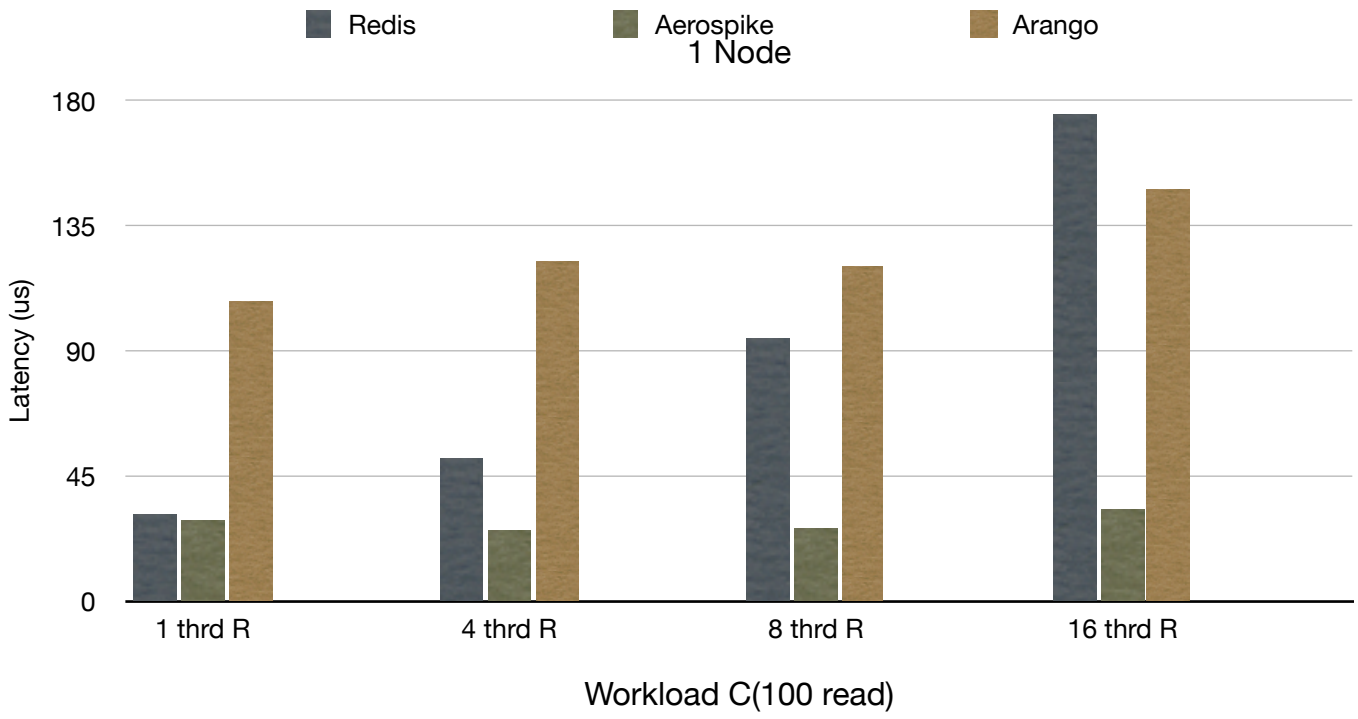
και για την χρόνο καθυστέρησης εκτέλεσης της κάθε εντολής (latency) έχουμε:



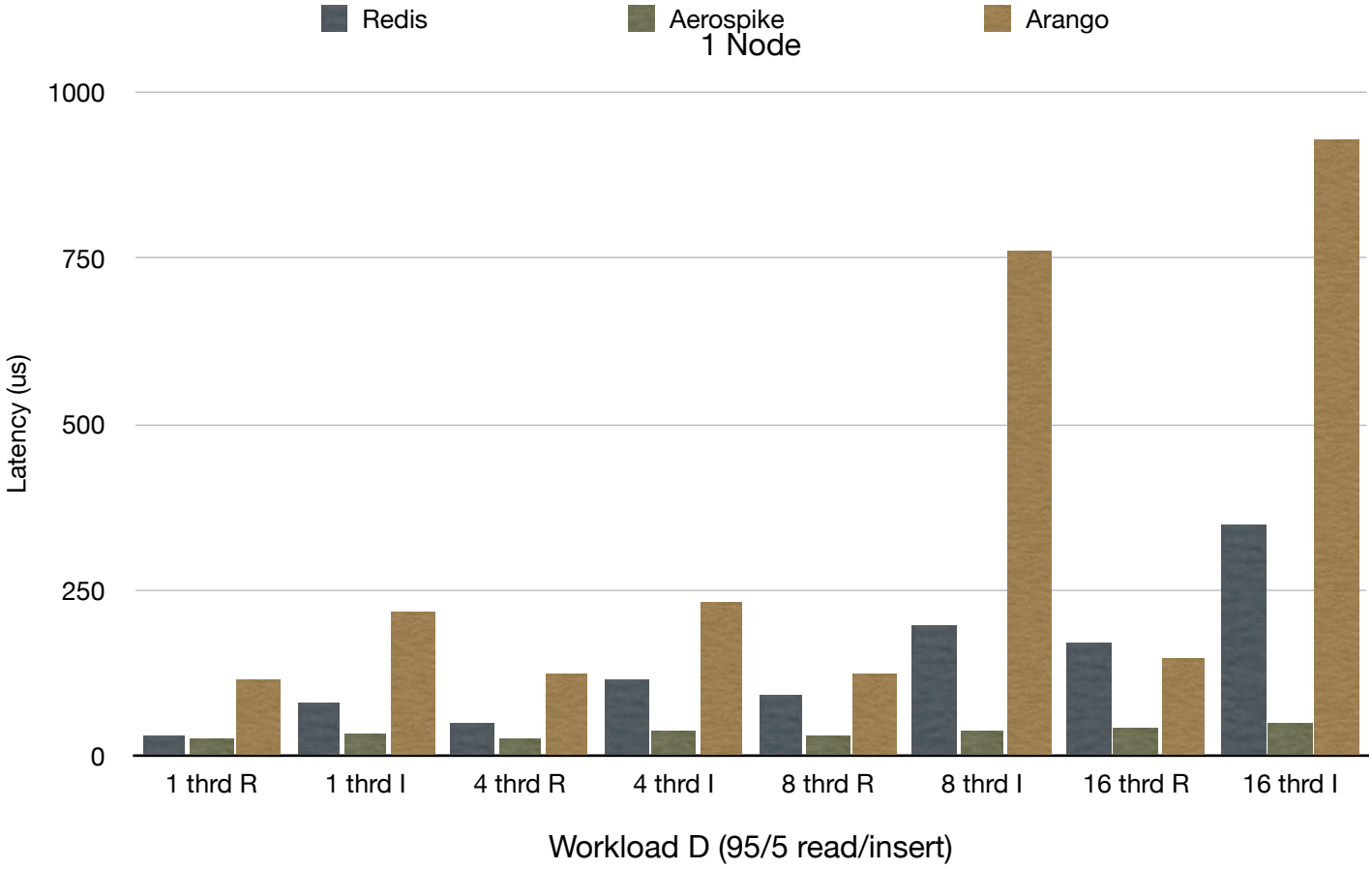
Διάγραμμα 6: Bare Metal Latency 1 Node WorklaodA



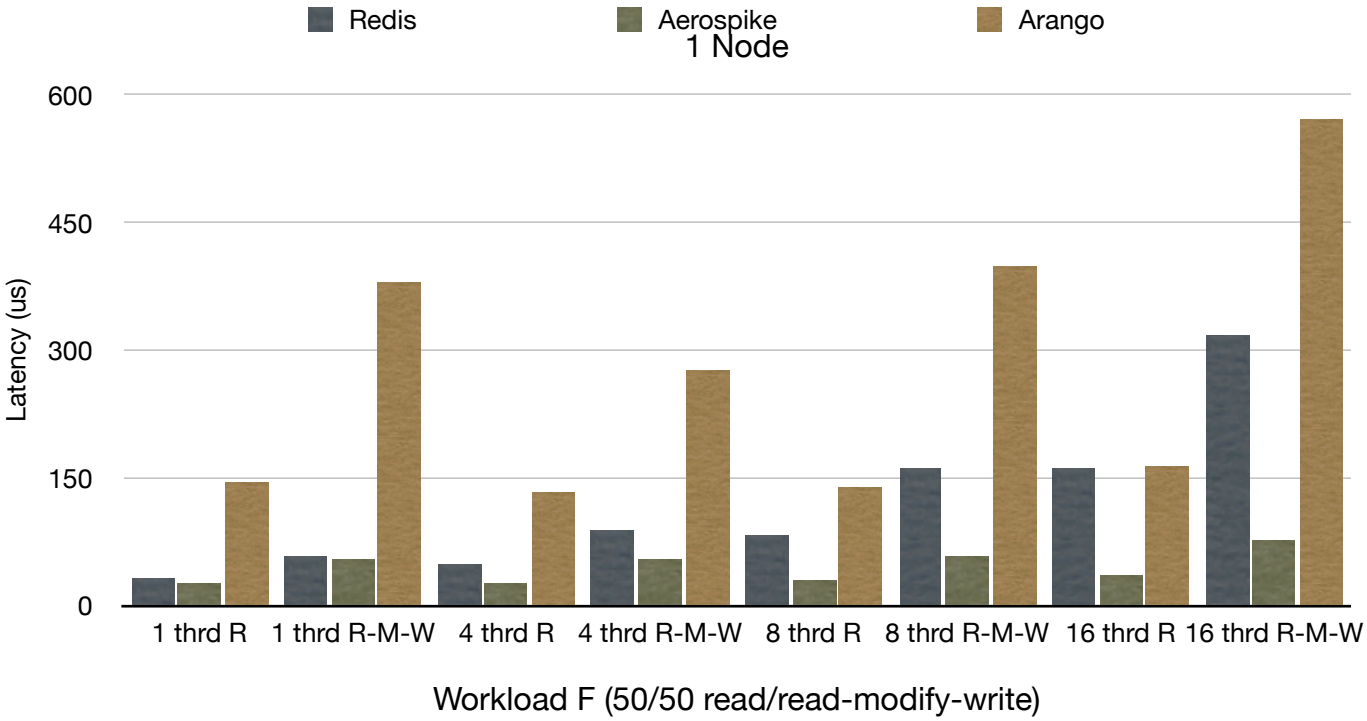
Διάγραμμα 7: Bare Metal Latency 1 Node WorklaodB



Διάγραμμα 8: Bare Metal Latency 1 Node WorklaodC



Διάγραμμα 9: Bare Metal Latency 1 Node WorklaodD

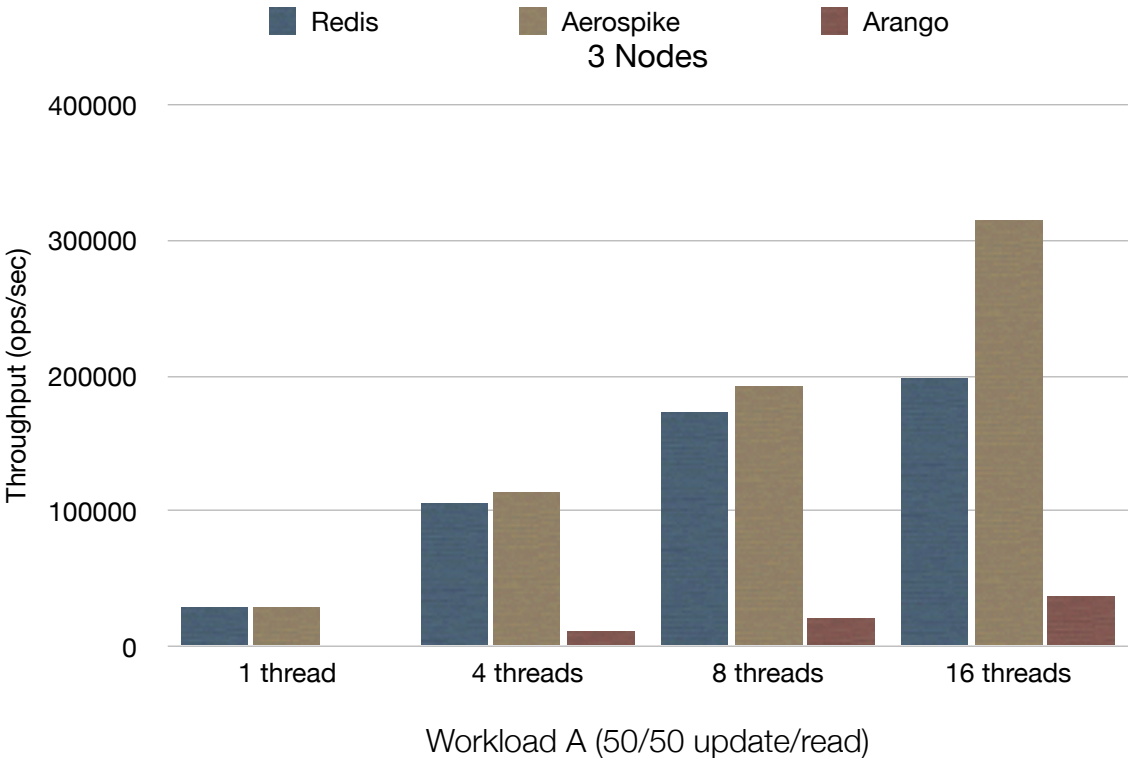


Διάγραμμα 10: Bare Metal Latency 1 Node WorklaodF

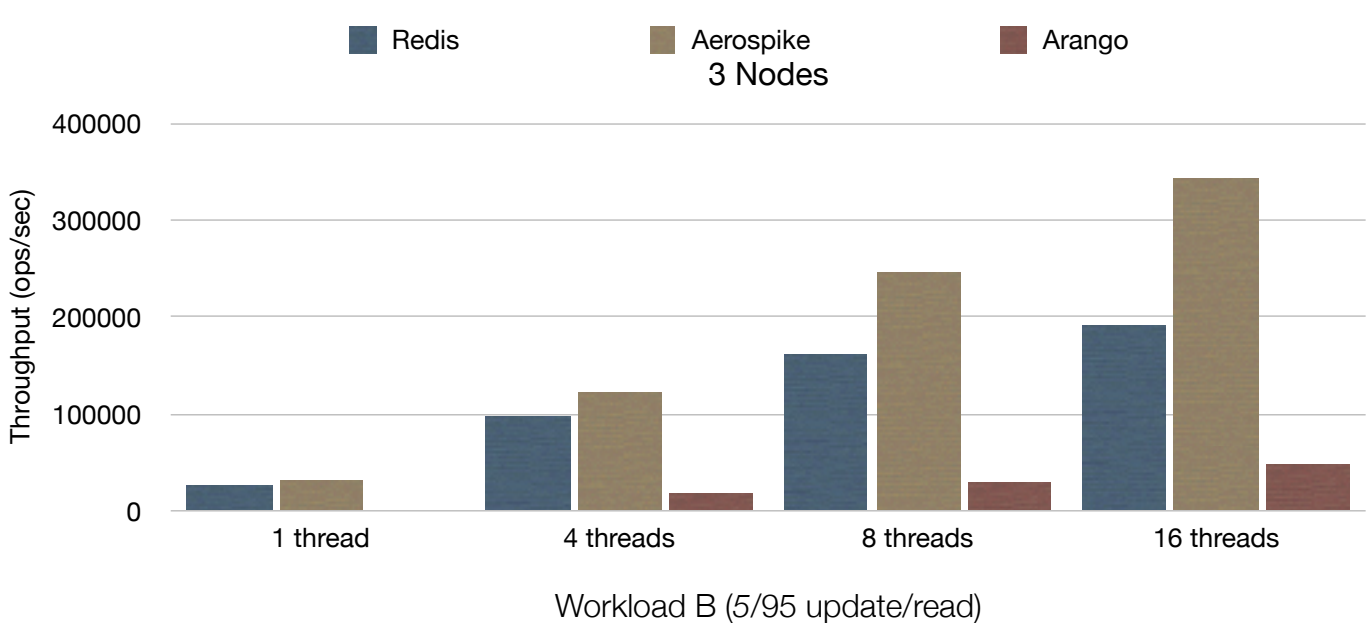
Από τα παραπάνω διαγράμματα παρατηρούμε ότι σε κάθε περίπτωση η Aerospike είναι η πιο γρήγορη βάση με την Redis και την ArangoDB να ποικίλουν ανάλογα με την περίπτωση. Πιο συγκεκριμένα παρατηρούμε ότι στο φόρτο εργασίας A (workload A) η Redis φτάνει εώς και 96,571 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 165 us και 168 us για τις εντολές ανάγνωσης (read) και τις εντολές ανανέωσης (update). Η Aerospike φτάνει εώς και 368,459 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 36 us και 36 us για τις εντολές ανάγνωσης (read) και τις εντολές ανανέωσης (update). Η ArangoDB φτάνει εώς και 60,324 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 174 us και 343 us για τις εντολές ανάγνωσης (read) και τις εντολές ανανέωσης (update). Για το φόρτο εργασίας B (workload B) η Redis φτάνει εώς και 90,514 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 173 us και 170 us για τις εντολές ανάγνωσης (read) και τις εντολές ανανέωσης (update). Η Aerospike φτάνει εώς και 383,240 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 35 us και 37 us για τις εντολές ανάγνωσης (read) και τις εντολές ανανέωσης (update). Η ArangoDB φτάνει εώς και 96,061 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 153 us και 297 us για τις εντολές ανάγνωσης (read) και τις εντολές ανανέωσης (update). Για το φόρτο εργασίας C (workload C) η Redis φτάνει εώς και 89,887 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 175 us για τις εντολές ανάγνωσης (read). Η Aerospike φτάνει εώς και 397,298 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 253 us για τις εντολές ανάγνωσης (read). Η ArangoDB φτάνει εώς και 103,626 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 148 us για τις εντολές ανάγνωσης (read). Για το φόρτο εργασίας D (workload D) η Redis φτάνει εώς και 86,873 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 171 us και 348 us για τις εντολές ανάγνωσης (read) και τις εντολές εισαγωγής (insert). Η Aerospike φτάνει εώς και 332,225 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 40 us και 48 us για τις εντολές ανάγνωσης (read) και τις εντολές εισαγωγής (insert). Η ArangoDB φτάνει εώς και 83,395 εντολές/δευτερόλεπτο με καθυστέρηση

εκτέλεσης εντολών 147 us και 927 us για τις εντολές ανάγνωσης (read) και τις εντολές εισαγωγής (insert). Για το φόρτο εργασίας F (workload F) η Redis φτάνει εώς και 66,155 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 160 us και 317 us για τις εντολές ανάγνωσης (read) και τις εντολές ανάγνωσης-τροποποίησης-εγγραφής(read-modify-write). Η Aerospike φτάνει εώς και 237,022 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 36 us και 75 us για τις εντολές ανάγνωσης (read) και τις εντολές ανάγνωσης-τροποποίησης-εγγραφής(read-modify-write). Η ArangoDB φτάνει εώς και 42,731 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 165 us και 570 us για τις εντολές ανάγνωσης (read) και τις εντολές ανάγνωσης-τροποποίησης-εγγραφής(read-modify-write). Από τα παραπάνω παρατηρούμε ότι η Aerospike μπορεί και κλιμακώνεται γραμμικά με την αύξηση των παράλληλων νημάτων εκτέλεσης. Αυτό είναι αναμενόμενο καθώς η Aerospike είναι ένα πολυνηματικής εκτέλεσης σύστημα (multi-threaded) και έχει σχεδιαστεί ειδικά για να προσφέρει πολύ καλή κλιμακωσιμότητα. Η Redis παρατηρούμε ότι δεν κλιμακώνεται γραμμικά. Για την ακρίβεια παρατηρούμε ότο όσο αυξάνουμε τα παράλληλα νήματα εκτέλεσης, η αύξηση της απόδοσης της είναι όλο και μικρότερη. Αυτό μπορεί να εξηγηθεί από το γεγονός ότι πρόκειται για ένα σύστημα μονής νηματικής εκτέλεσης (single-threaded). Η Arango επίσης εμφανίζει γραμμική κλιμακωσιμότητα για τους ίδιους λόγους με την Aerospike.

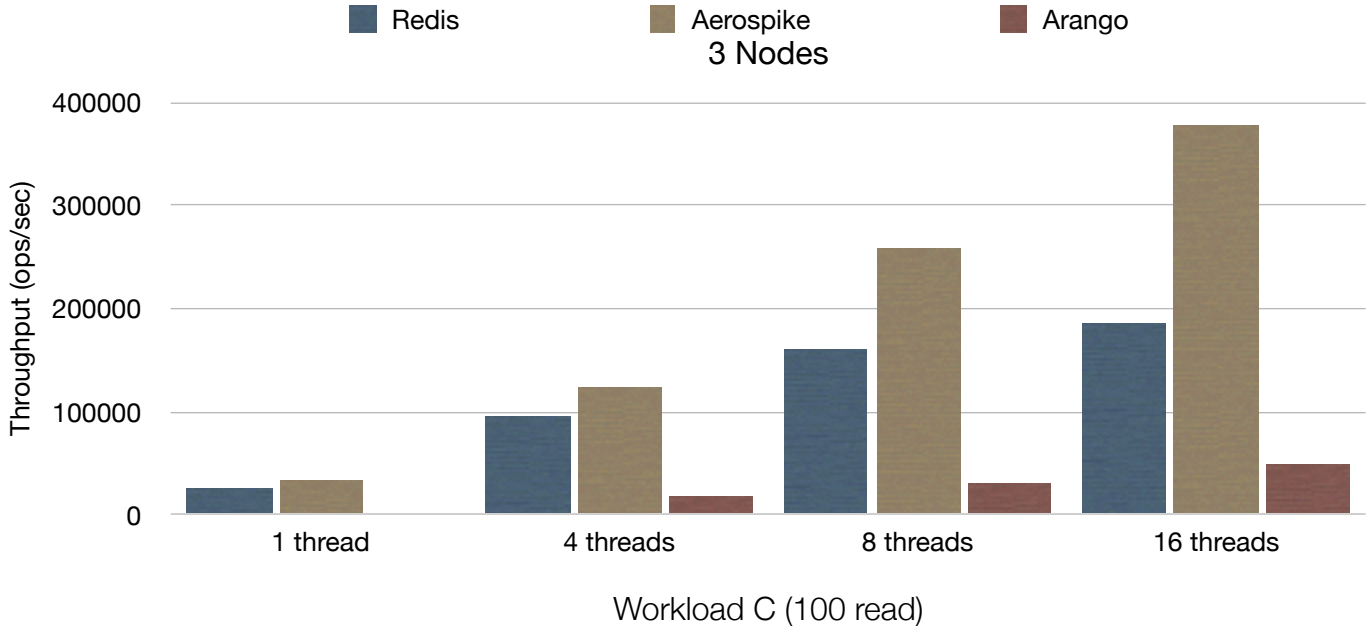
Συνεχίζοντας τα πειράματα μας η επόμενη διάταξη είναι αυτή του συμπλέγματος 3 κόμβων. Σε αυτή την περίπτωση επειδή η Redis προυποθέτει τουλάχιστον 6 κόμβους για τη δημιουργία ενός συμπλέγματος, σηκώσαμε 6 κόμβους, φτιάξαμε το σύμπλεγμα και έπειτα σβήσαμε 3 κόμβους με αποτέλεσμα να έχουμε ένα σύμπλεγμα με μόνο 3 κύριους κόμβους χωρίς κανένα υποτελή (3 masters). Τα αποτελέσματα που πήραμε για όλες τις βάσεις είναι τα εξής:



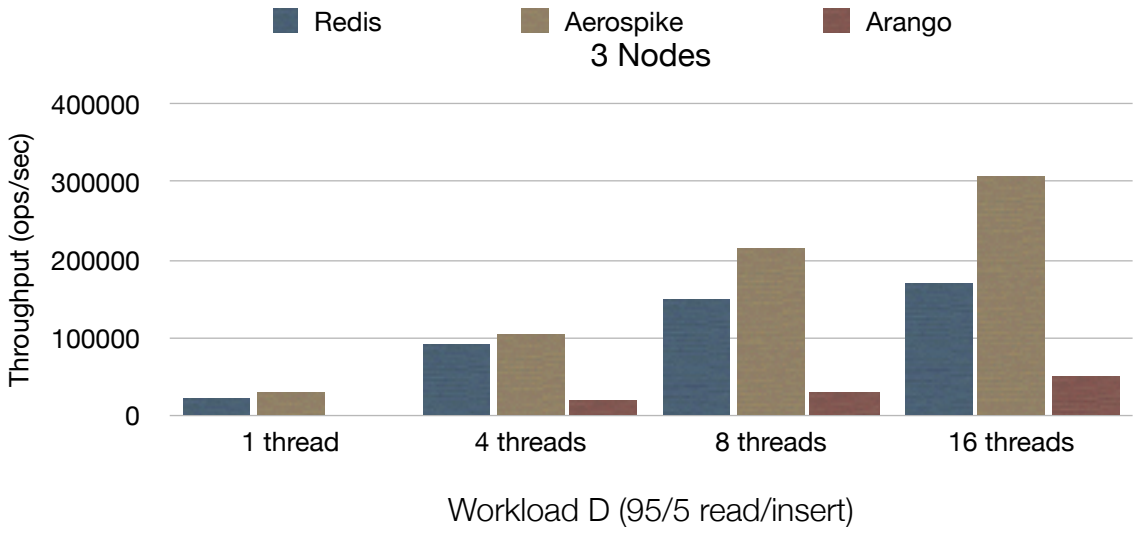
Διάγραμμα 11: Bare Metal Throughput 3 Nodes WorklaodA



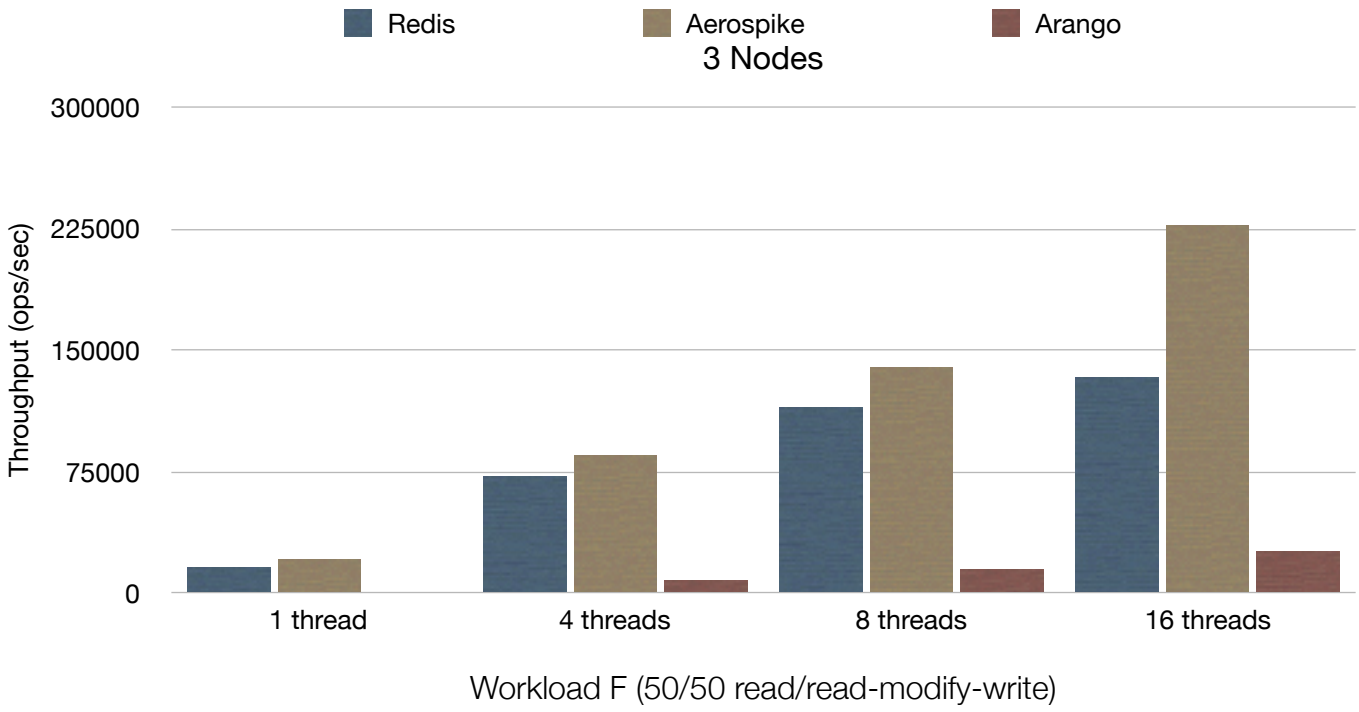
Διάγραμμα 12: Bare Metal Throughput 3 Nodes WorklaodB



Διάγραμμα 13: Bare Metal Throughput 3 Nodes WorklaodC

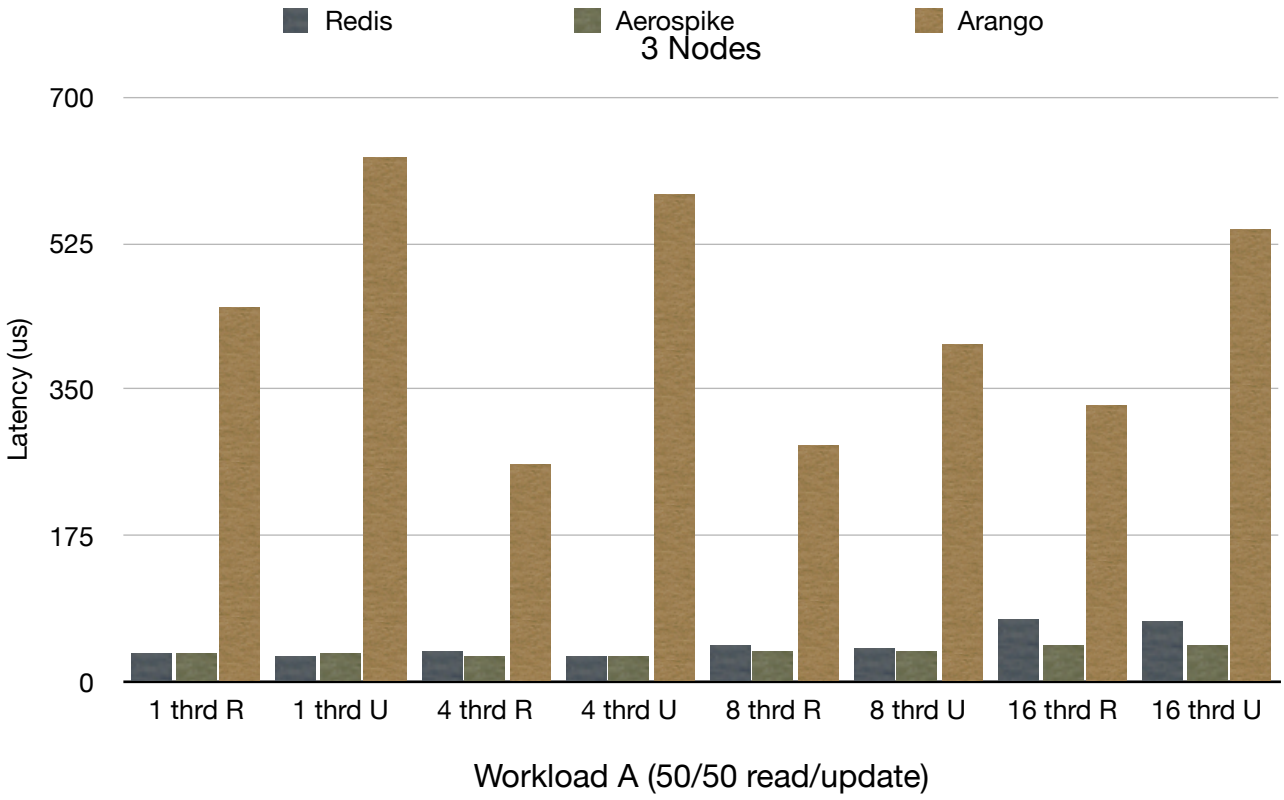


Διάγραμμα 14: Bare Metal Throughput 3 Nodes WorklaodD

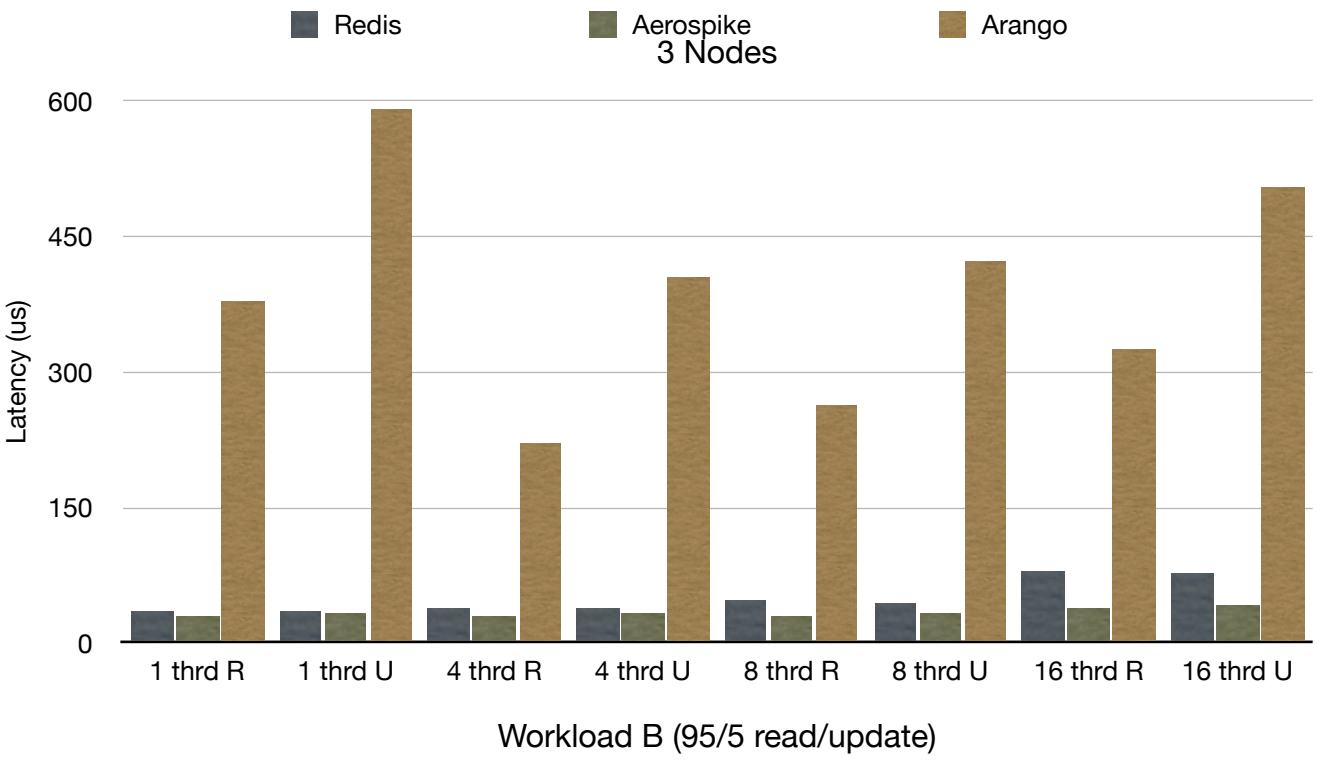


Διάγραμμα 15: Bare Metal Throughput 3 Nodes WorklaodF

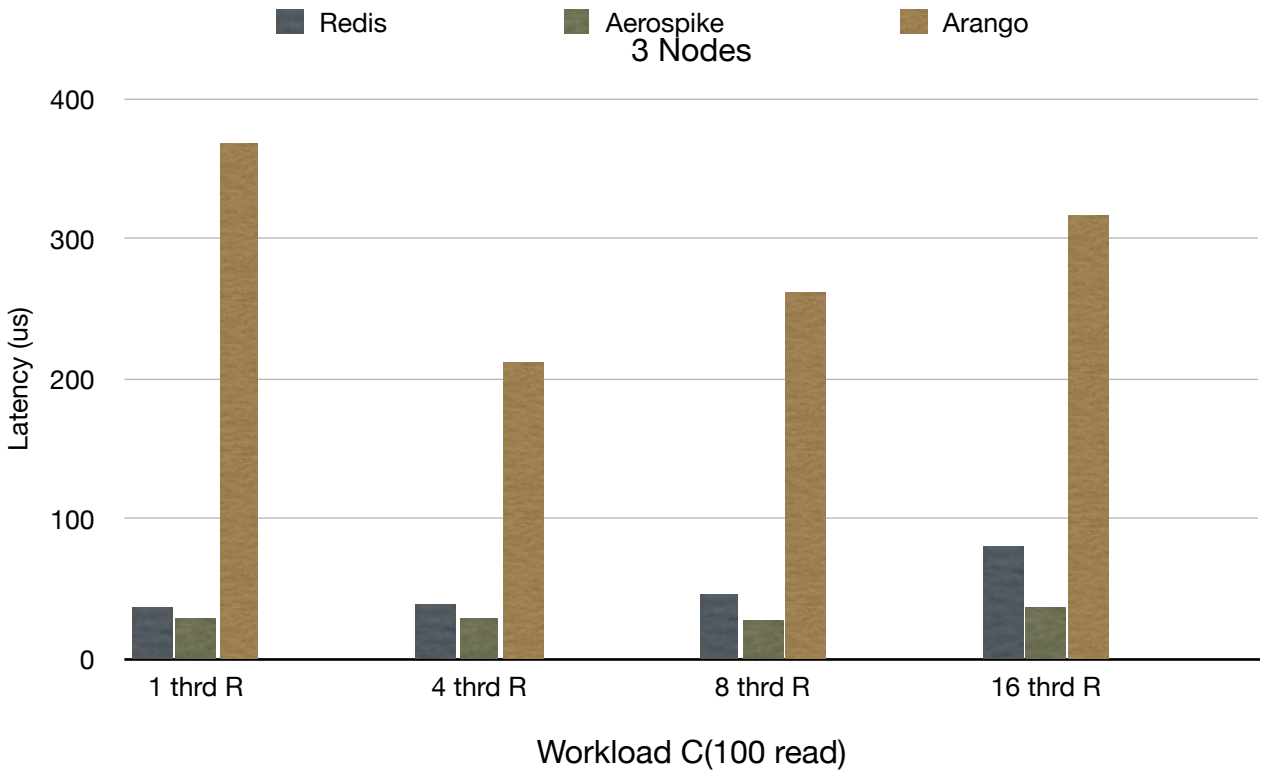
και για την χρόνο καθυστέρησης εκτέλεσης της κάθε εντολής (latency) έχουμε:



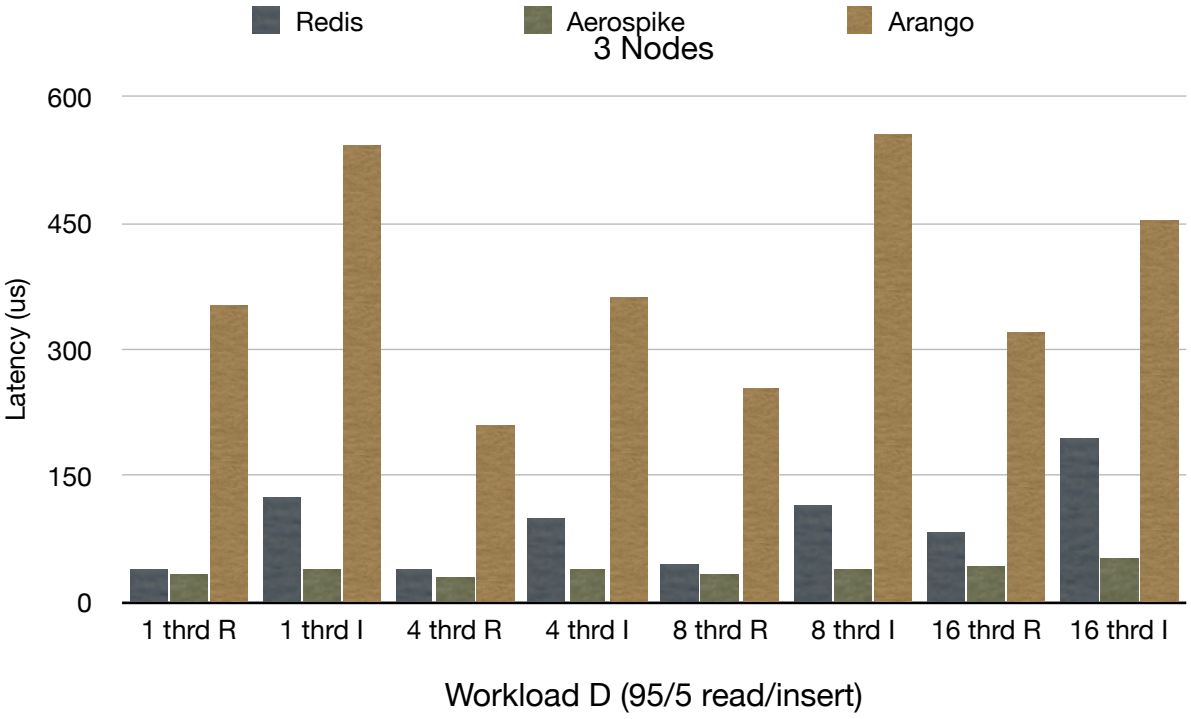
Διάγραμμα 16: Bare Metal Latency 3 Nodes WorklaodA



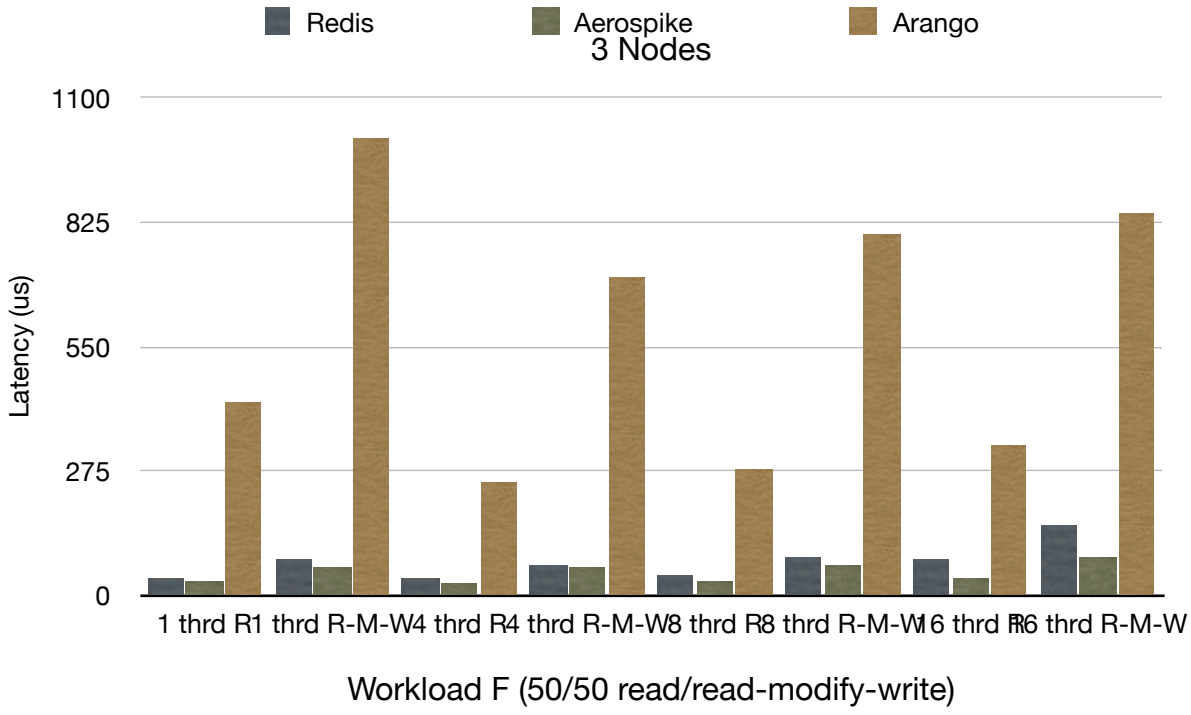
Διάγραμμα 17: Bare Metal Latency 3 Nodes WorklaodB



Διάγραμμα 18: Bare Metal Latency 3 Nodes WorklaodC



Διάγραμμα 19: Bare Metal Latency 3 Nodes WorklaodD



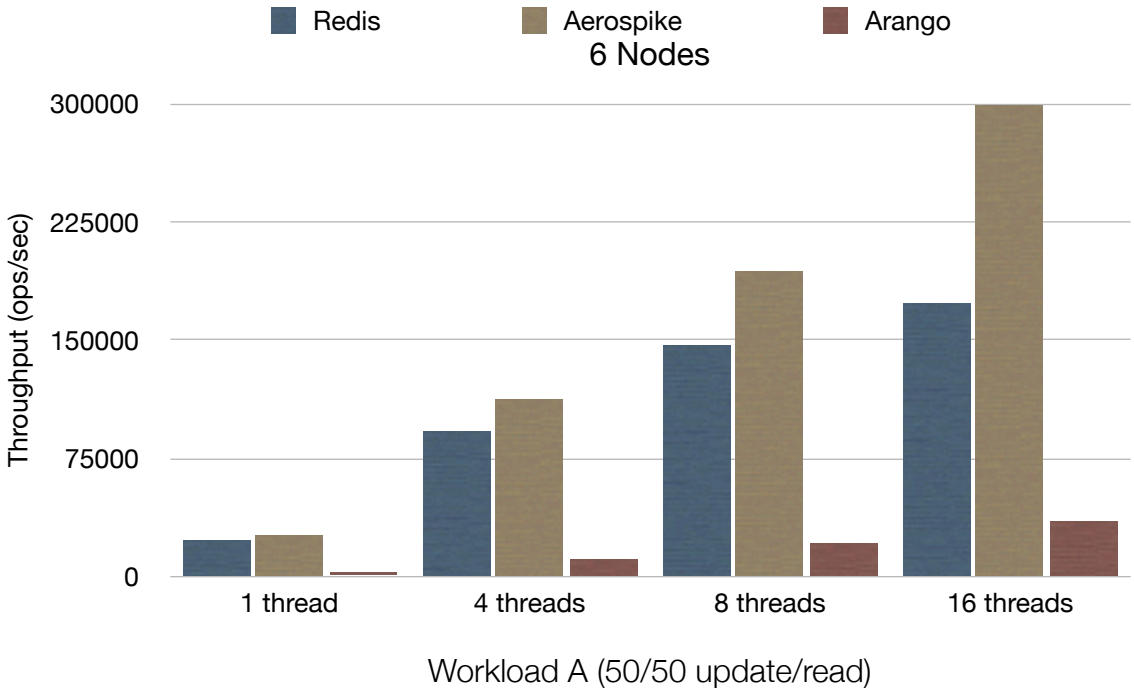
Διάγραμμα 20: Bare Metal Latency 3 Nodes WorklaodF

Από τα παραπάνω διαγράμματα παρατηρούμε πάλι ότι σε κάθε περίπτωση η Aerospike είναι η πιο γρήγορη βάση με την Redis να είναι η δεύτερη πιο γρήγορη και τέλος την ArangoDB. Πιο συγκεκριμένα παρατηρούμε ότι στο φόρτο εργασίας A (workload A) η Redis φτάνει εώς και 197,122 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 77 us και 72 us για τις εντολές

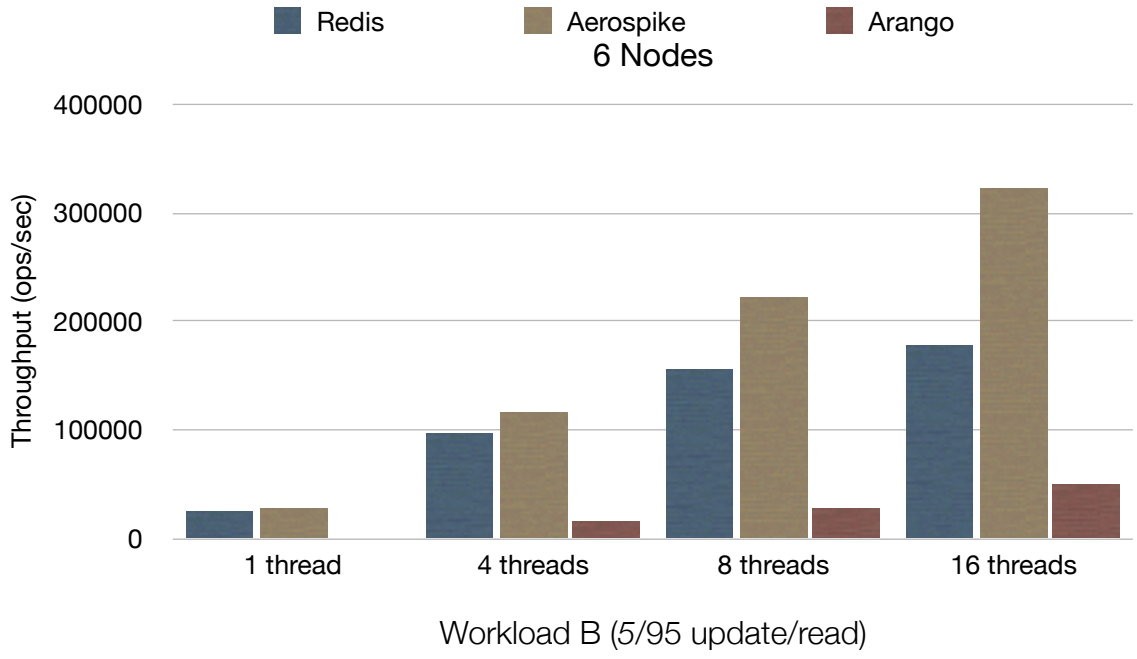
ανάγνωσης (read) και τις εντολές ανανέωσης (update). Η Aerospike φτάνει εώς και 313,479 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 43 us και 43 us για τις εντολές ανάγνωσης (read) και τις εντολές ανανέωσης (update). Η ArangoDB φτάνει εώς και 36,020 εντολές/ δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 332 us και 541 us για τις εντολές ανάγνωσης (read) και τις εντολές ανανέωσης (update). Για το φόρτο εργασίας B (workload B) η Redis φτάνει εώς και 190,367 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 78 us και 76 us για τις εντολές ανάγνωσης (read) και τις εντολές ανανέωσης (update). Η Aerospike φτάνει εώς και 341,296 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 39 us και 40 us για τις εντολές ανάγνωσης (read) και τις εντολές ανανέωσης (update). Η ArangoDB φτάνει εώς και 47,265 εντολές/ δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 324 us και 505 us για τις εντολές ανάγνωσης (read) και τις εντολές ανανέωσης (update). Για το φόρτο εργασίας C (workload C) η Redis φτάνει εώς και 184,945 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 80 us για τις εντολές ανάγνωσης (read). Η Aerospike φτάνει εώς και 376,931 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 36 us για τις εντολές ανάγνωσης (read). Η ArangoDB φτάνει εώς και 49,573 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 317 us για τις εντολές ανάγνωσης (read). Για το φόρτο εργασίας D (workload D) η Redis φτάνει εώς και 169,923 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 82 us και 195 us για τις εντολές ανάγνωσης (read) και τις εντολές εισαγωγής (insert). Η Aerospike φτάνει εώς και 306,654 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 43 us και 51 us για τις εντολές ανάγνωσης (read) και τις εντολές εισαγωγής (insert). Η ArangoDB φτάνει εώς και 47,986 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 321 us και 554 us για τις εντολές ανάγνωσης (read) και τις εντολές εισαγωγής (insert). Για το φόρτο εργασίας F (workload F) η Redis φτάνει εώς και 133,120 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 76 us και 153 us για τις εντολές ανάγνωσης (read) και τις εντολές ανάγνωσης-τροποποίησης-εγγραφής(read-modify-write). Η Aerospike φτάνει εώς και 226,193 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 40 us και 83 us για τις εντολές ανάγνωσης (read) και τις εντολές ανάγνωσης-τροποποίησης-εγγραφής(read-modify-write). Η ArangoDB φτάνει εώς και 26,793 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 334 us και 845 us για τις εντολές ανάγνωσης (read) και τις εντολές ανάγνωσης-τροποποίησης-εγγραφής(read-modify-write). Παρατηρούμε ότι η Redis έχει υπερδιπλασιάσει την επίδοσή της σε σχέση με τη λειτουργία μονού κόμβου. Το αποτέλεσμα αυτό ήταν αναμενόμενο καθώς ο φόρτος εργασίας και τα παράλληλα νήματα εκτέλεσης μοιράζονται πλέον σε 3 κόμβους με αποτέλεσμα περισσότερη διεκπεραιωτική απόδοση καθώς η single threaded αρχιτεκτονική της μπορεί να "προσομοιωθεί" ως multithreaded προσθέτοντας περαιτέρω κόμβους στο σύμπλεγμα. Από την άλλη πλευρά παρατηρούμε ότι η επίδοση της Aerospike έχει μειωθεί ελάχιστα. Και αυτό είναι αναμενόμενο καθώς κάθε κόμβος πλέον είναι και υπεύθυνος και για άλλες λειτουργίες, όπως τη λειτουργία heartbeat με την οποία οι κόμβοι επικοινωνούν συνεχώς μεταξύ τους για να προσδιορίσουν την κατάσταση των υπόλοιπων κόμβων του συμπλέγματος όπως επίσης και τη λειτουργία του μεταξύ τους συντονισμού όσον αφορά τα δεδομένα. Στην ArangoDB παρατηρούμε

μείωση καθώς πλέον τα δεδομένα είναι χωρισμένα σε διάφορους κόμβους και η ανάκτηση τους και η επεξεργασία τους είναι πιο αργές διαδικασίες.

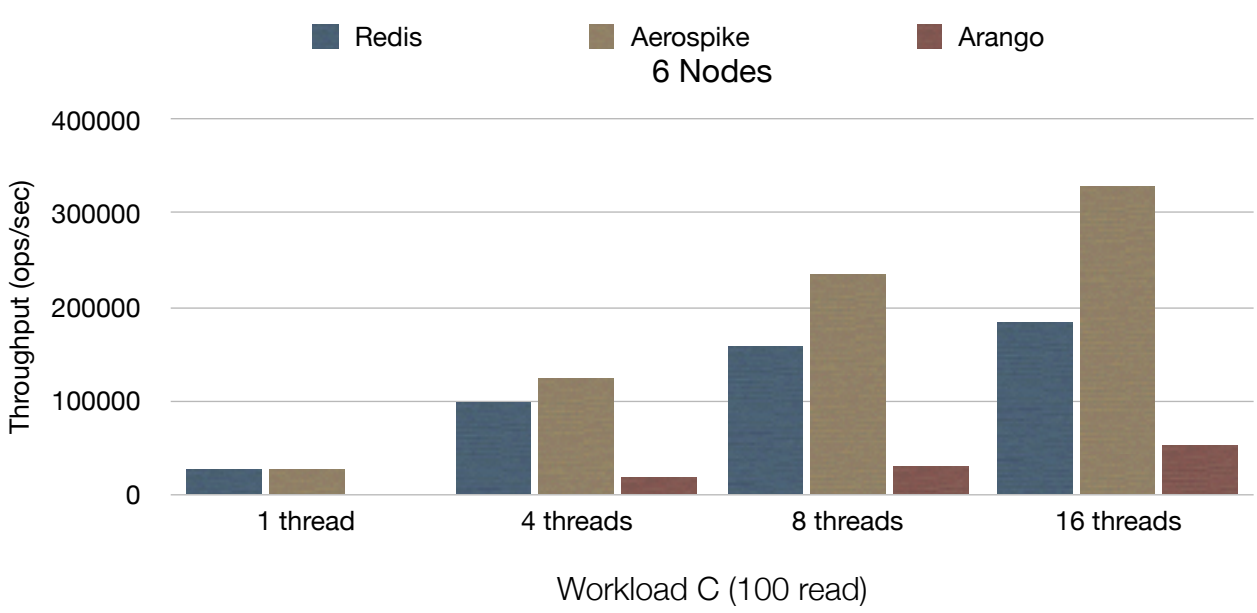
Συνεχίζοντας τα πειράματα μας η επόμενη διάταξη είναι αυτή του συμπλέγματος 6 κόμβων. Σε αυτή την περίπτωση στη Redis σε κάθε κύριο κόμβο αντιστοιχίζεται ένας υποτελής. Τα αποτελέσματα που συλλέγουμε για κάθε φόρτο εργασία όσο αφορα τη διεκπεραιωτική ικανότητα (throughput) είναι τα εξής:



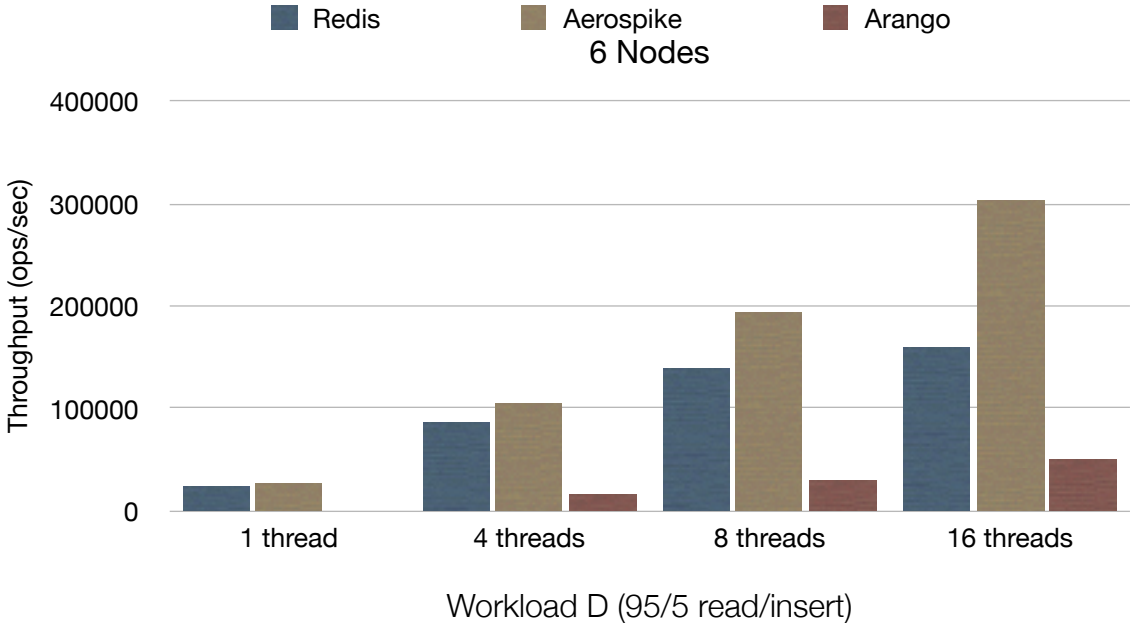
Διάγραμμα 21: Bare Metal Throughput 6 Nodes WorklaodA



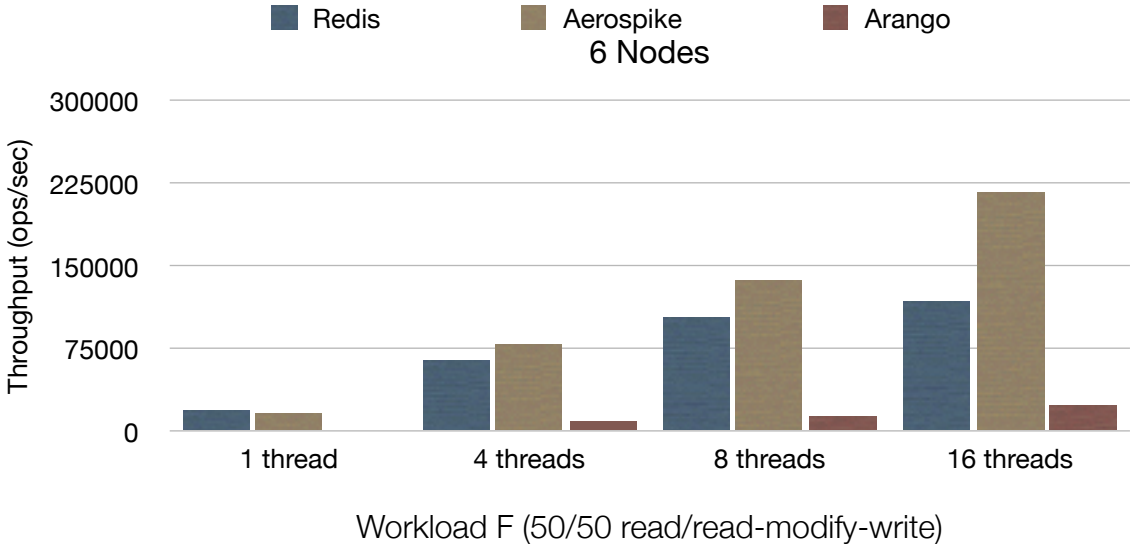
Διάγραμμα 22: Bare Metal Throughput 6 Nodes WorklaodB



Διάγραμμα 23: Bare Metal Throughput 6 Nodes WorklaodC

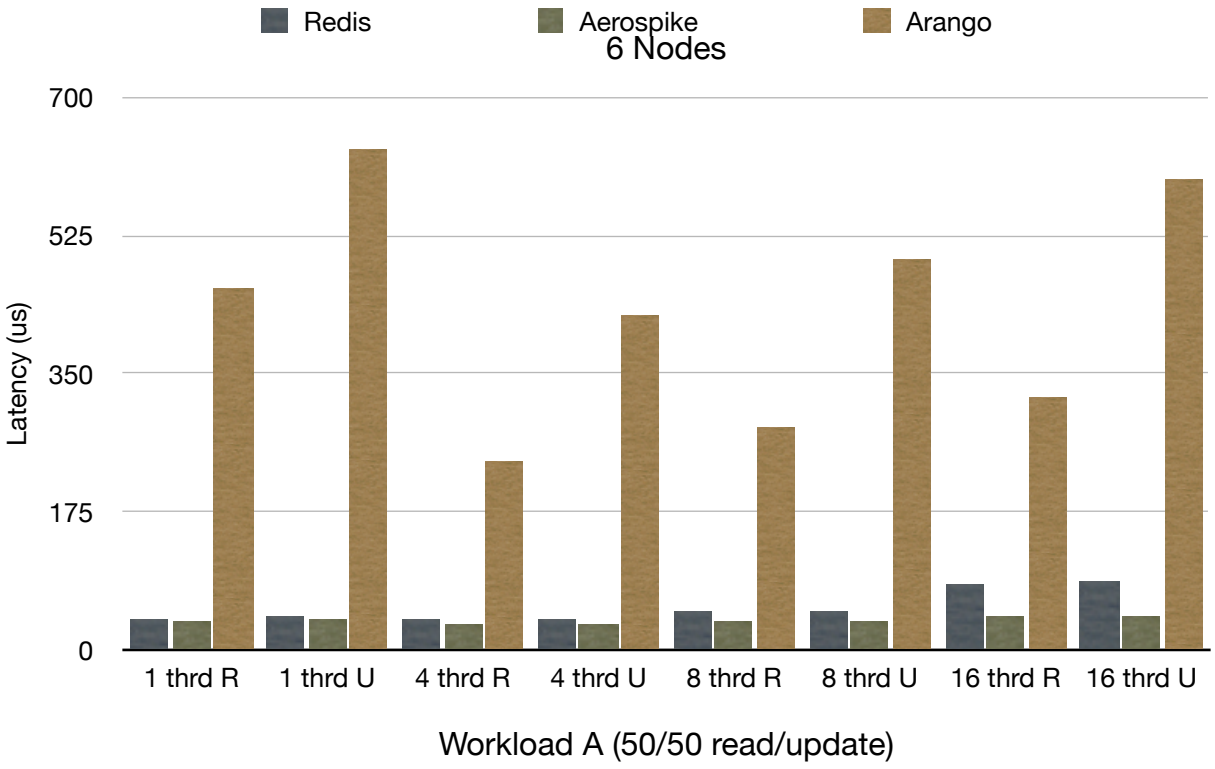


Διάγραμμα 24: Bare Metal Throughput 6 Nodes WorklaodD

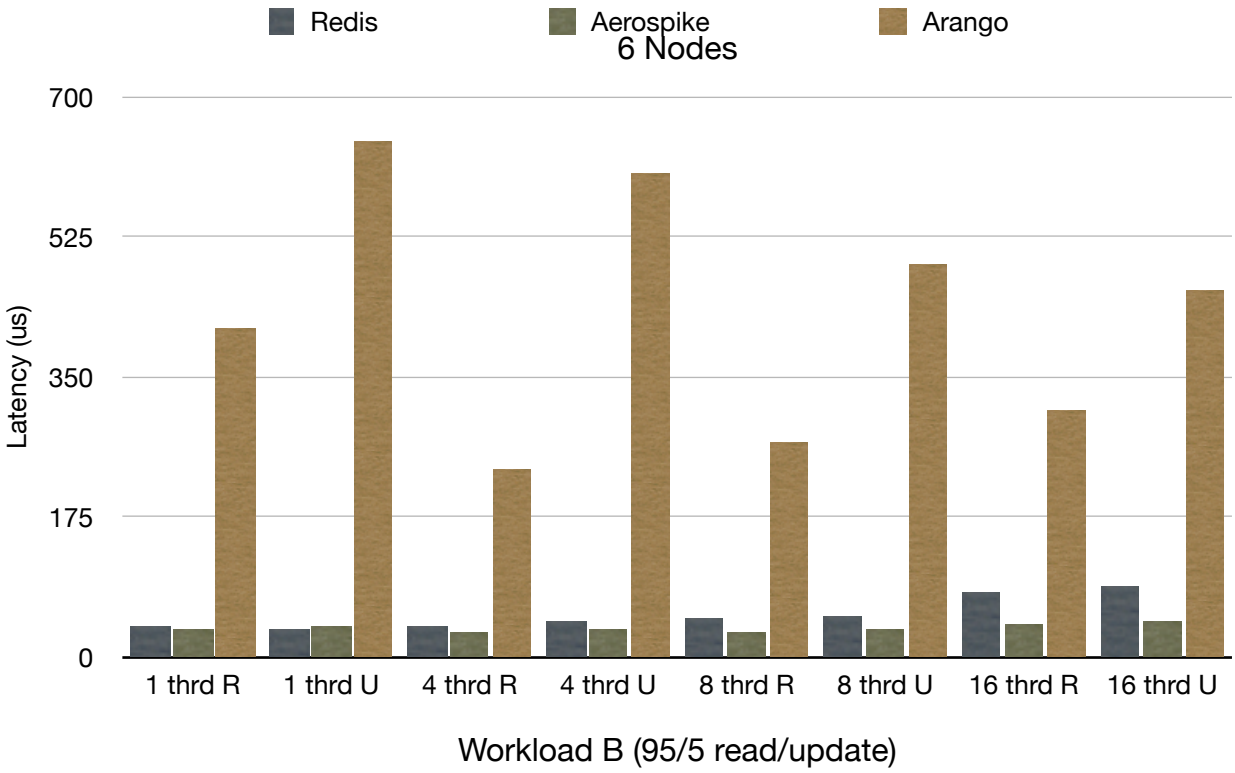


Διάγραμμα 25: Bare Metal Throughput 6 Nodes WorklaodF

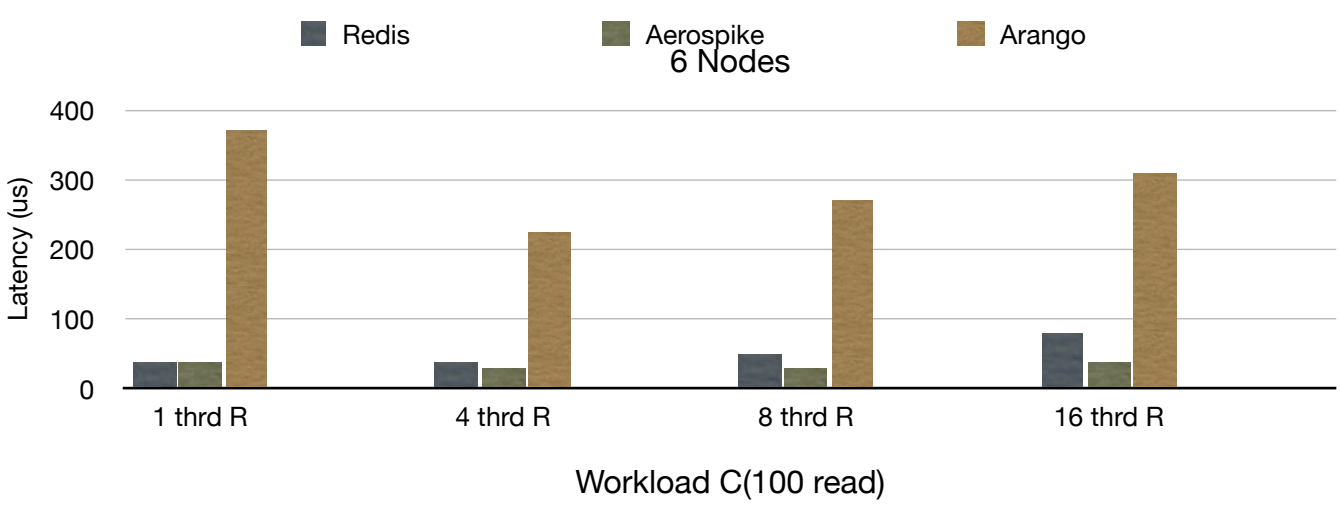
και για την χρόνο καθυστέρησης εκτέλεσης της κάθε εντολής (latency) έχουμε:



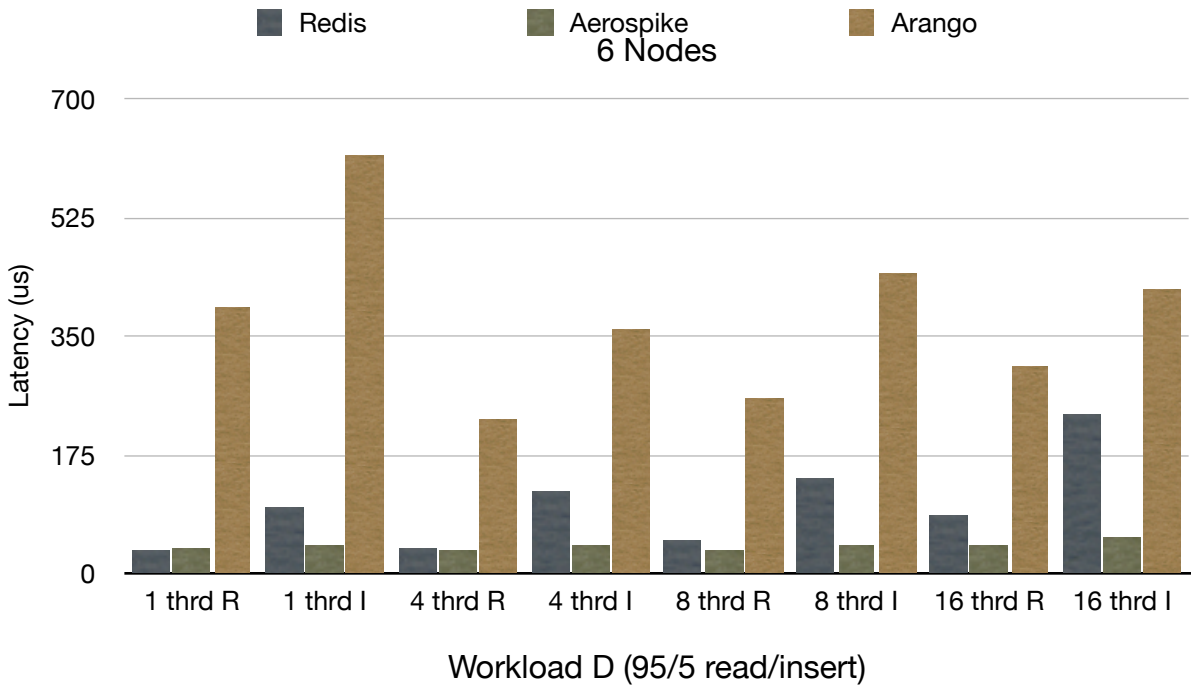
Διάγραμμα 26: Bare Metal Latency 6 Nodes WorklaodA



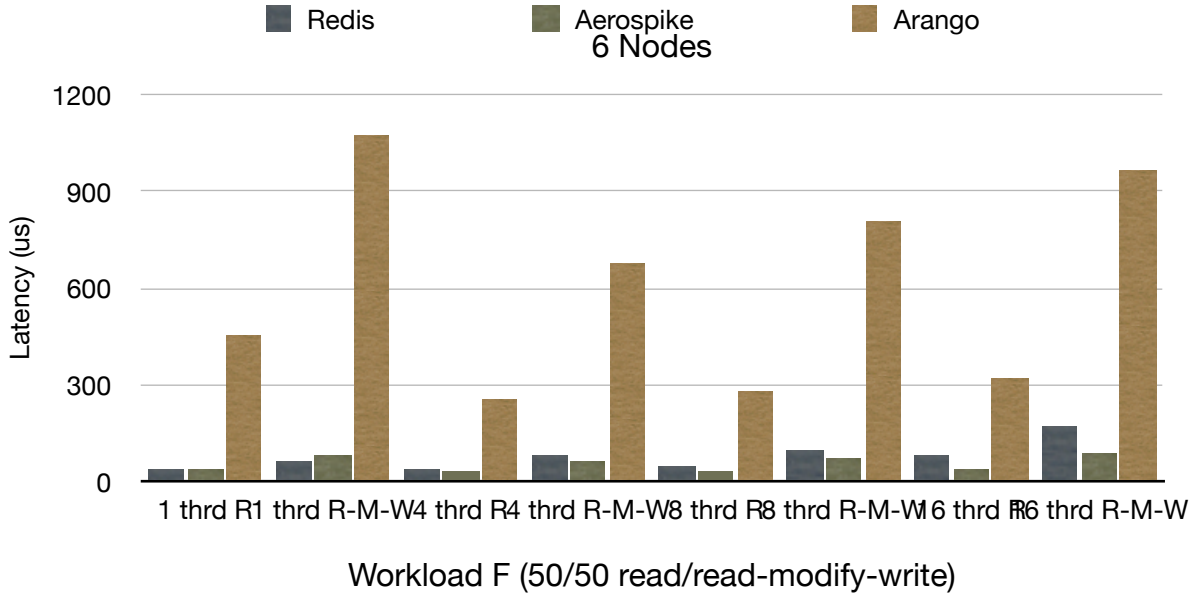
Διάγραμμα 27: Bare Metal Latency 6 Nodes WorklaodB



Διάγραμμα 28: Bare Metal Latency 6 Nodes WorklaodC



Διάγραμμα 29: Bare Metal Latency 6 Nodes WorklaodD

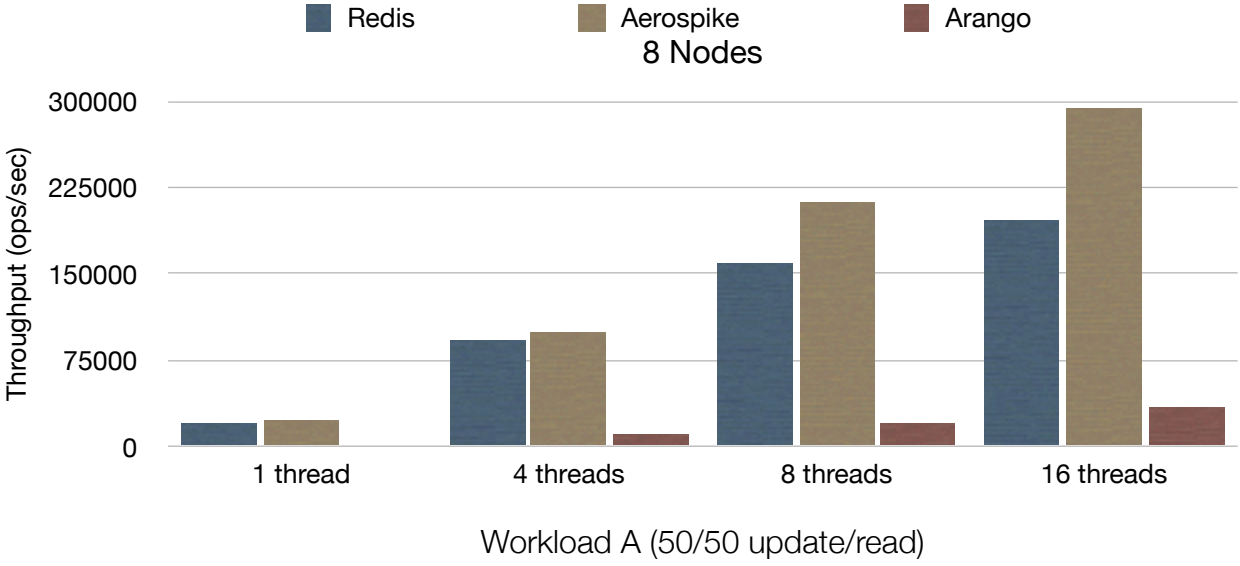


Διάγραμμα 30: Bare Metal Latency 6 Nodes WorklaodF

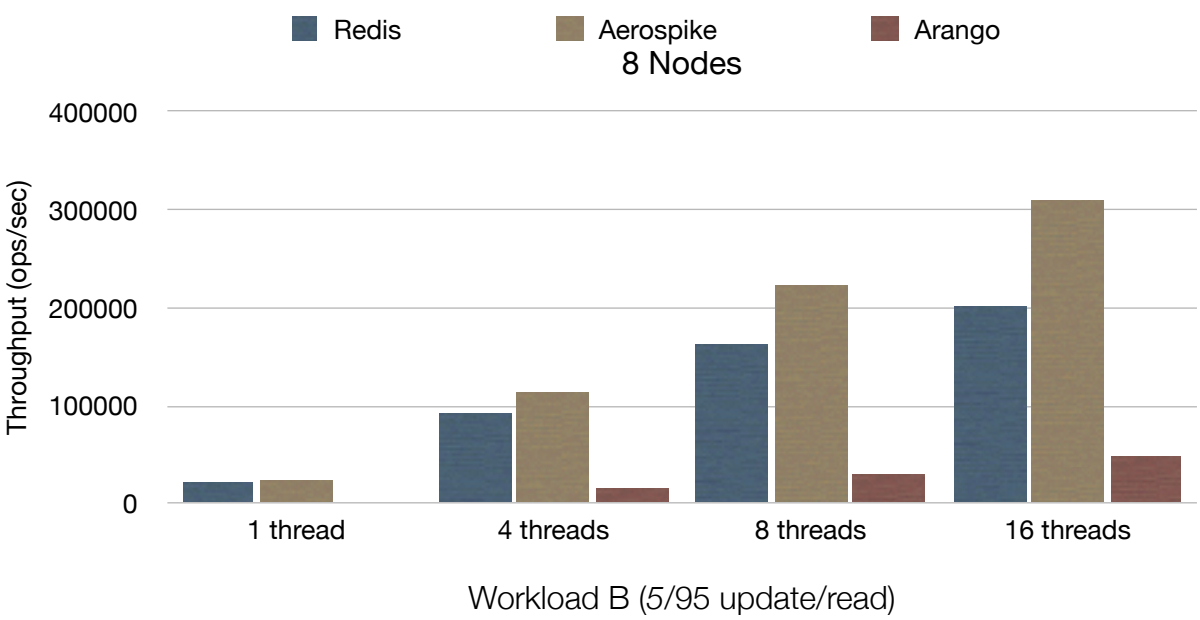
Από τα παραπάνω διαγράμματα παρατηρούμε πάλι ότι σε κάθε περίπτωση η Aerospike είναι η πιο γρήγορη βάση με την Redis να είναι η δεύτερη πιο γρήγορη και τέλος την ArangoDB. Πιο συγκεκριμένα παρατηρούμε ότι στο φόρτο εργασίας A (workload A) η Redis φτάνει εώς και 172,711 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 73 us και 76 us για τις εντολές ανάγνωσης (read) και τις εντολές ανανέωσης (update). Η Aerospike φτάνει εώς και 297,796 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 43 us και 43 us για τις εντολές ανάγνωσης (read) και τις εντολές ανανέωσης (update). Η ArangoDB φτάνει εώς και 34,304 εντολές/ δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 321 us και 596 us για τις εντολές ανάγνωσης (read) και τις εντολές ανανέωσης (update). Για το φόρτο εργασίας B (workload B) η Redis φτάνει εώς και 176,273 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 82 us και 87 us για τις εντολές ανάγνωσης (read) και τις εντολές ανανέωσης (update). Η Aerospike φτάνει εώς και 322,893 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 40 us και 43 us για τις εντολές ανάγνωσης (read) και τις εντολές ανανέωσης (update). Η ArangoDB φτάνει εώς και 49,654 εντολές/ δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 309 us και 458 us για τις εντολές ανάγνωσης (read) και τις εντολές ανανέωσης (update). Για το φόρτο εργασίας C (workload C) η Redis φτάνει εώς και 181,752 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 81 us για τις εντολές ανάγνωσης (read). Η Aerospike φτάνει εώς και 329,706 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 38 us για τις εντολές ανάγνωσης (read). Η ArangoDB φτάνει εώς και 51,080 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 308 us για τις εντολές ανάγνωσης (read). Για το φόρτο εργασίας D (workload D) η Redis φτάνει εώς και 158,856 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 86 us και 235 us για τις εντολές ανάγνωσης (read) και τις εντολές εισαγωγής (insert). Η Aerospike φτάνει εώς και 304,043 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 43 us και 55 us για τις εντολές ανάγνωσης (read) και τις εντολές εισαγωγής (insert). Η ArangoDB φτάνει εώς και 50,576 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 305 us και 419 us για τις εντολές ανάγνωσης (read) και τις εντολές εισαγωγής (insert). Για το φόρτο εργασίας F (workload F) η Redis φτάνει εώς και 119,388 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 82 us και 168 us για τις εντολές ανάγνωσης (read) και τις εντολές ανάγνωσης-τροποποίησης-εγγραφής(read-modify-write). Η Aerospike φτάνει εώς και 217,060 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 43 us και 85 us για τις εντολές ανάγνωσης (read) και τις εντολές ανάγνωσης-τροποποίησης-εγγραφής(read-modify-write). Η ArangoDB φτάνει εώς και 24,669 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 317 us και 967 us για τις εντολές ανάγνωσης (read) και τις εντολές ανάγνωσης-τροποποίησης-εγγραφής(read-modify-write). Από τα παραπάνω αποτελέσματα παρατηρούμε ότι πρώτη φόρα οι εντολές τύπου update είναι πιο αργές από τις εντολές τύπου read. Αυτό συμβαδίζει με τη θεωρία καθώς οι εντολές ανάγνωσης είναι πιο γρήγορες από τις εντολές εγγραφής. Ο λόγος που συμβαίνει αυτό, δηλαδή να οι εντολές ανάγνωσης να έχουν πιο χαμηλή διεκπεραιωτική ικανότητα από τις εντολές εγγραφής στη Redis συμβαίνει επειδή δε χρησιμοποιείται η τεχνική pipelining στον πελάτη (ycsb client) που χρησιμοποιούμε [30]. Ο λόγος όμως που στη συγκεκριμένη περίπτωση η διεκπεραιωτική ικανότητα των εντολών ανάγνωσης είναι μεγαλύτερη από αυτή των εντολών

εγγραφής είναι επειδή στο σύμπλεγα αυτό έχουμε για κάθε κύριο κόμβο έναν υποτελή. Αυτό σημαίνει ότι ο κύριος κόμβος μπορεί να δρομολογήσει αιτήματα μόνο τύπου ανάγνωσης σε περιόδους όπου είναι πολύ αυξημένα τα αιτήματα ώστε να μεγαλώσει τη διεκπεραιωτική ικανότητα του συστήματος. Επίσης παρατηρούμε ότι στους φόρτους εργασία οι οποίοι έχουν εντολές εγγραφής η διεκπεραιωτική ικανότητα είναι μικρότερη σε σχέση με το σύμπλεγμα των 3 κόμβων. Το αποτέλεσμα αυτό απορρέει από το γεγονός ότι πλέον οι κύριοι κόμβοι πρέπει να ενημερώσουν και τους υποτελείς κάθε φορά που λαμβάνουν εντολές εγγραφής με αποτέλεσμα να δεσμεύονται περισσότεροι πόροι του συστήματος και να μειώνεται η διεκπεραιωτική ικανότητα του συμπλέγματος. Η συμπεριφορά της Aerospike δεν αλλάζει με τη προσθήκη των παραπάνω κόμβων, δηλαδή παρατηρούμε μια ελάχιστη μείωση της διεκπεραιωτικής ικανότητας και αύξησης της καθυστέρησης εκτέλεσης των εντολών σε όλους τους φόρτους εργασίας σε σχέση με το σύμπλεγμα των 3 κόμβων καθώς τώρα το συμπλεγμα έχει 6 κόμβους. Αυτό σημαίνει ότι καταναλώνονται περισσότεροι πόροι στη λειτουργία καρδιακού παλμού (heartbeat) όπως επίσης και για το θρυμματισμό των δεδομένων όπως αναφέραμε και στην προηγούμενη ανάλυση. Όσον αφορά την ArangoDB παρατηρούμε η διεκπεραιωτική ικανότητα και ο χρόνος καθυστέρησης των εντολών έχει παραμείνει σχεδόν ο ίδιος με μικρές παρεκκλίσεις. Αυτό το καταφέρνει καθώς σε αντίθεση με την Aerospike όπου κάθε κόμβος-διακομιστής αναλαμβάνει για όλες τις παραπάνω λειτουργίες, η ArangoDB όταν βρίσκεται σε λειτουργία συμπλέγματος ξεκινάει πέρα από τους κόμβους-διακομιστές και τις λειτουργίες των Μέσων (Agents) οι οποίοι είναι υπεύθυνοι για τον συντονισμό τους συμπλέγματος. Επίσης ξεκινάει και τους συντονιστές (Coordinators) οι οποίοι είναι υπεύθυνοι για την επικοινωνία με τους πελάτες (clients) και για την διεκπεραίωση των υπηρεσιών Foxx.

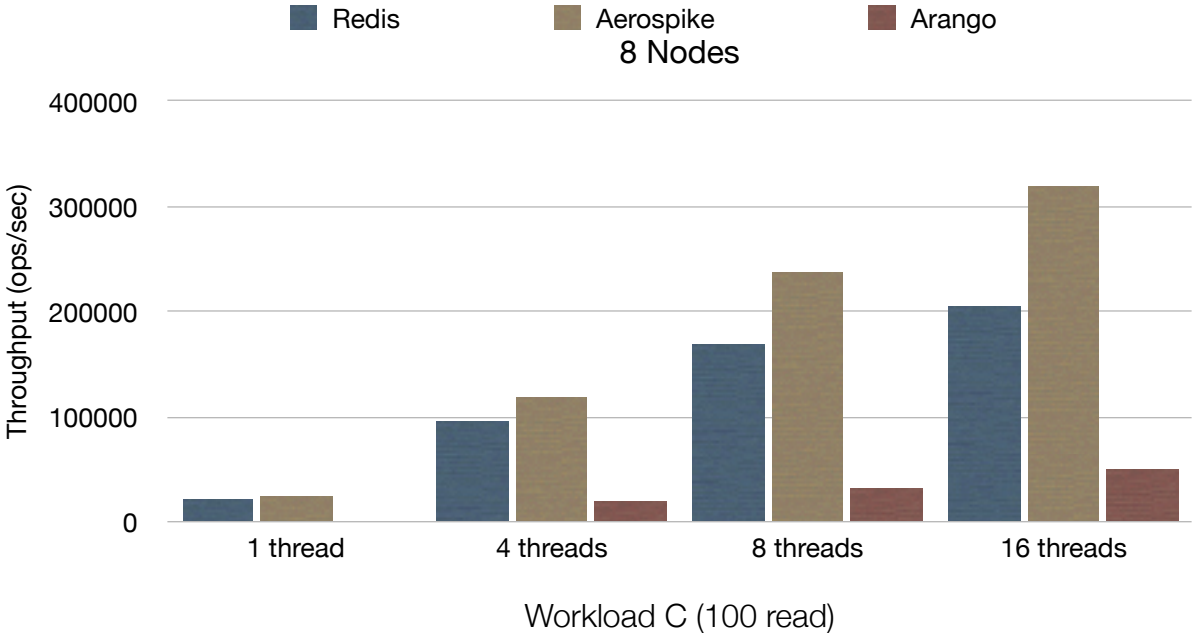
Συνεχίζοντας τα πειράματα μας η επόμενη διάταξη είναι αυτή του συμπλέγματος 8 κόμβων.Τα αποτελέσματα που συλλέγουμε για κάθε φόρτο εργασία όσο αφορά τη διεκπεραιωτική ικανότητα (throughput) είναι τα εξής:



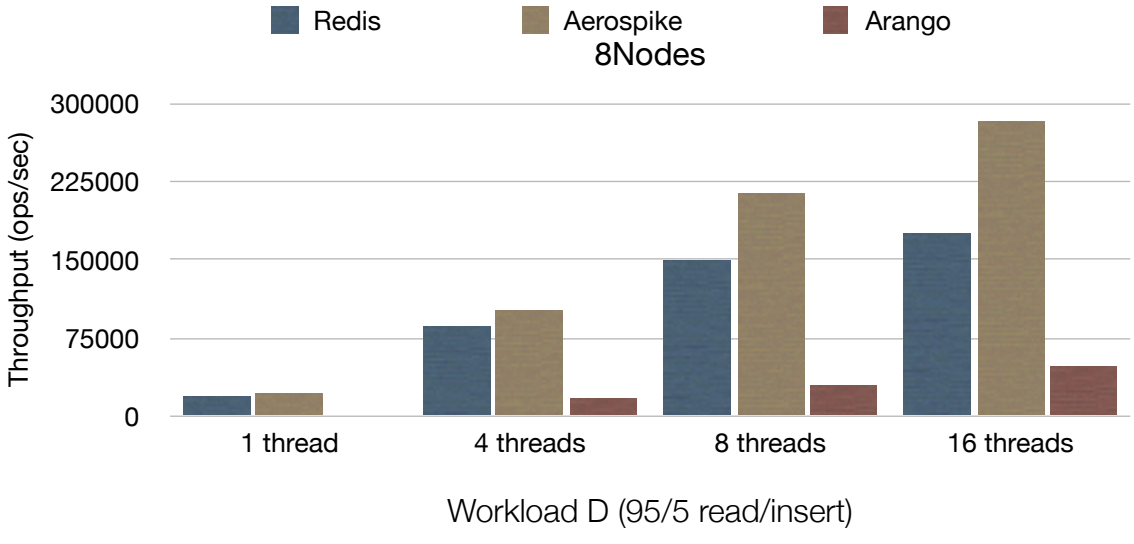
Διάγραμμα 31: Bare Metal Throughput 8 Nodes WorklaodA



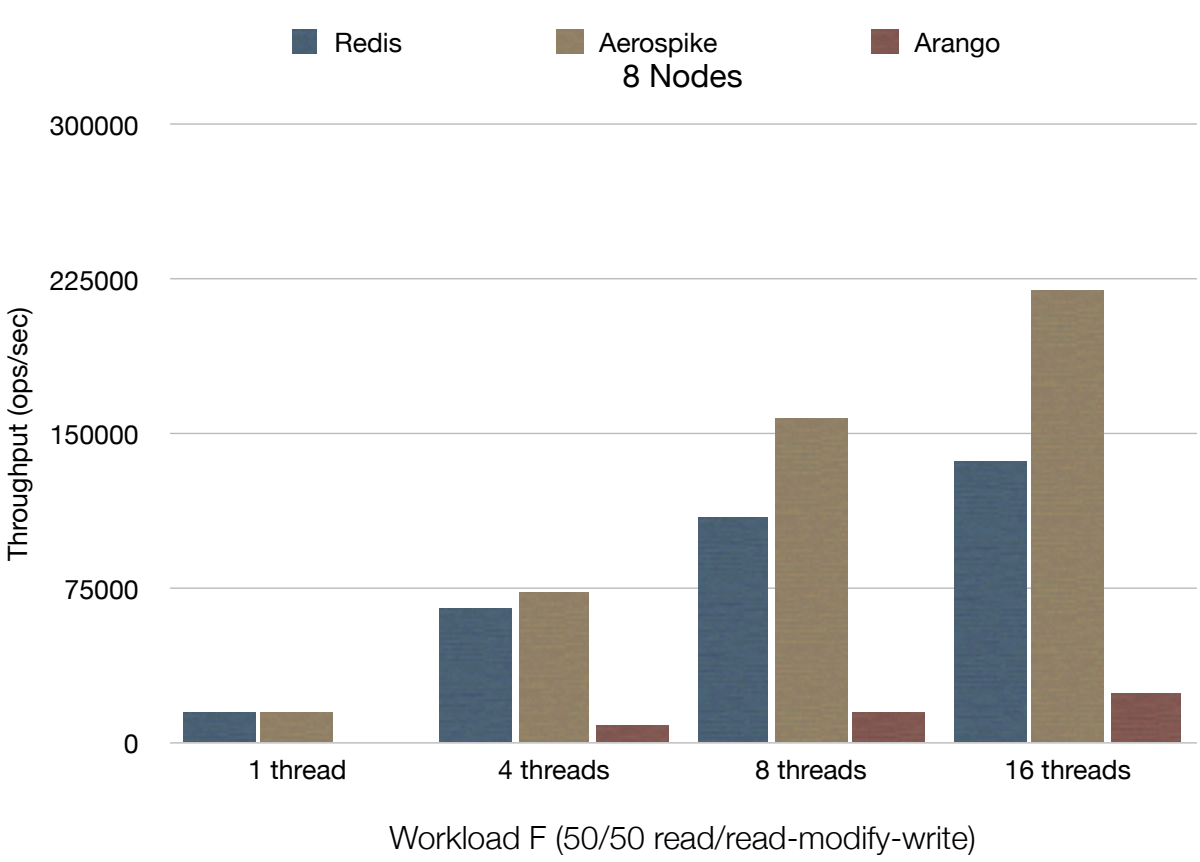
Διάγραμμα 32: Bare Metal Throughput 8 Nodes WorklaodB



Διάγραμμα 33: Bare Metal Throughput 8 Nodes WorklaodC

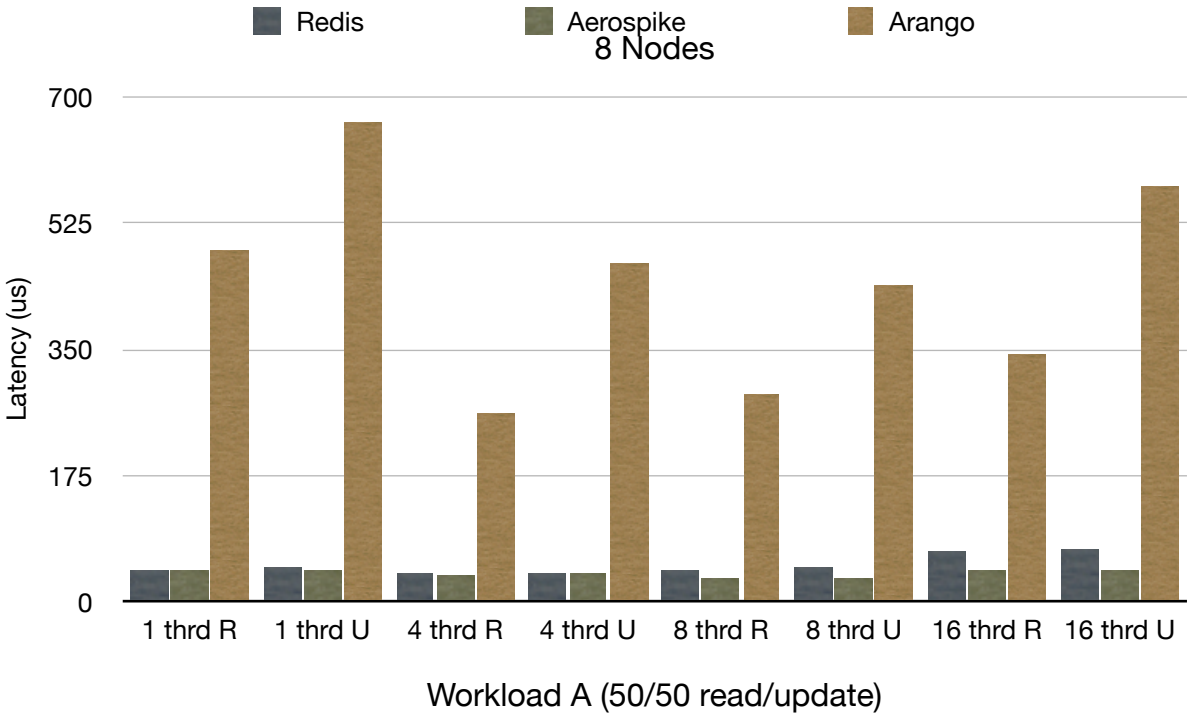


Διάγραμμα 34: Bare Metal Throughput 8 Nodes WorklaodD

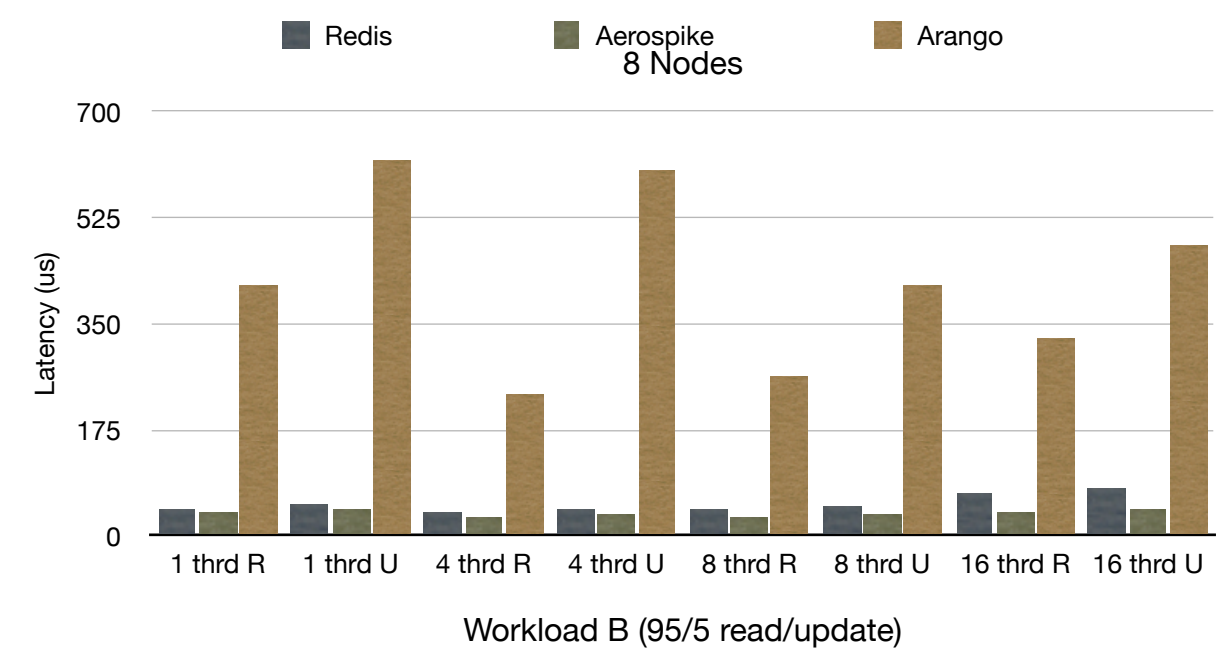


Διάγραμμα 35: Bare Metal Throughput 8 Nodes WorklaodF

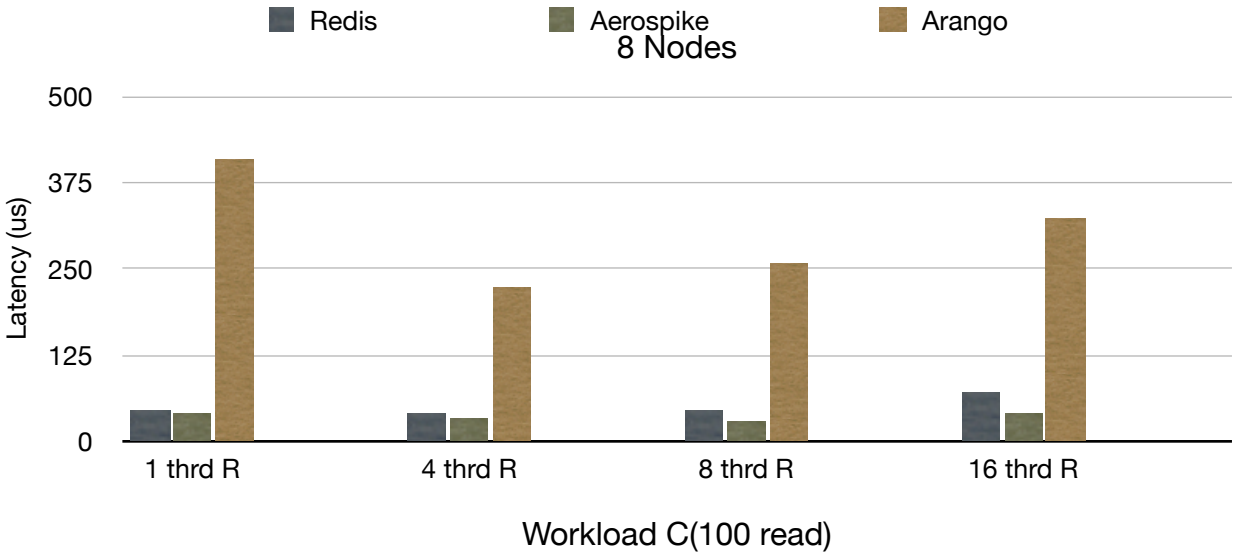
και για την χρόνο καθυστέρησης εκτέλεσης της κάθε εντολής (latency) έχουμε:



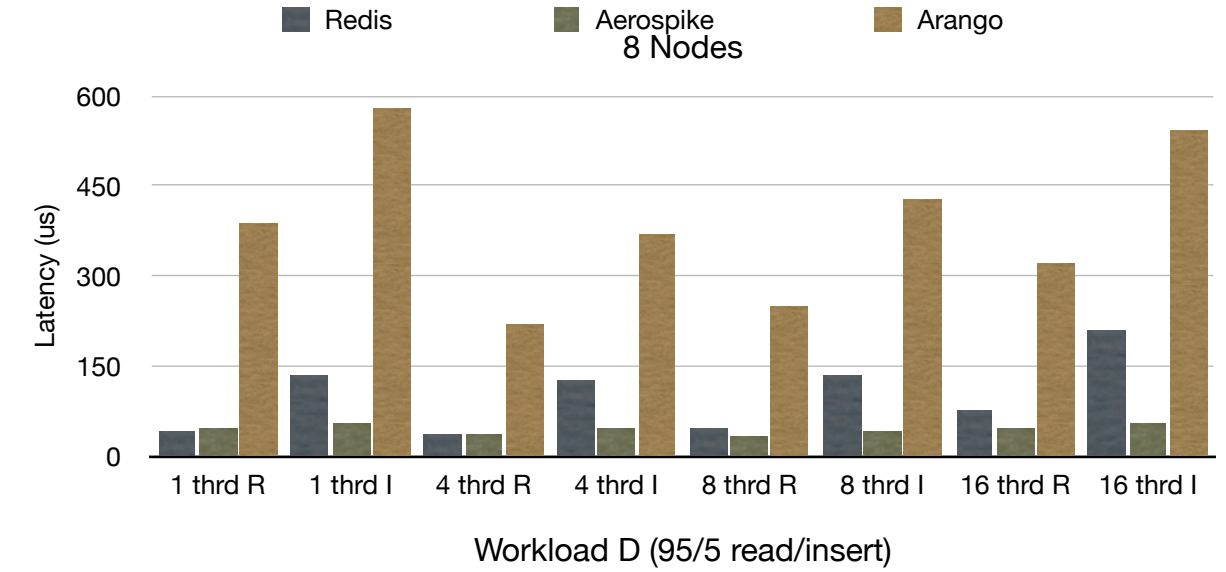
Διάγραμμα 36: Bare Metal Latency 8 Nodes WorklaodA



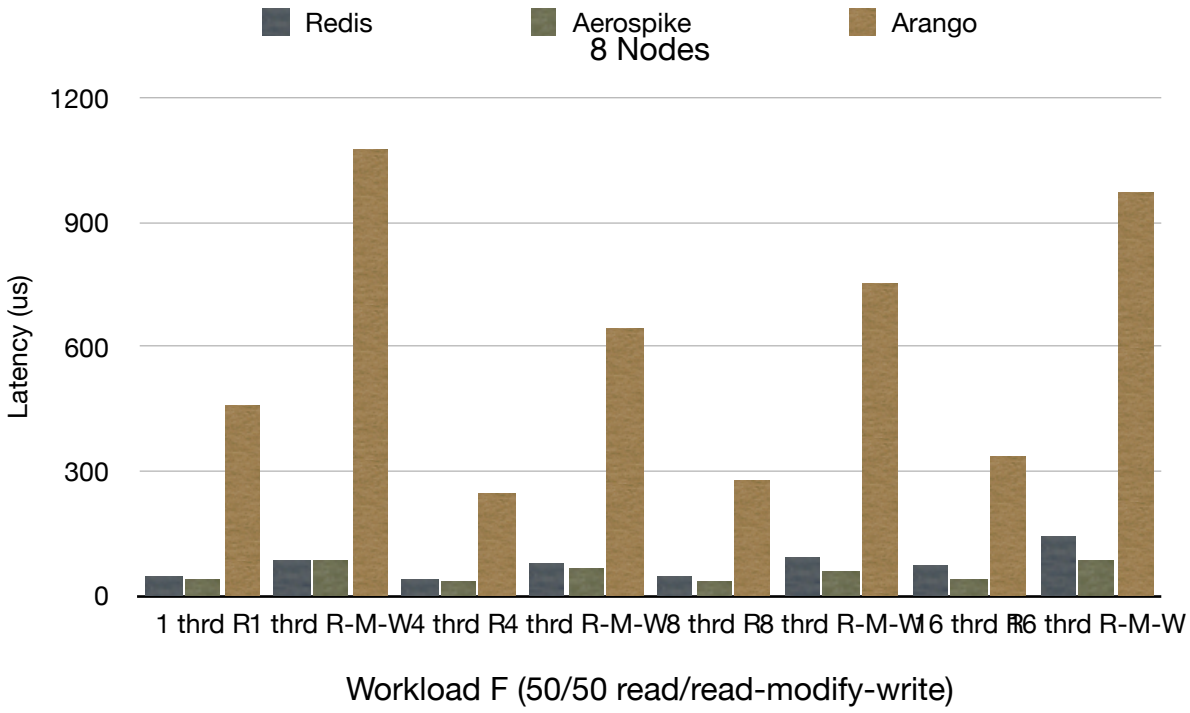
Διάγραμμα 37: Bare Metal Latency 8 Nodes WorklaodB



Διάγραμμα 38: Bare Metal Latency 8 Nodes WorklaodC



Διάγραμμα 39: Bare Metal Latency 8 Nodes WorklaodD



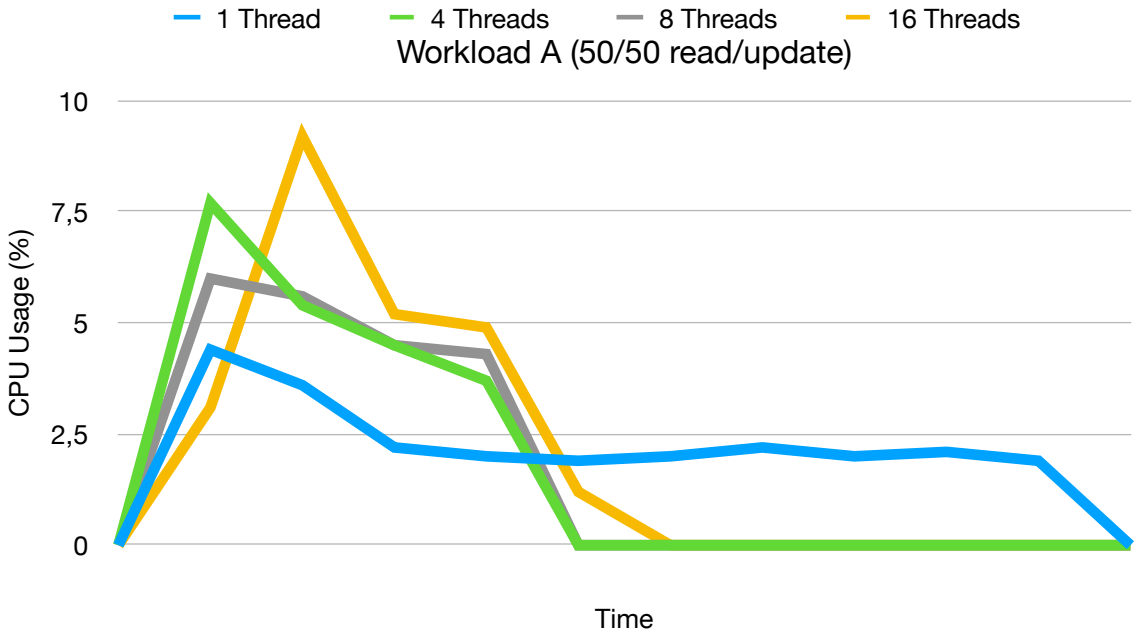
Διάγραμμα 40: Bare Metal Latency 8 Nodes WorklaodF

Από τα παραπάνω διαγράμματα παρατηρούμε πάλι ότι σε κάθε περίπτωση η Aerospike είναι η πιο γρήγορη βάση με την Redis να είναι η δεύτερη πιο γρήγορη και τέλος την ArangoDB. Πιο συγκεκριμένα παρατηρούμε ότι στο φόρτο εργασίας A (workload A) η Redis φτάνει εώς και 197,199 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 71 us και 73 us για τις εντολές ανάγνωσης (read) και τις εντολές ανανέωσης (update). Η Aerospike φτάνει εώς και 292,997 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 42 us και 43 us για τις εντολές ανάγνωσης (read) και τις εντολές ανανέωσης (update). Η ArangoDB φτάνει εώς και 34,285 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 342 us και 577 us για τις εντολές ανάγνωσης (read) και τις εντολές ανανέωσης (update). Για το φόρτο εργασίας B (workload B) η Redis φτάνει εώς και 201,369 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 70 us και 78 us για τις εντολές ανάγνωσης (read) και τις εντολές ανανέωσης (update). Η Aerospike φτάνει εώς και 310,269 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 40 us και 43 us για τις εντολές ανάγνωσης (read) και τις εντολές ανανέωσης (update). Η ArangoDB φτάνει εώς και 47,420 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 324 us και 479 us για τις εντολές ανάγνωσης (read) και τις εντολές ανανέωσης (update). Για το φόρτο εργασίας C (workload C) η Redis φτάνει εώς και 205,380 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 69 us για τις εντολές ανάγνωσης (read). Η Aerospike φτάνει εώς και 319,081 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 39 us για τις εντολές ανάγνωσης (read). Η ArangoDB φτάνει εώς και 48,718 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 323 us για τις εντολές ανάγνωσης (read). Για το φόρτο εργασίας D (workload D) η Redis φτάνει εώς και 175,685 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 75 us και 208 us για τις εντολές ανάγνωσης (read) και τις εντολές εισαγωγής (insert). Η Aerospike φτάνει εώς και 282,167 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 44 us και 53 us για τις εντολές ανάγνωσης (read) και τις εντολές εισαγωγής (insert). Η ArangoDB φτάνει εώς και 47,054 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 323 us και 543 us για τις εντολές ανάγνωσης (read) και τις εντολές εισαγωγής (insert). Για

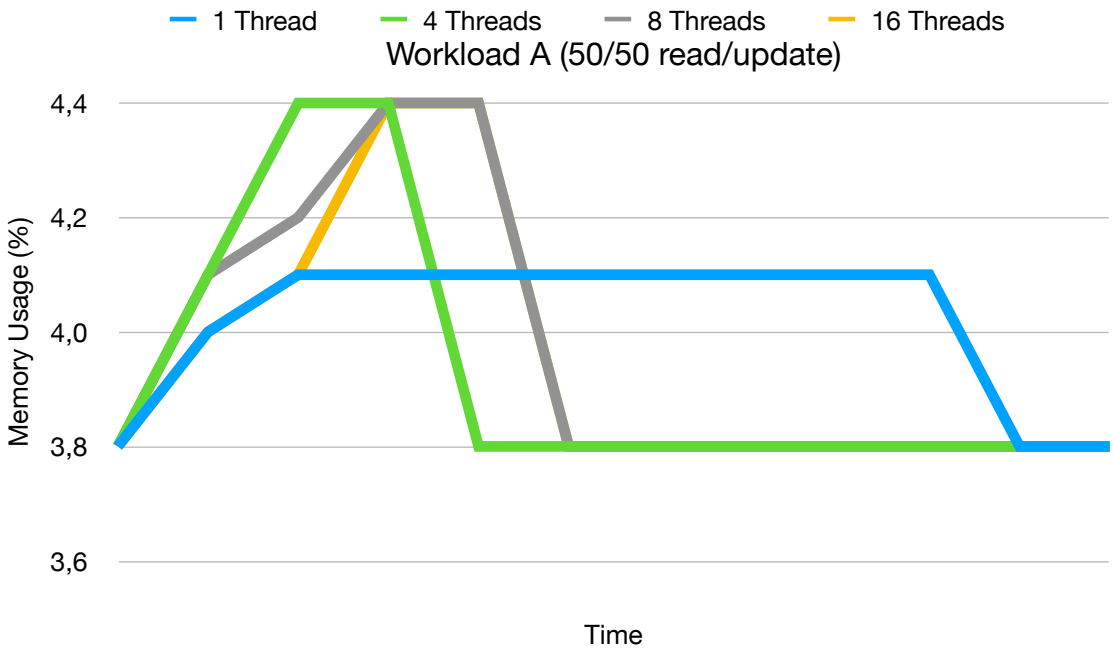
το φόρτο εργασίας F (workload F) η Redis φτάνει εώς και 137,042 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 69 us και 143 us για τις εντολές ανάγνωσης (read) και τις εντολές ανάγνωσης-τροποποίησης-εγγραφής(read-modify-write). Η Aerospike φτάνει εώς και 219,298 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 39 us και 84 us για τις εντολές ανάγνωσης (read) και τις εντολές ανάγνωσης-τροποποίησης-εγγραφής(read-modify-write). Η ArangoDB φτάνει εώς και 24,303 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 332 us και 971 us για τις εντολές ανάγνωσης (read) και τις εντολές ανάγνωσης-τροποποίησης-εγγραφής(read-modify-write). Από τα παραπάνω αποτελέσματα παρατηρούμε ότι η Redis έχει αυξημένη διεκπεραιωτική ικανότητα και μειωμένο χρόνο καθυστέρησης εκτέλεσης εντολών σε σχέση με το σύμπλεγμα των 6 κόμβων. Το αποτέλεσμα αυτό συμβαδίζει με αυτό που περιμέναμε καθώς κάθε κόμβος τώρα είναι υπεύθυνος για λιγότερες εγγραφές με αποτέλεσμα να μπορεί να εξυπηρετήσει περισσότερα παράλληλα αιτήματα από τον πελάτη (ycsb client). Η Aerospike εμφανίζει παρόμοια συμπεριφορά, δηλαδή παρουσιάζει ελάχιστα μικρότερη επίδοση καθώς ο αριθμό των κόμβων έχει πλέον αυξηθεί. Η ArangoDB παρουσιάζει και αυτή παρόμοια συμπεριφορά με το σύμπλεγμα των 6 κόμβων καθώς αν και οι κόμβοι αυξήθηκαν η απόδοση του συμπλέγματος έχει παραμείνει σταθερή.

5.2.2 Πειραματικά Αποτελέσματα σε Πραγματικό Περιβάλλον - CPU & RAM Usage

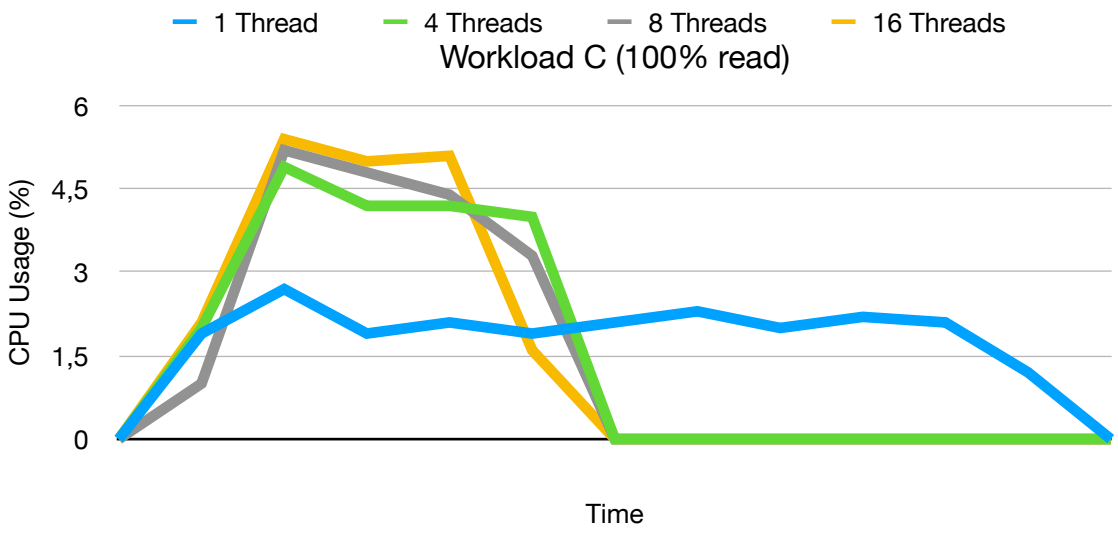
Συνεχίζοντας την ανάλυση μας αποφασίσαμε να συλλέξουμε την χρησιμοποίηση μνήμης και επεξεργαστικής ισχύος κάτω από τα διάφορους φόρτους εργασίας. Αποφασίσαμε ότι η διάταξη μονού κόμβου και το σύμπλεγμα 6 κόμβων είναι αρκετά αντιπροσωπευτικές στους φόρτους εργασίας A,C και D. Τα αποτελέσματα που συλλέξαμε για την Redis στη διάταξη μονού κόμβου είναι:



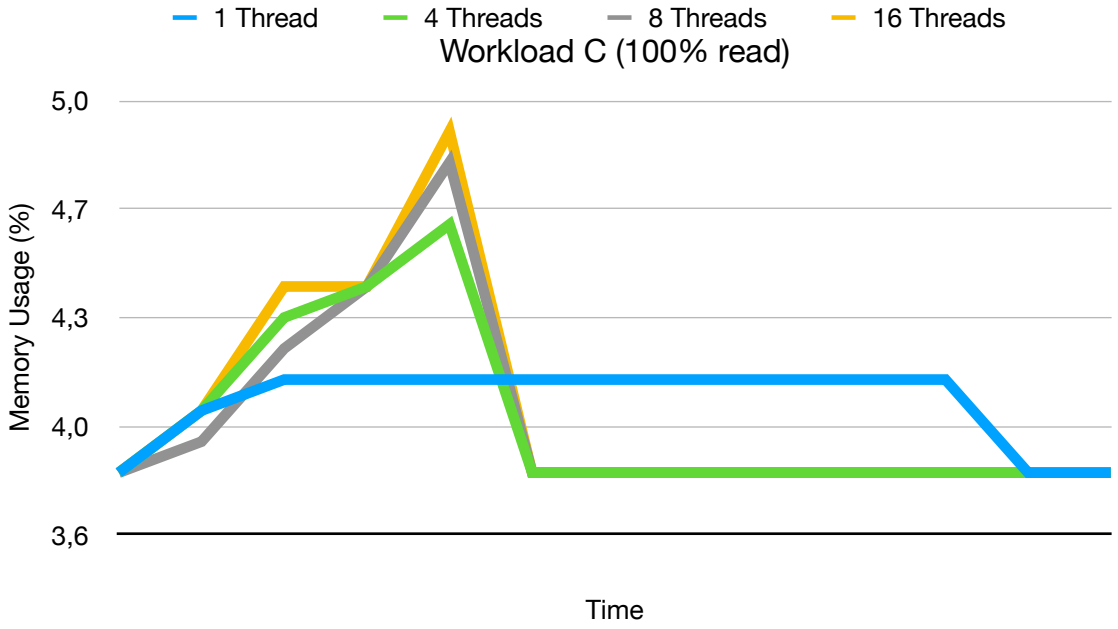
Διάγραμμα 41: Redis CPU Usage WorkloadA 1 Node



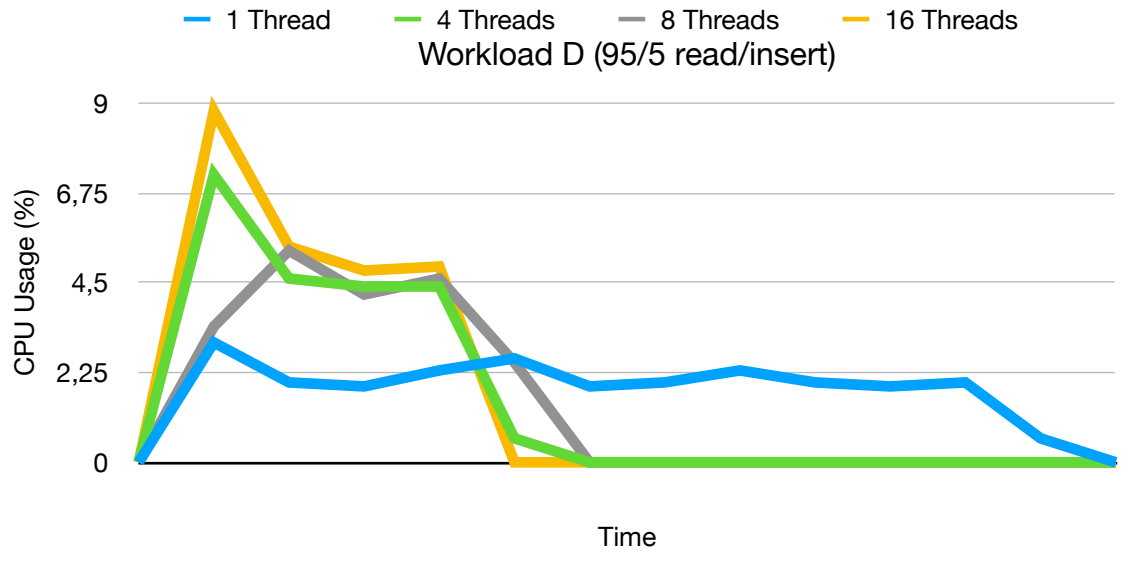
Διάγραμμα 42: Redis Memory Usage WorkloadA 1 Node



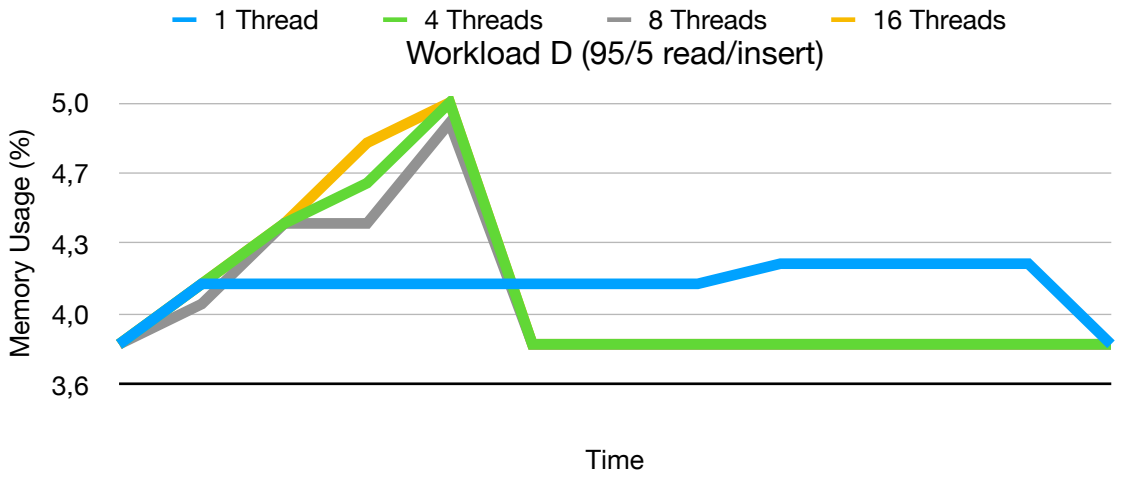
Διάγραμμα 43: Redis CPU Usage WorkloadC 1 Node



Διάγραμμα 44: Redis Memory Usage WorkloadC 1 Node

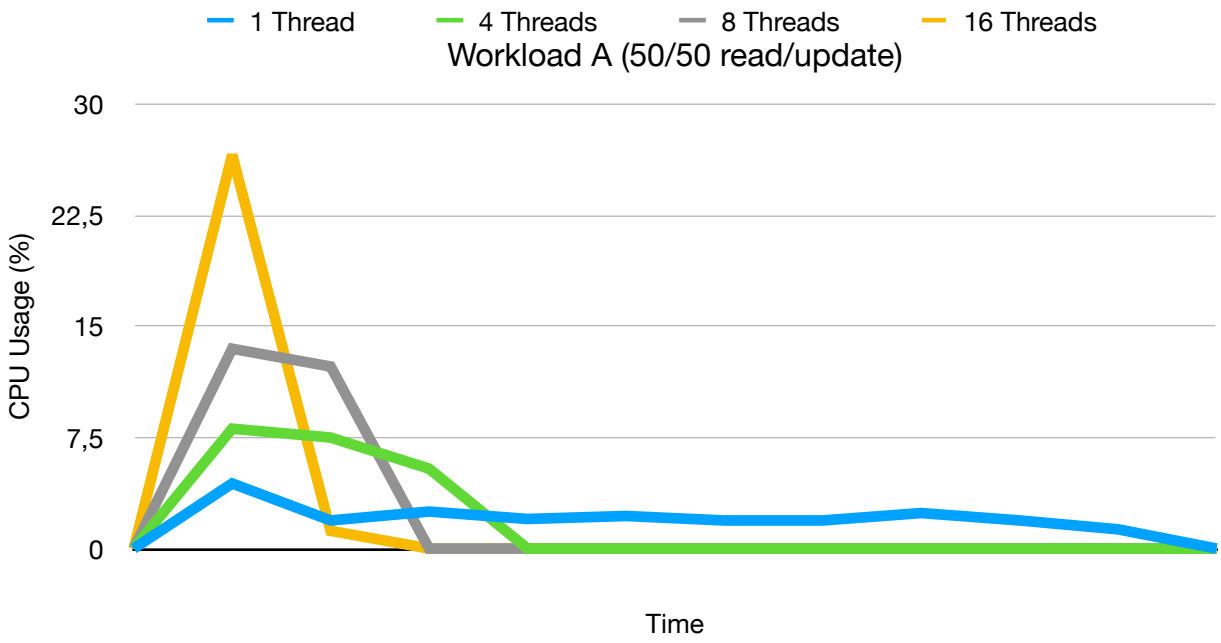


Διάγραμμα 45: Redis CPU Usage WorkloadD 1 Node

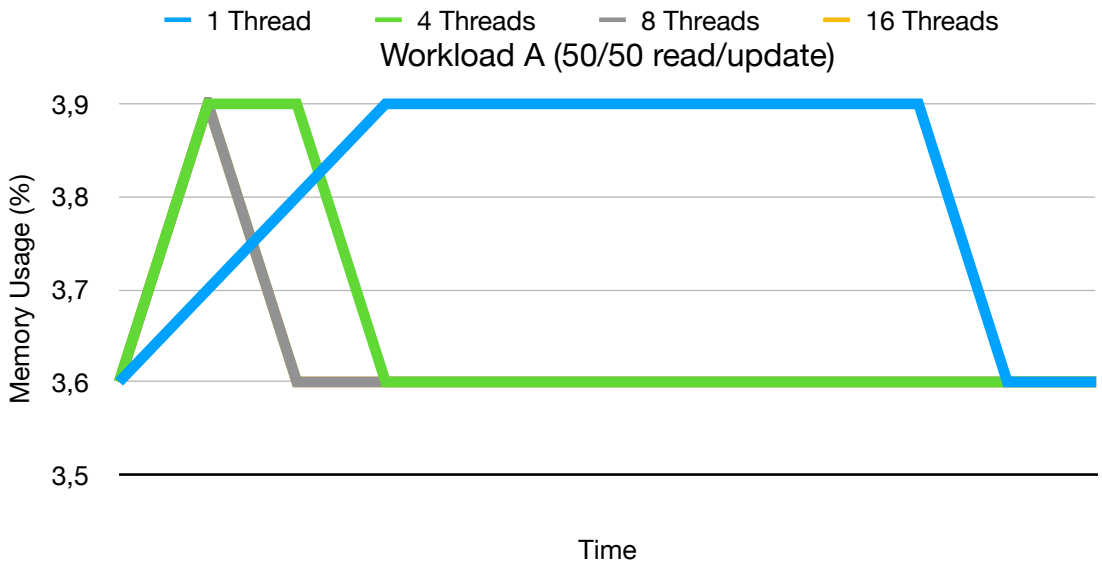


Διάγραμμα 46: Redis Memory Usage WorkloadD 1 Node

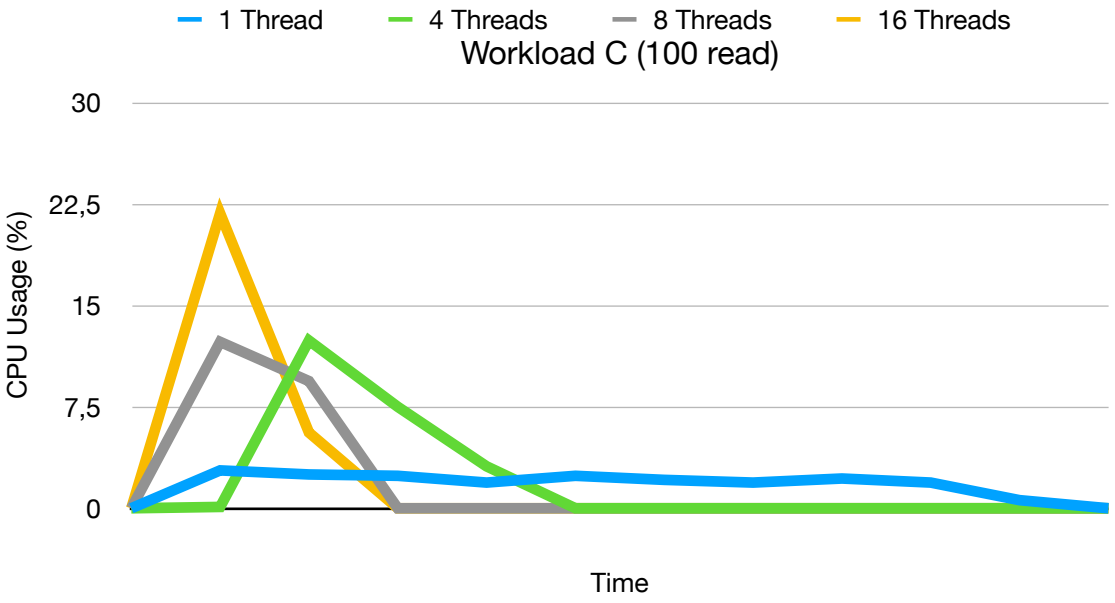
Για την Aerospike για τη διάταξη μονού κόμβου έχουμε:



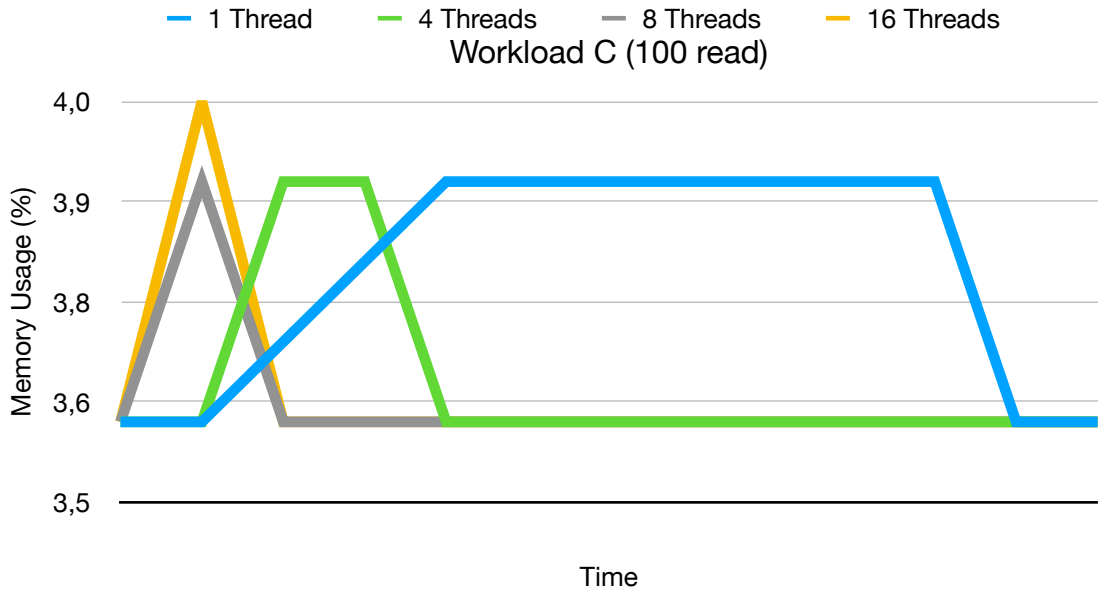
Διάγραμμα 47: Aerospike CPU Usage WorkloadA 1 Node



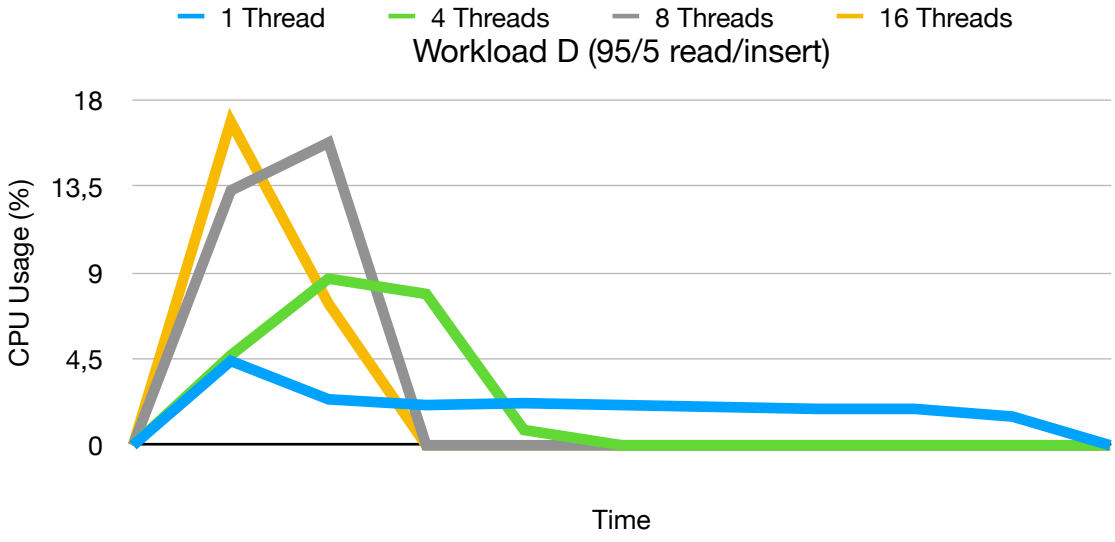
Διάγραμμα 48: Aerospike Memory Usage WorkloadA 1 Node



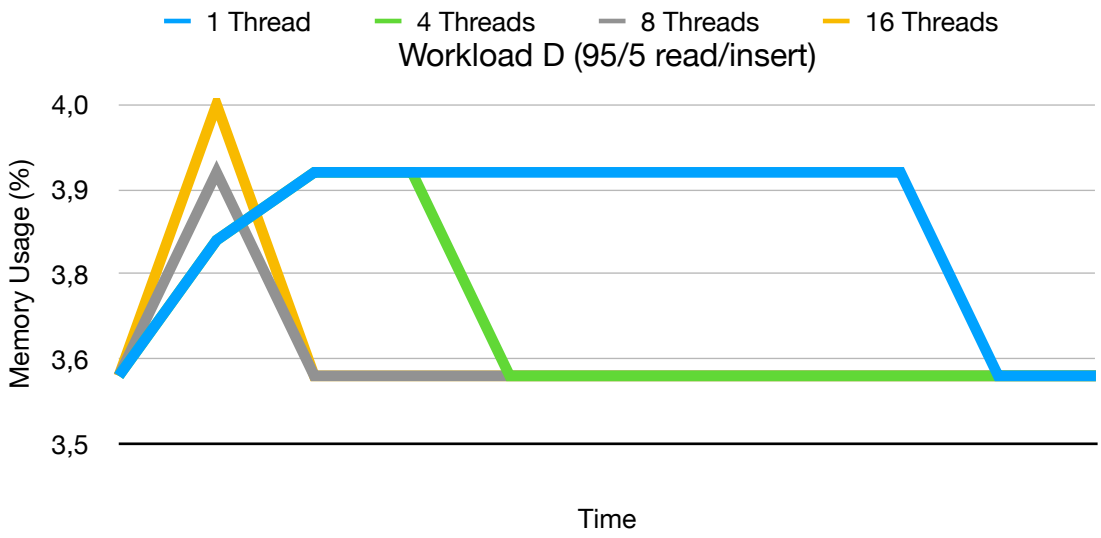
Διάγραμμα 49: Aerospike CPU Usage WorkloadC 1 Node



Διάγραμμα 50: Aerospike Memory Usage WorkloadC 1 Node

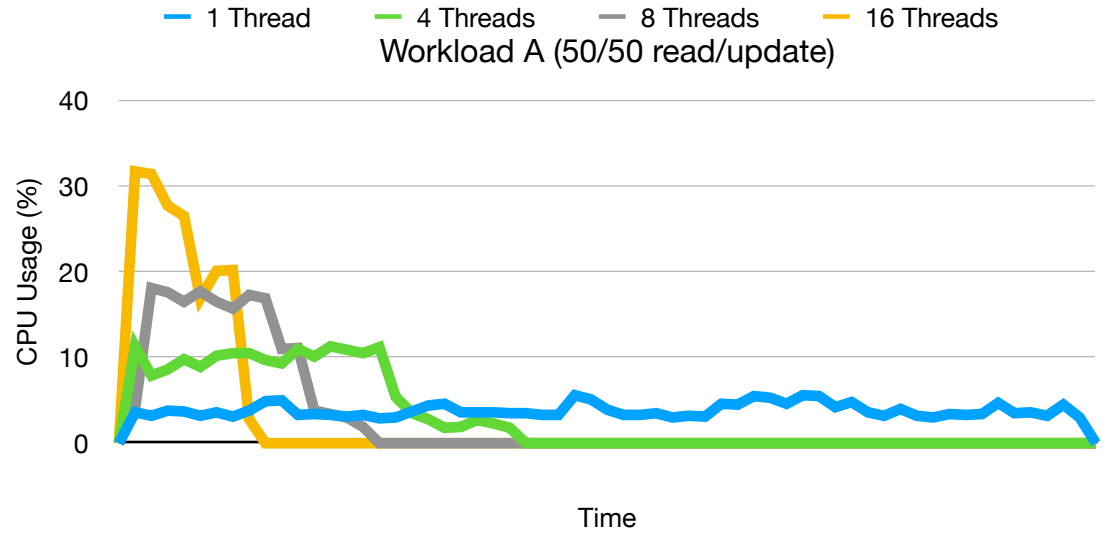


Διάγραμμα 51: Aerospike CPU Usage WorkloadD 1 Node

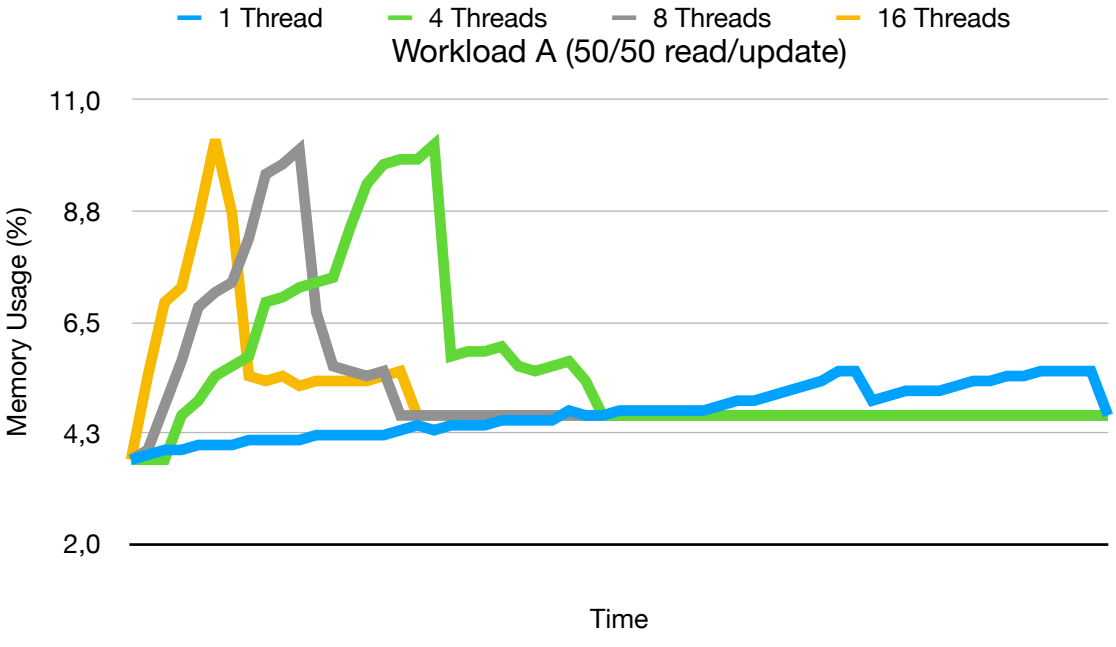


Διάγραμμα 52: Aerospike Memory Usage WorkloadD 1 Node

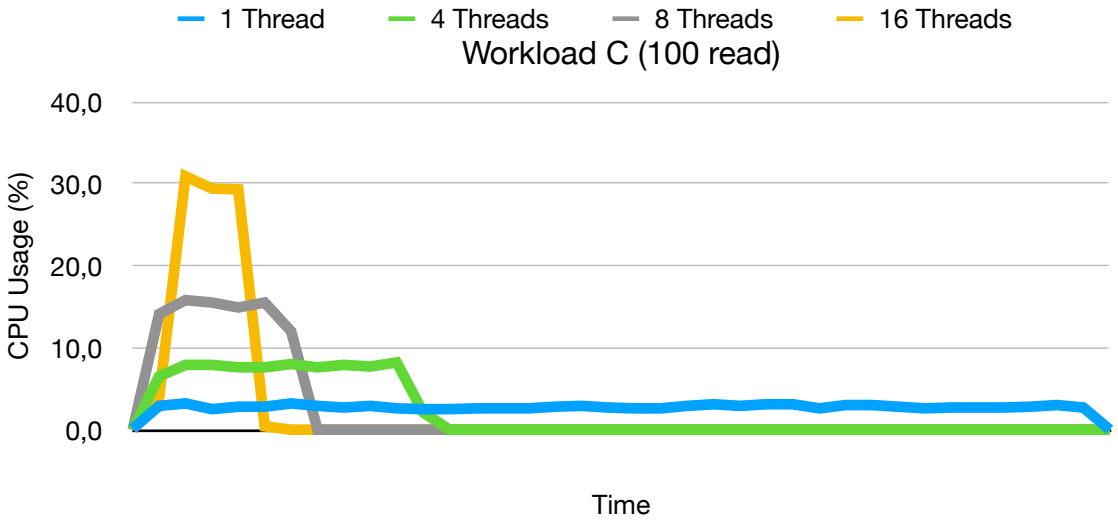
και για την ArangoDB για τη διάταξη μονού κόμβου έχουμε:



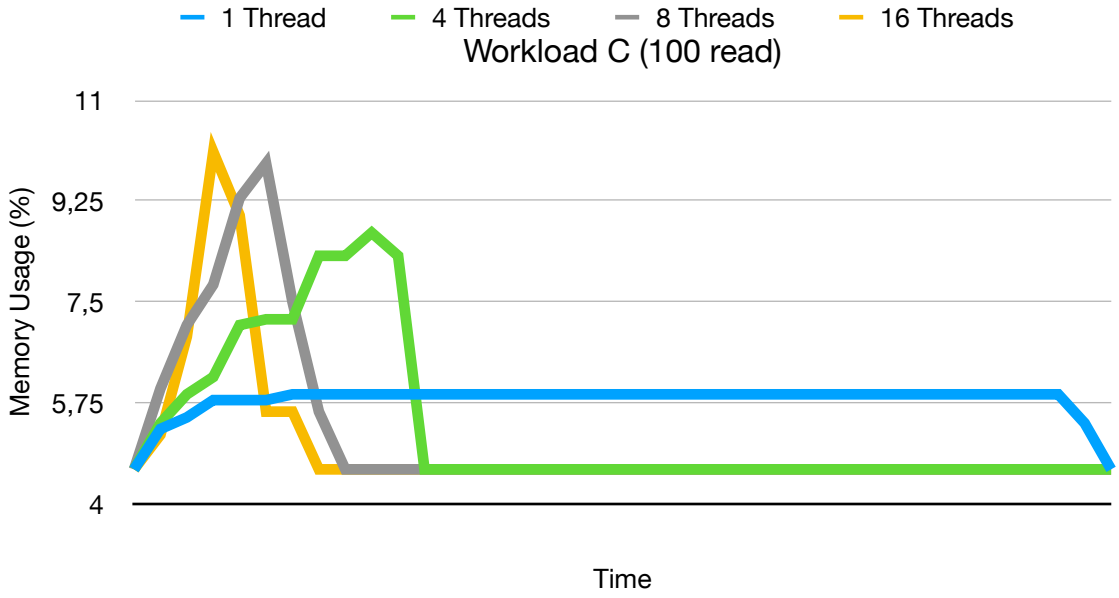
Διάγραμμα 53: ArangoDB CPU Usage WorkloadA 1 Node



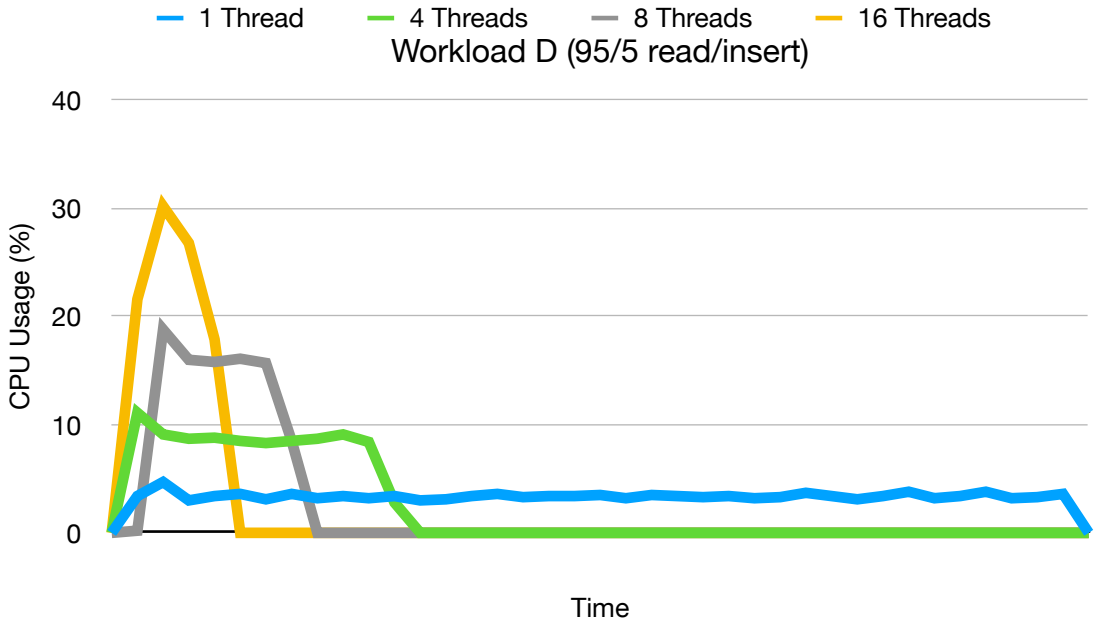
Διάγραμμα 54: ArangoDB Memory Usage WorkloadA 1 Node



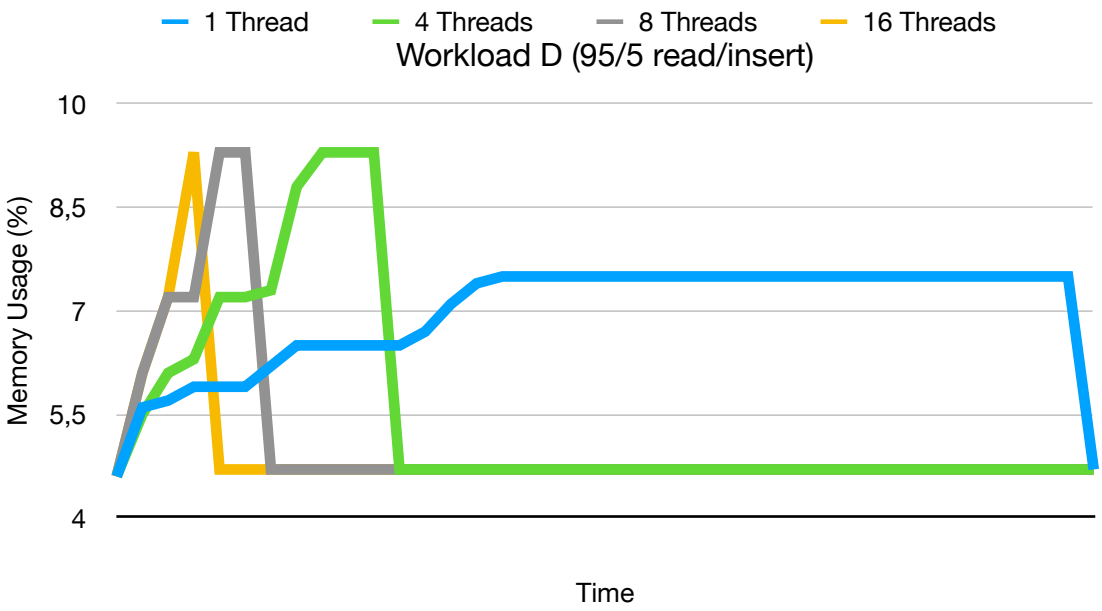
Διάγραμμα 55: ArangoDB CPU Usage WorkloadC 1 Node



Διάγραμμα 56: ArangoDB Memory Usage WorkloadC 1 Node



Διάγραμμα 57: ArangoDB CPU Usage WorkloadD 1 Node

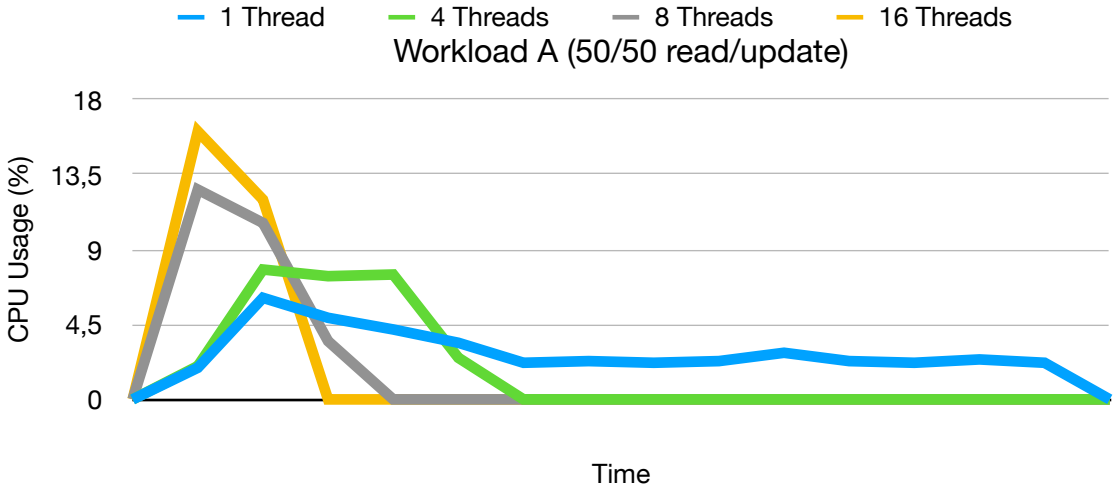


Διάγραμμα 58: ArangoDB Memory Usage WorkloadD 1 Node

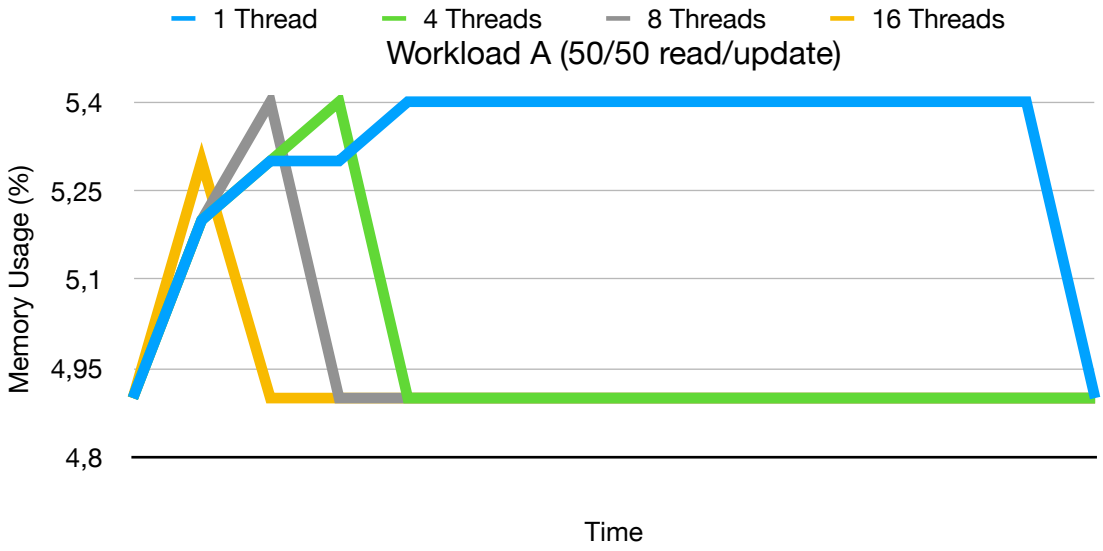
Παρατηρούμε ότι στο φόρτο εργασίας A (workload A) η Redis όταν λειτουργούν παράλληλα 16 νήματα φτάνει εώς και 9,2% χρησιμοποίηση της ισχύος του επεξεργαστή και ξεκινώντας από το 3,8% φτάνει και μέχρι 4,4% χρησιμοποίηση της μνήμης. Για την Aerospike παρατηρούμε ότι όταν λειτουργούν παράλληλα 16 νήματα φτάνει εώς και 26,6% χρησιμοποίηση της ισχύος του επεξεργαστή και ξεκινώντας από το 3,6% φτάνει και μέχρι 4,9% χρησιμοποίηση της μνήμης. Για την ArangoDB παρατηρούμε ότι όταν λειτουργούν παράλληλα 16 νήματα φτάνει εώς και 31,7% χρησιμοποίηση της ισχύος του επεξεργαστή και ξεκινώντας από το 3,7% φτάνει και μέχρι 4,6% χρησιμοποίηση της μνήμης. Για το φόρτο εργασίας C (workload C) η Redis όταν λειτουργούν παράλληλα 16 νήματα φτάνει εώς και 5,4% χρησιμοποίηση της ισχύος του επεξεργαστή και ξεκινώντας από το 3,8% φτάνει και μέχρι 4,9% χρησιμοποίηση της μνήμης. Για την Aerospike παρατηρούμε ότι όταν λειτουργούν παράλληλα 16 νήματα φτάνει εώς και 21,8% χρησιμοποίηση της ισχύος του επεξεργαστή και ξεκινώντας από το 3,6% φτάνει και μέχρι 4,1% χρησιμοποίηση της μνήμης. Για την ArangoDB παρατηρούμε ότι όταν λειτουργούν παράλληλα 16 νήματα φτάνει εώς και 30,9% χρησιμοποίηση της ισχύος του επεξεργαστή και ξεκινώντας από το 4,6% φτάνει και μέχρι 9% χρησιμοποίηση της μνήμης. Για το φόρτο εργασίας D (workload D) η Redis όταν λειτουργούν παράλληλα 16 νήματα φτάνει εώς και 8,4% χρησιμοποίηση της ισχύος του επεξεργαστή και ξεκινώντας από το 3,8% φτάνει και μέχρι 5% χρησιμοποίηση της μνήμης. Για την Aerospike παρατηρούμε ότι όταν λειτουργούν παράλληλα 16 νήματα φτάνει εώς και 21,8% χρησιμοποίηση της ισχύος του επεξεργαστή και ξεκινώντας από το 3,6% φτάνει και μέχρι 4% χρησιμοποίηση της μνήμης. Για την ArangoDB παρατηρούμε ότι όταν λειτουργούν παράλληλα 16 νήματα φτάνει εώς και 30,2% χρησιμοποίηση της ισχύος του επεξεργαστή και ξεκινώντας από το 4,6% φτάνει και μέχρι 9,3% χρησιμοποίηση της μνήμης. Από τα παραπάνω παρατηρούμε ότι όλες οι βάσεις αυξάνουν γραμμικά τη χρησιμοποίηση της ισχύος του επεξεργαστή όσο αυξάνονται τα παράλληλα νήματα εκτέλεσης. Η Aerospike κατά τη διάρκεια εκτέλεσης του φόρτου εργασίας χρησιμοποιεί τη λιγότερη μνήμη, με την Redis στη δεύτερη θέση και τέλος η ArangoDB που χρησιμοποιεί την περισσότερη μνήμη από όλες. Ακόμα παρατηρούμε ότι η ArangoDB και η Aerospike χρησιμοποιούν σχεδόν διπλάσια υπολογιστική ισχύ σε σχέση με Redis. Έπειτα

παρατηρούμε ότι στο τέλος της εκτέλεσης του κάθε φόρτου εργασίας η μνήμη και η χρήση του επεξεργαστή επανέρχονται στην αρχική τους κατάσταση πριν από την εκτέλεση του φόρτου εργασίας. Υπάρχει όμως μια εξαίρεση. Για την ArangoDB παρατηρούμε ότι στον φόρτο εργασίας A η μνήμη δεν επανέρχεται στην αρχική της κατάσταση. Αυτό συμβαίνει επειδή οι αλλαγές που συμβαίνουν σε αυτό τον φόρτο εργασίας αποθηκεύονται προσωρινά στη μνήμη. Το μέγεθος όμως είναι μεγαλύτερο από τα πραγματικά κλειδιά που ανανεώνονται. Αυτό συμβαίνει επειδή η RocksDB, που είναι η μηχανή βάσης που χρησιμοποιεί η ArangoDB, για κάθε 1MB μνήμης που δεσμεύει, δεσμεύει και άλλο τόσο με dummy δεδομένα προκειμένου να μπορεί να παρακολουθεί το μέγεθος της προσωρινής μνήμης cache ακριβέστερα και να μπορεί να αποδεσμεύσει μπλοκς μνήμης όταν χρειαστει [31]. Επίσης αυτή τη συμπεριφορά την παρατηρούμε και στο φόρτο εργασίας D όπου εισάγονται νέες εγγραφές.

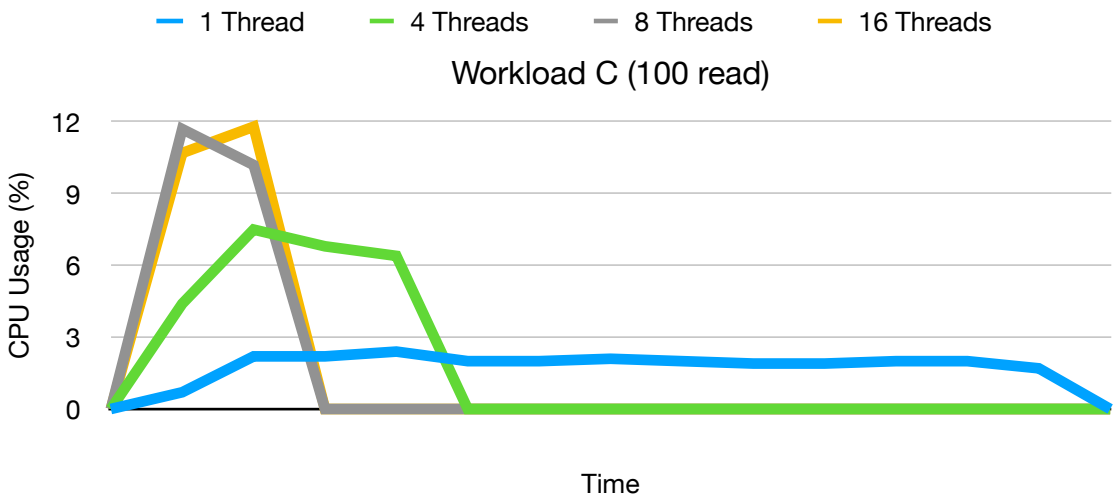
Συνεχίζοντας με το σύμπλεγμα των 6 κόμβων, τα αποτελέσματα που συλλέξαμε για την Redis είναι:



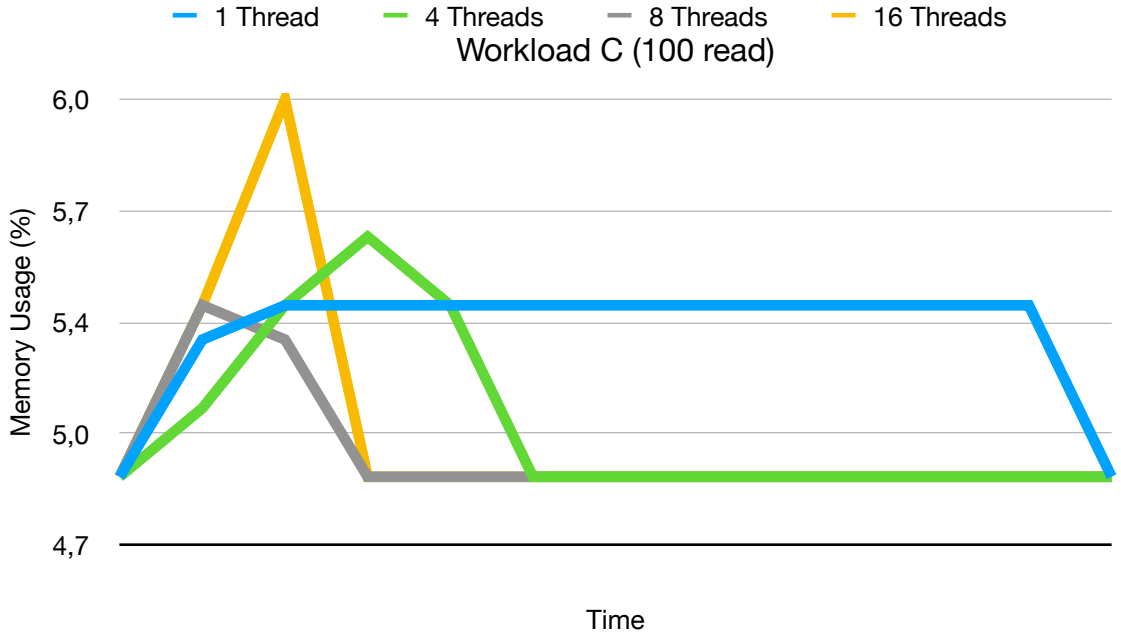
Διάγραμμα 59: Redis CPU Usage WorkloadA 6 Nodes



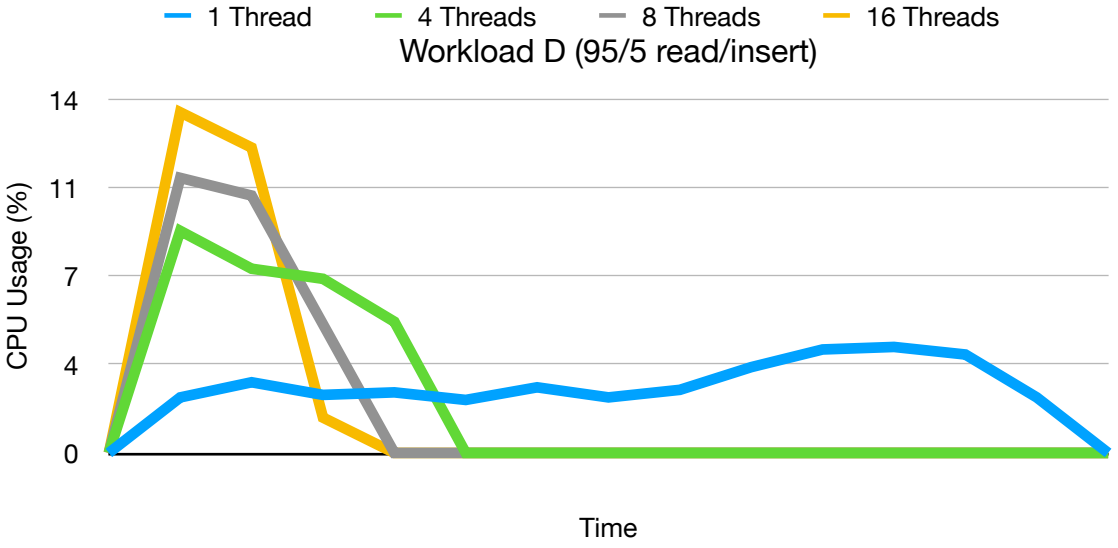
Διάγραμμα 60: Redis Memory Usage WorkloadA 6 Nodes



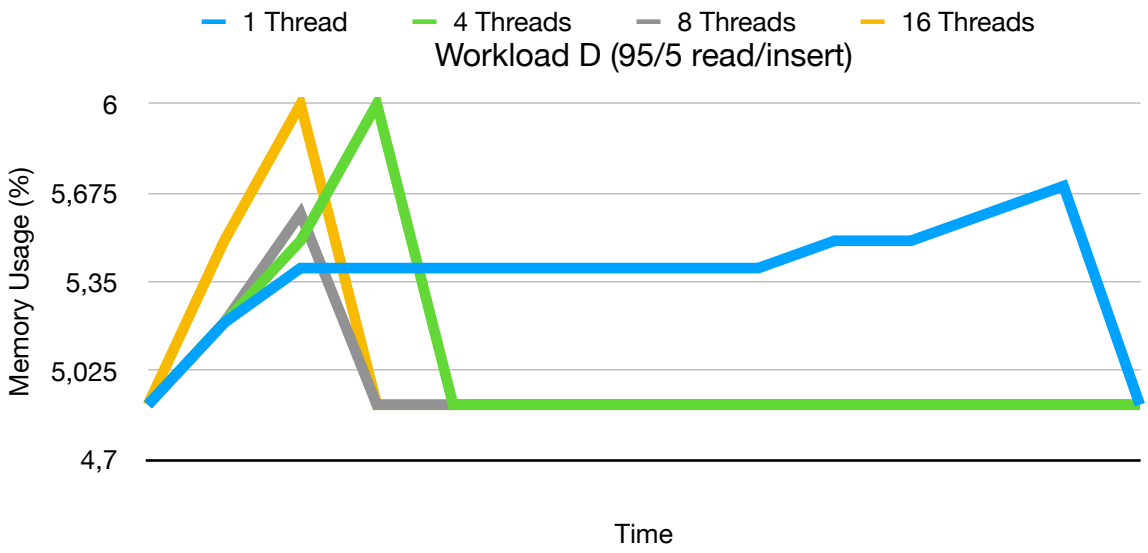
Διάγραμμα 61: Redis CPU Usage WorkloadC 6 Nodes



Διάγραμμα 62: Redis Memory Usage WorkloadC 6 Nodes

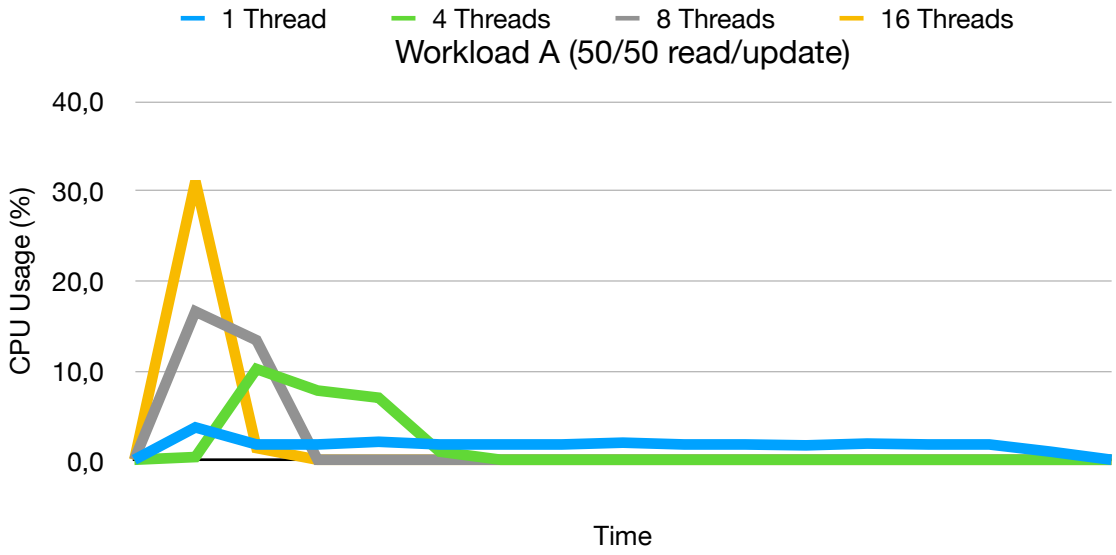


Διάγραμμα 63: Redis CPU Usage WorkloadD 6 Nodes

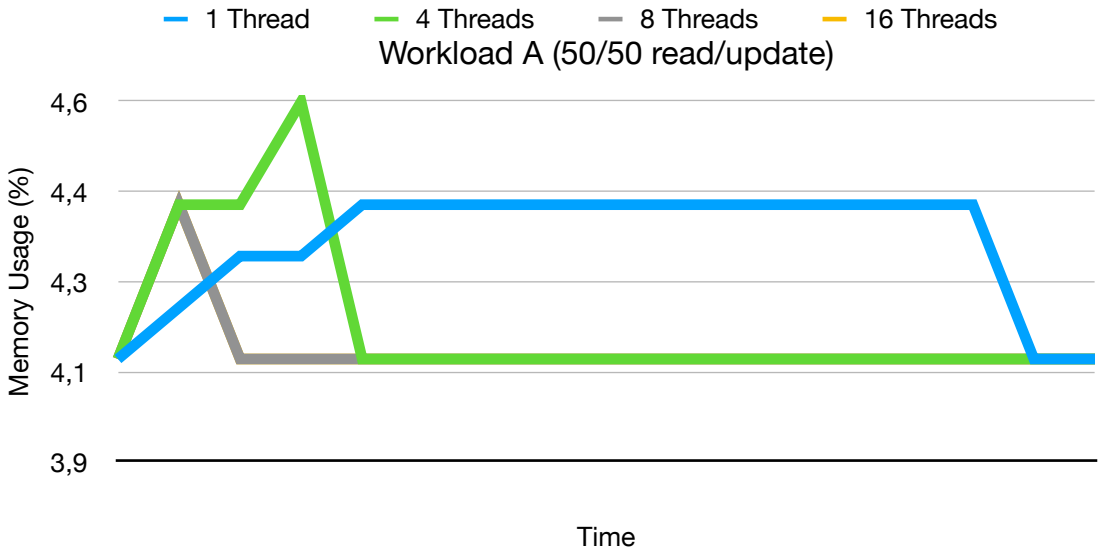


Διάγραμμα 64: Redis Memory Usage WorkloadD 6 Nodes

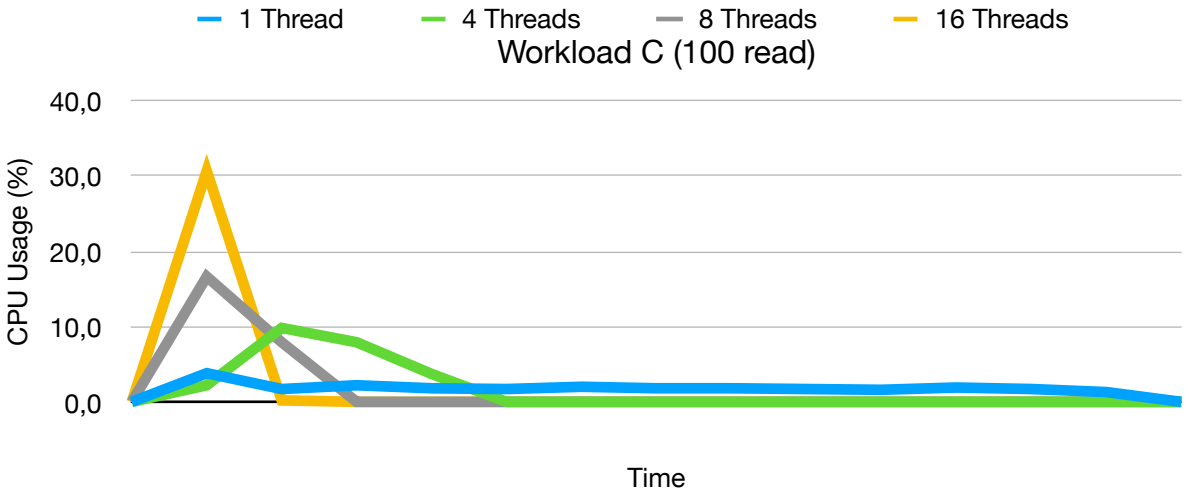
Για την Aerospike έχουμε:



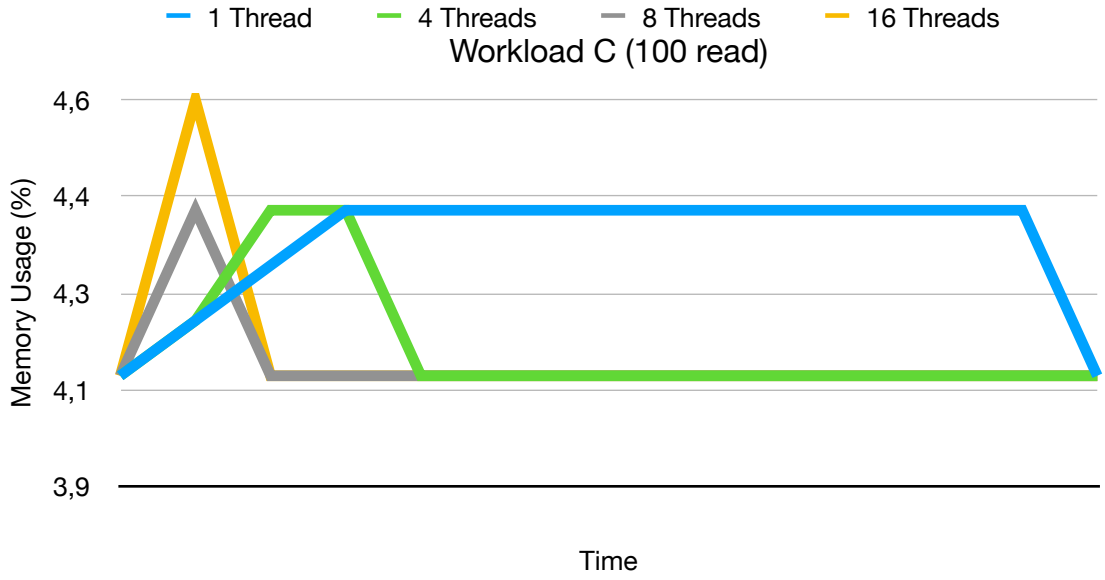
Διάγραμμα 65: Aerospike CPU Usage WorkloadA 6 Nodes



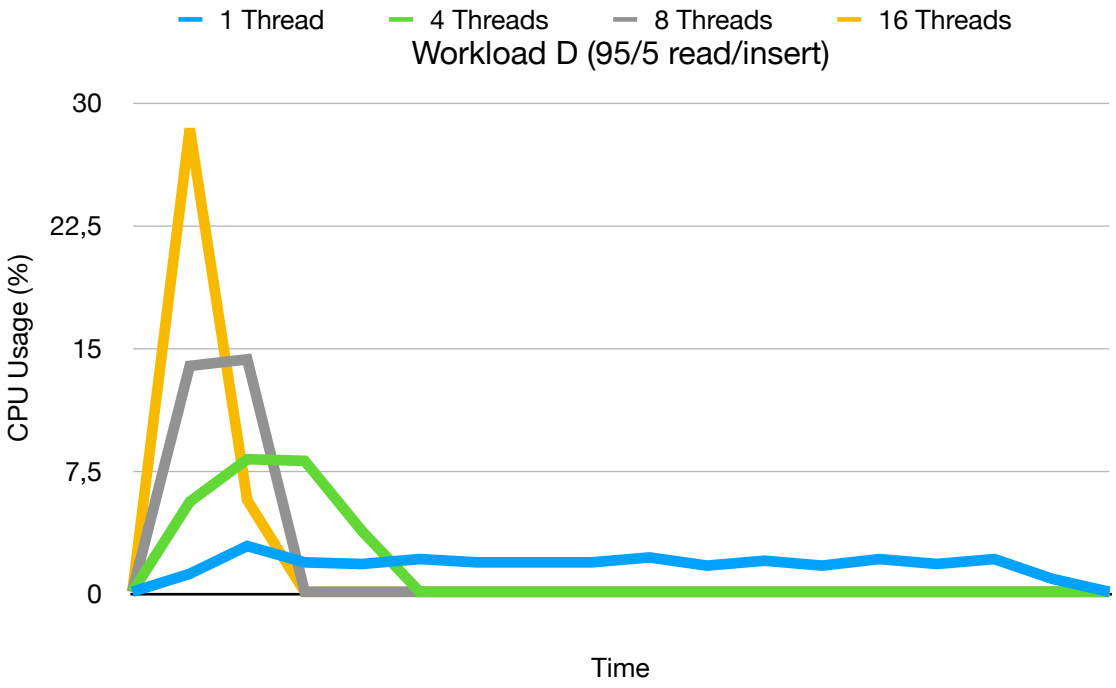
Διάγραμμα 66: Aerospike Memory Usage WorkloadA 6 Nodes



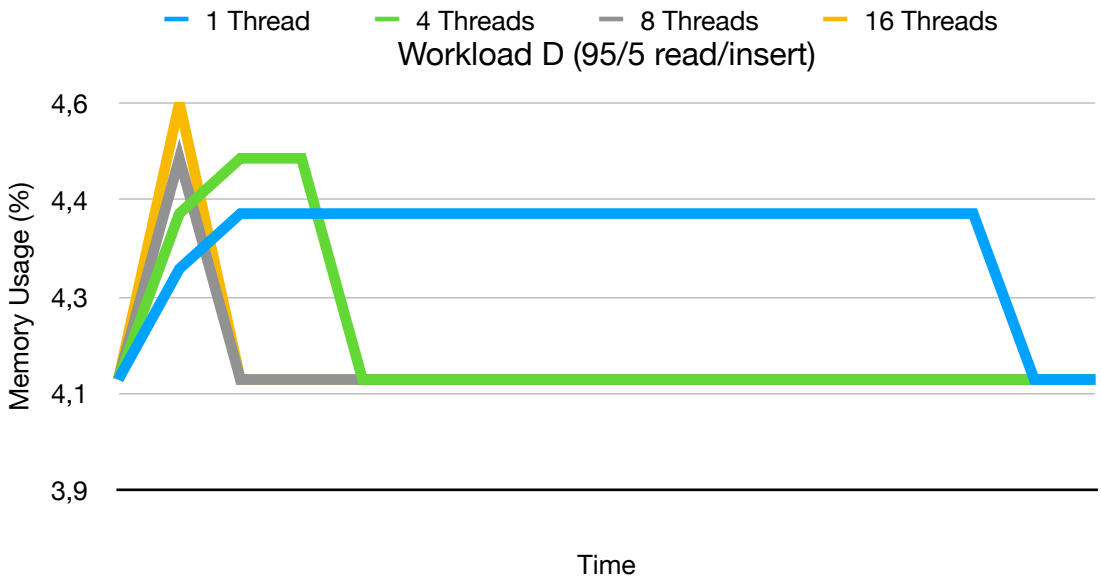
Διάγραμμα 67: Aerospike CPU Usage WorkloadC 6 Nodes



Διάγραμμα 68: Aerospike Memory Usage WorkloadC 6 Nodes

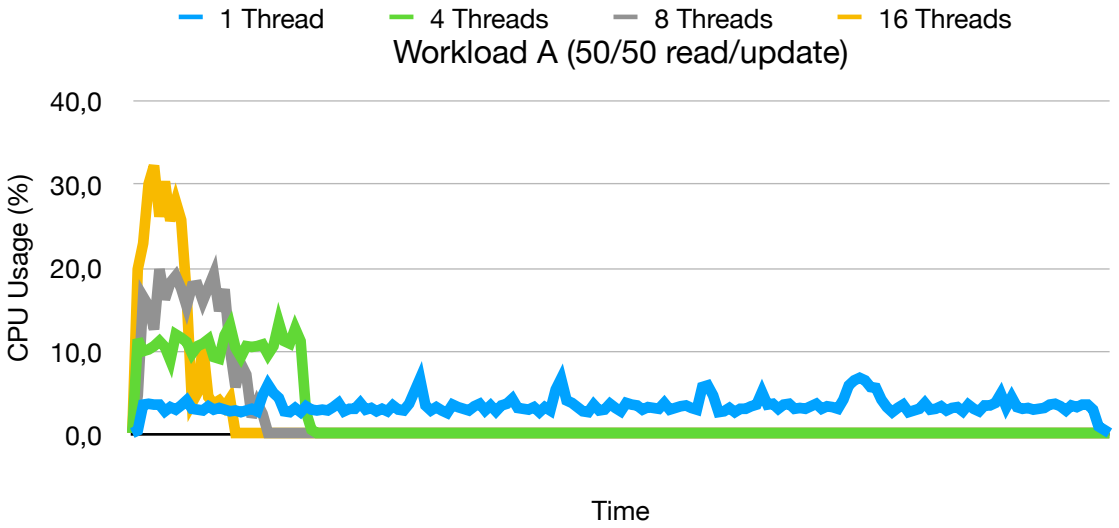


Διάγραμμα 69: Aerospike CPU Usage WorkloadD 6 Nodes

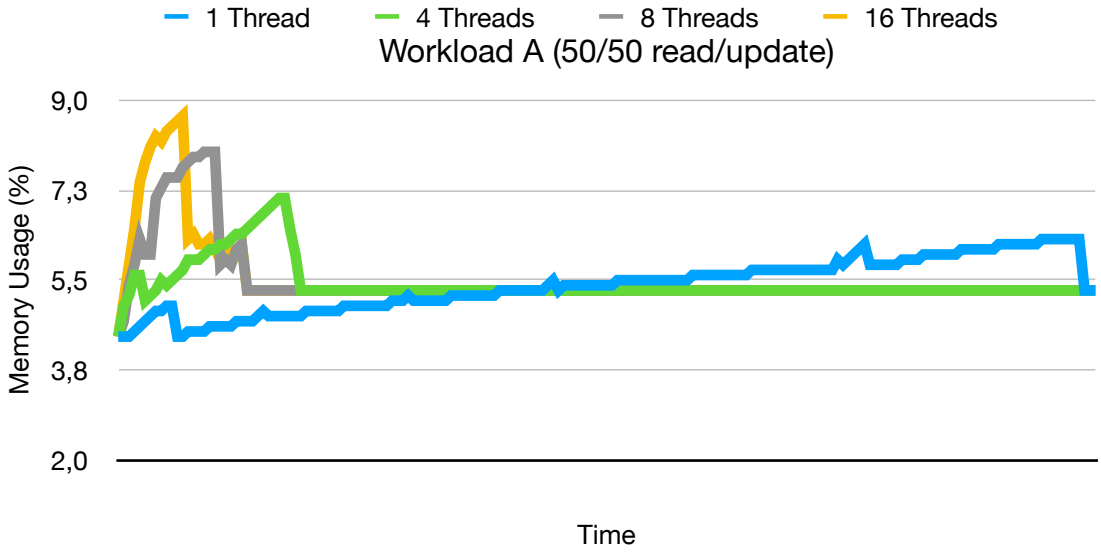


Διάγραμμα 70: Aerospike Memory Usage WorkloadD 6 Nodes

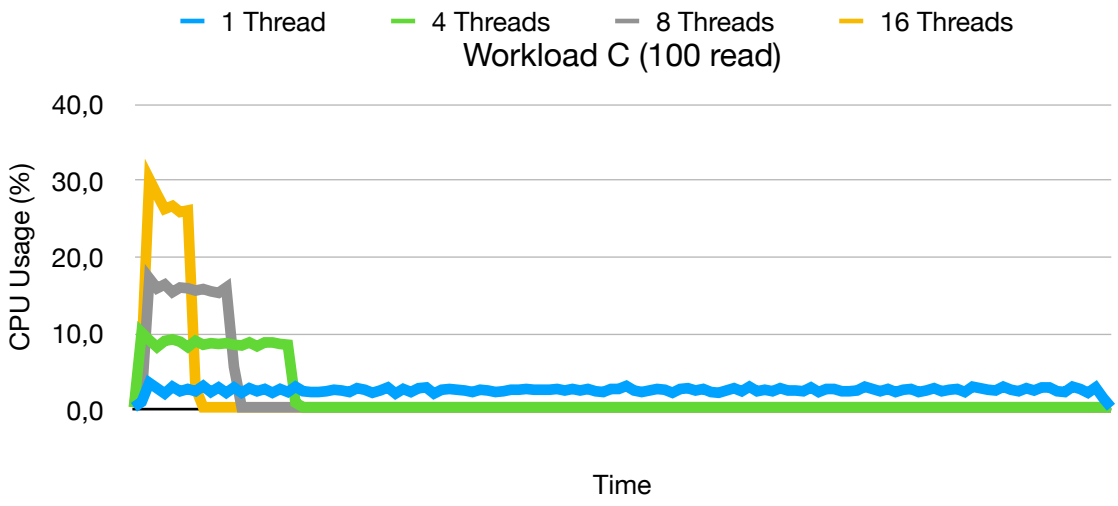
και για την ArangoDB έχουμε:



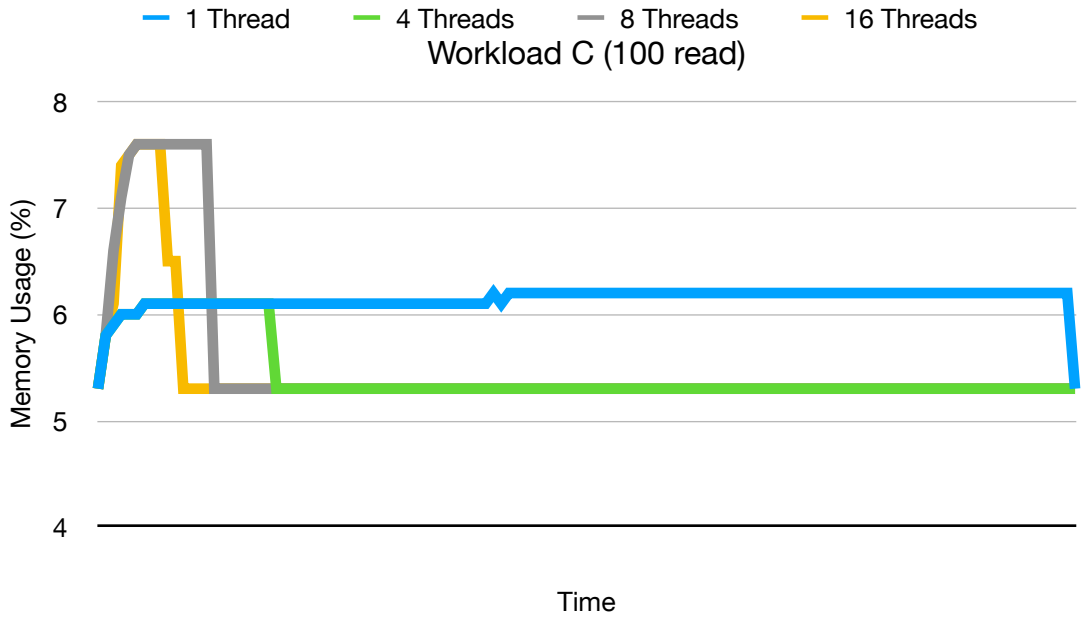
Διάγραμμα 71: ArangoDB CPU Usage WorkloadA 6 Nodes



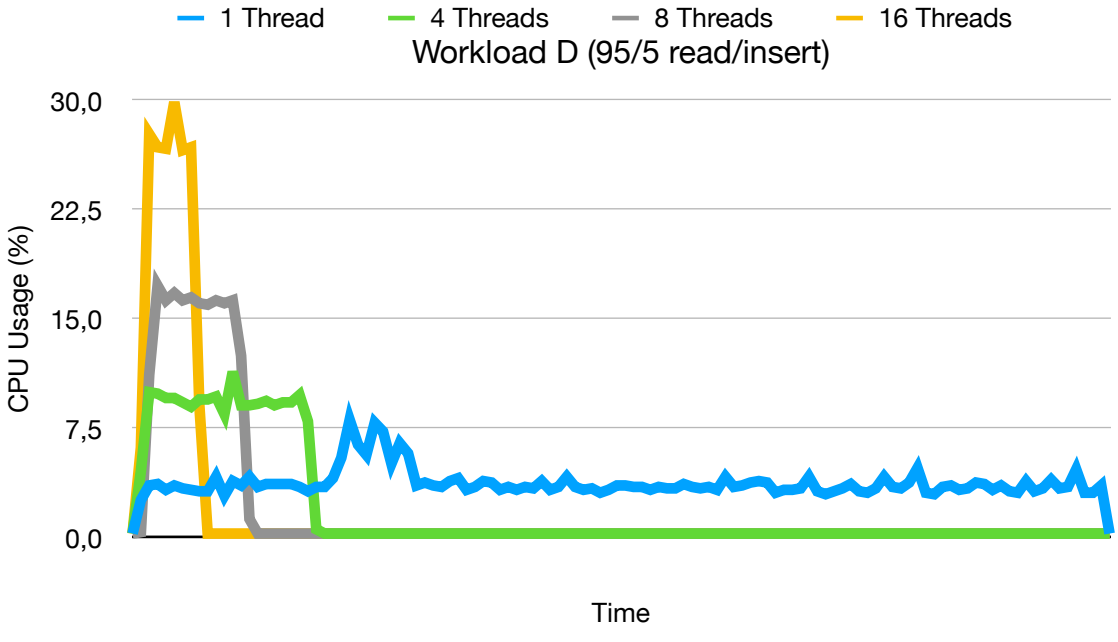
Διάγραμμα 72: ArangoDB Memory Usage WorkloadA 6 Nodes



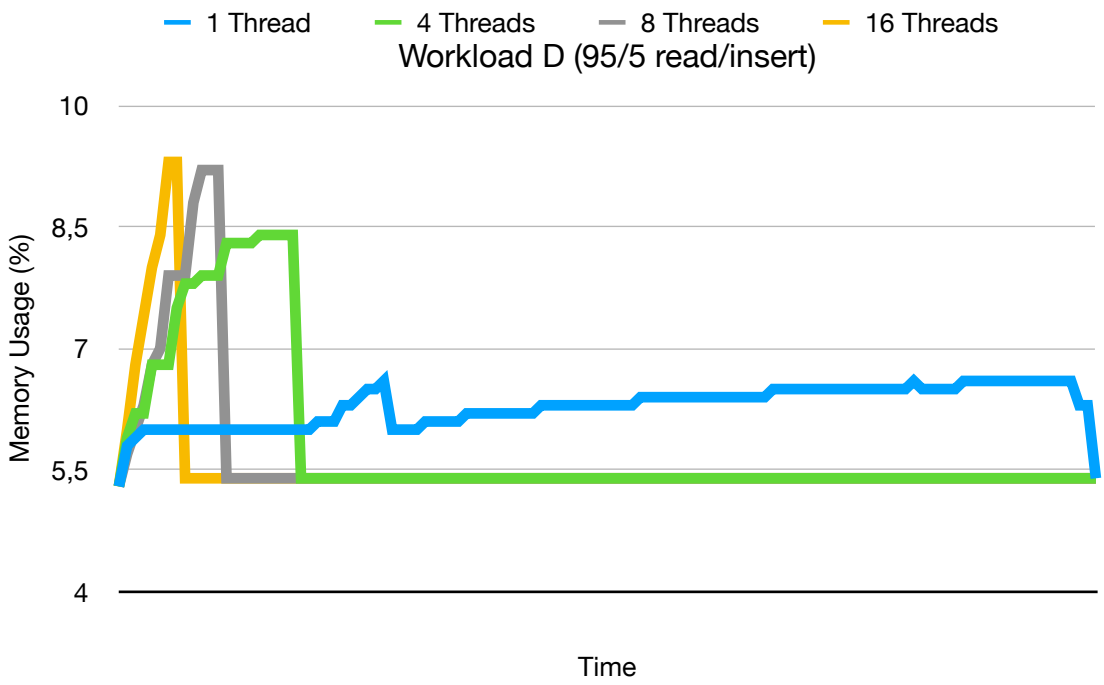
Διάγραμμα 73: ArangoDB CPU Usage WorkloadC 6 Nodes



Διάγραμμα 74: ArangoDB Memory Usage WorkloadC 6 Nodes



Διάγραμμα 75: ArangoDB CPU Usage WorkloadD 6 Nodes



Διάγραμμα 76: ArangoDB Memory Usage WorkloadD 6 Nodes

Παρατηρούμε ότι στο φόρτο εργασίας A (workload A) η Redis όταν λειτουργούν παράλληλα 16 νήματα φτάνει εώς και 16,1% χρησιμοποίηση της ισχύος του επεξεργαστή και ξεκινώντας από το 4,9% φτάνει και μέχρι 5,4% χρησιμοποίηση της μνήμης. Για την Aerospike παρατηρούμε ότι όταν λειτουργούν παράλληλα 16 νήματα φτάνει εώς και 31,1% χρησιμοποίηση της ισχύος του επεξεργαστή και ξεκινώντας από το 4,1% φτάνει και μέχρι 4,6% χρησιμοποίηση της μνήμης. Για την ArangoDB παρατηρούμε ότι όταν λειτουργούν παράλληλα 16 νήματα φτάνει εώς και 32,3% χρησιμοποίηση της ισχύος του επεξεργαστή και ξεκινώντας από το 4,4% φτάνει και μέχρι 7,9% χρησιμοποίηση της μνήμης. Για το φόρτο εργασίας C (workload C) η Redis όταν λειτουργούν παράλληλα 16 νήματα φτάνει εώς και 11,8% χρησιμοποίηση της ισχύος του επεξεργαστή και ξεκινώντας από το 4,9% φτάνει και μέχρι 6% χρησιμοποίηση της μνήμης. Για την Aerospike παρατηρούμε ότι όταν λειτουργούν παράλληλα 16 νήματα φτάνει εώς και 30,7% χρησιμοποίηση της ισχύος του επεξεργαστή και ξεκινώντας από το 4,1% φτάνει και μέχρι 4,6% χρησιμοποίηση της μνήμης. Για την ArangoDB παρατηρούμε ότι όταν λειτουργούν παράλληλα 16 νήματα φτάνει εώς και 30,2% χρησιμοποίηση της ισχύος του επεξεργαστή και ξεκινώντας από το 5,3% φτάνει και μέχρι 7,6% χρησιμοποίηση της μνήμης. Για το φόρτο εργασίας D (workload D) η Redis όταν λειτουργούν παράλληλα 16 νήματα φτάνει εώς και 13,6% χρησιμοποίηση της ισχύος του επεξεργαστή και ξεκινώντας από το 4,9% φτάνει και μέχρι 6% χρησιμοποίηση της μνήμης. Για την Aerospike παρατηρούμε ότι όταν λειτουργούν παράλληλα 16 νήματα φτάνει εώς και 28,4% χρησιμοποίηση της ισχύος του επεξεργαστή και ξεκινώντας από το 4,1% φτάνει και μέχρι 4,6% χρησιμοποίηση της μνήμης. Για την ArangoDB παρατηρούμε ότι όταν λειτουργούν παράλληλα 16 νήματα φτάνει εώς και 29,7% χρησιμοποίηση της ισχύος του επεξεργαστή και ξεκινώντας από το 5,3% φτάνει και μέχρι 9,3% χρησιμοποίηση της μνήμης. Από τα παραπάνω αποτελέσματα παρατηρούμε ότι η χρήση υπολογιστικής ισχύς έχει αυξηθεί σε όλες τις βάσεις δεδομένων, που είναι λογικό καθώς πλέον έχουν αυξηθεί οι κόμβοι των συμπλεγμάτων. Επίσης για την ArangoDB

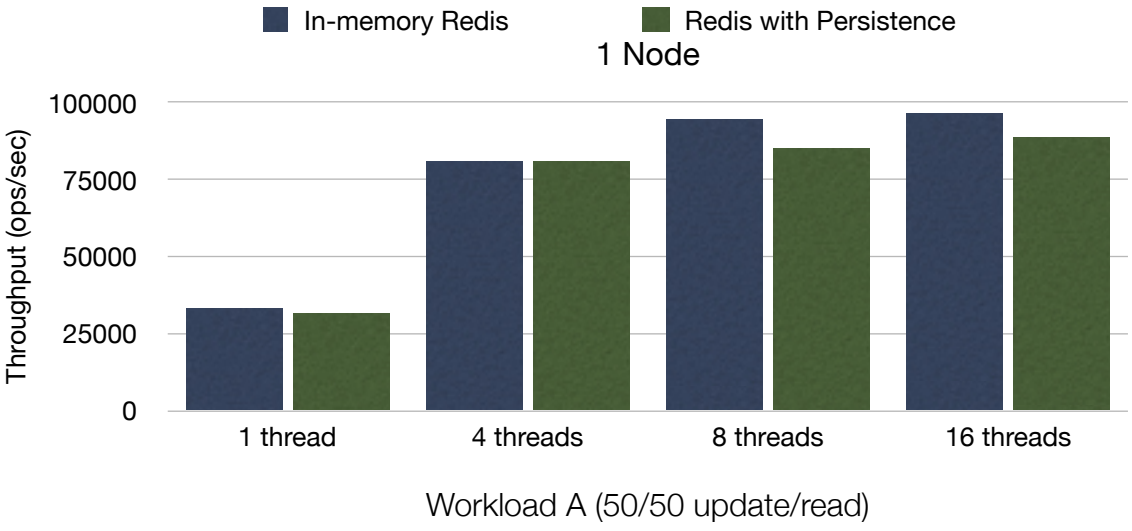
παρατηρούμε ότι σε αυτή τη διάταξη, το σημείο αναφοράς της ισχύος του επεξεργαστή είναι το 0,3% και όχι το 0% όπως σε όλες τις άλλε περιπτώσεις. Αυτό συμβαίνει διότι η ArangoDB πέρα από τους κόμβους-εξυπηρετητές που έχει, σηκώνει επίσης του Συντονιστές και τα Μέσα έτσι ώστε να μπορεί να λειτουργήσει πιο αποδοτικά το σύμπλεγμα.

5.2.3 Πειραματικά Αποτελέσματα σε Πραγματικό Περιβάλλον - In Memory Redis VS Redis with Persistence

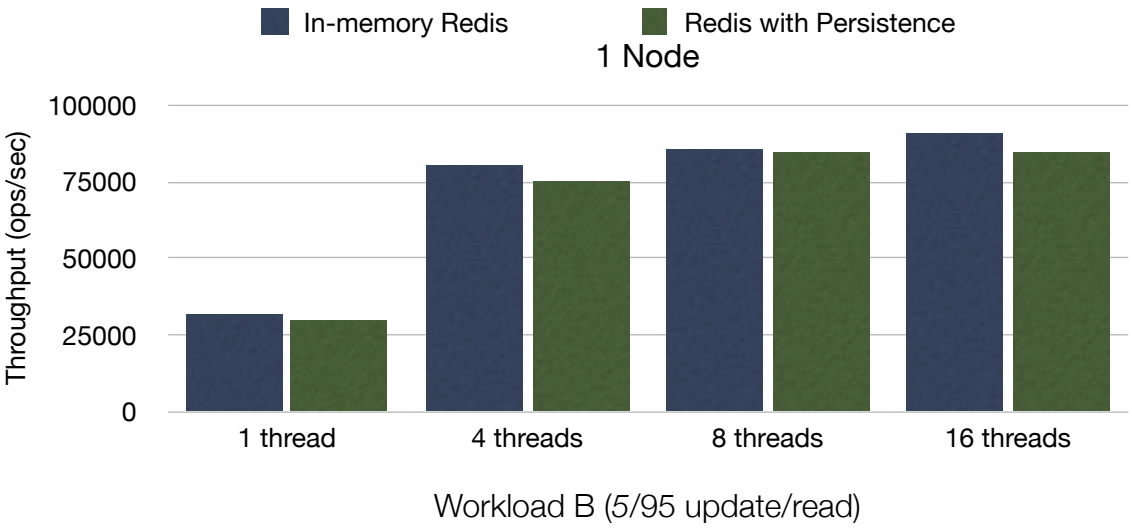
Μία άλλη διάσταση που θα δώσουμε στα τεστ μας είναι να συγκρίνουμε τις επιδόσεις σε λειτουργία μνήμης και σε λειτουργία μόνιμης αποθήκευσης της Redis. Ο λόγος που δεν μπορούμε να κάνουμε το ίδιο και με την Aerospike είναι επειδή η Aerospike δεν υποστηρίζει λειτουργία μόνιμης αποθήκευσης σε σκληρούς δίσκους HDD, που έχει το μηχάνημά μας. Για την λειτουργία αυτή θα βγάλουμε από σχόλια τις παρακάτω γραμμές στο αρχείο redis.conf

```
# save 900 1 –> save 900 1
# save 300 10 –> save 300 10
# save 60 10000 –> save 60 10000
```

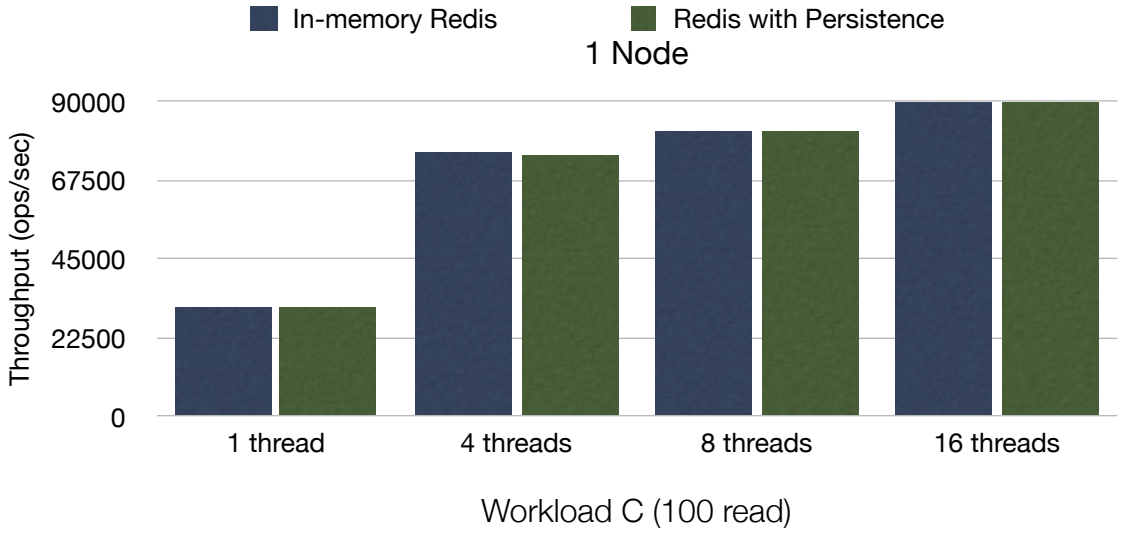
Αυτό θα ενεργοποιήσει τη λειτουργία ***snapshotting*** της Redis. Η λειτουργία αυτή επιτρέπει στη Redis να γράφει τα αρχεία στο δίσκο σε ένα δυαδικό αρχείο με όνομα ***dump.rdb***. Οι παραπάνω γραμμές ερμηνεύονται με τον εξής τρόπο. Αν αλλάξουν παραπάνω από 900 κλειδιά τότε η Redis αποθηκεύει κάθε δευτερόλεπτο όσα περισσότερα δεδομένα στο δίσκο μπορεί μέχρι να τελειώσει. Αντίστοιχα η δεύτερη γραμμή ερμηνεύεται ότι στην περίπτωση που αλλάξουν τουλάχιστον 300 κλειδιά τότε τότε η Redis αποθηκεύει κάθε 10 δευτερόλεπτα όσα περισσότερα δεδομένα στο δίσκο μπορεί μέχρι να τελειώσει. Αντίστοιχα ερμηνεύεται και η τρίτη γραμμή. Προκειμένου να γίνει αυτή η αποθήκευση τη στιγμή που εκπληρωθεί μια από τις παραπάνω προϋποθέσεις τότε εκτελεί την εντολή των linux fork() [33] και δημιουργεί δύο διεργασίες. Μια διεργασία-γονέα και μια διεργασία-παιδί. Η διεργασία-παιδί αρχίζει και γράφει τα δεδομένα στο δίσκο σε ένα προσωρινό. Όταν τελειώσει αυτή τη διαδικασία ενημερώνει το αρχείο διεργασίας-γονεά ***dump.rdb*** και έπειτα σταματά τη λειτουργία του. Αυτή η διαδικασία εκμεταλλεύεται την τεχνική copy-on-write.Τα αποτελέσματα τα οποία συλλέγουμε για τη διεκπεραιωτική ικανότητα σε λειτουργία μονού κόμβου και στις δύο περιπτώσεις εκτέλεσης είναι τα:



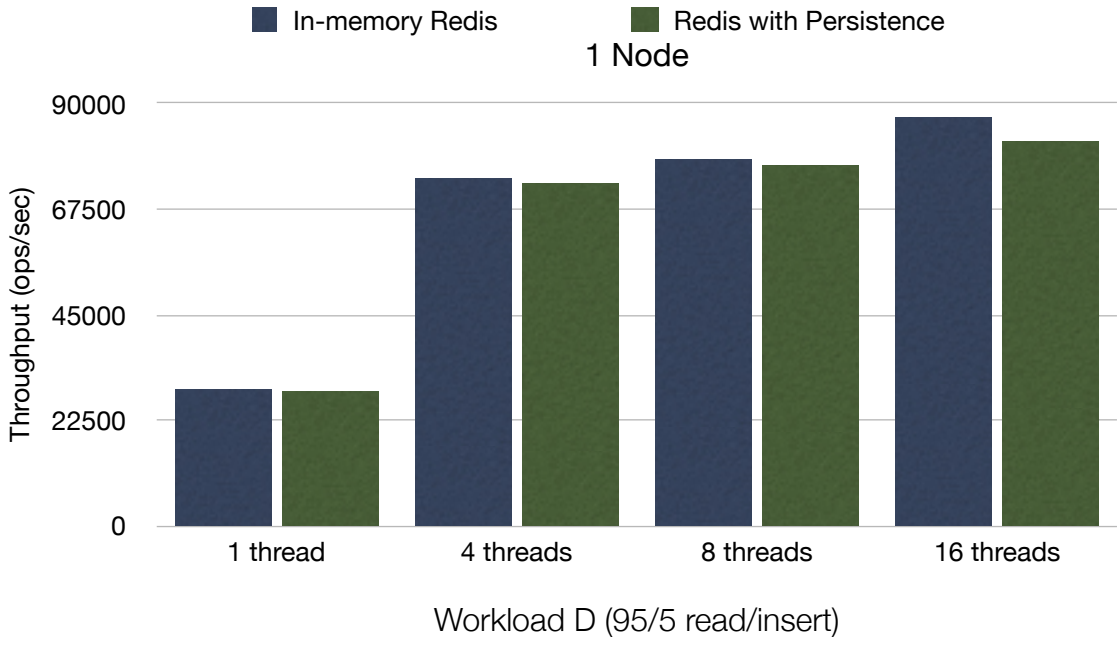
Διάγραμμα 77: In memory VS Persistent Redis WorkloadA Throughput



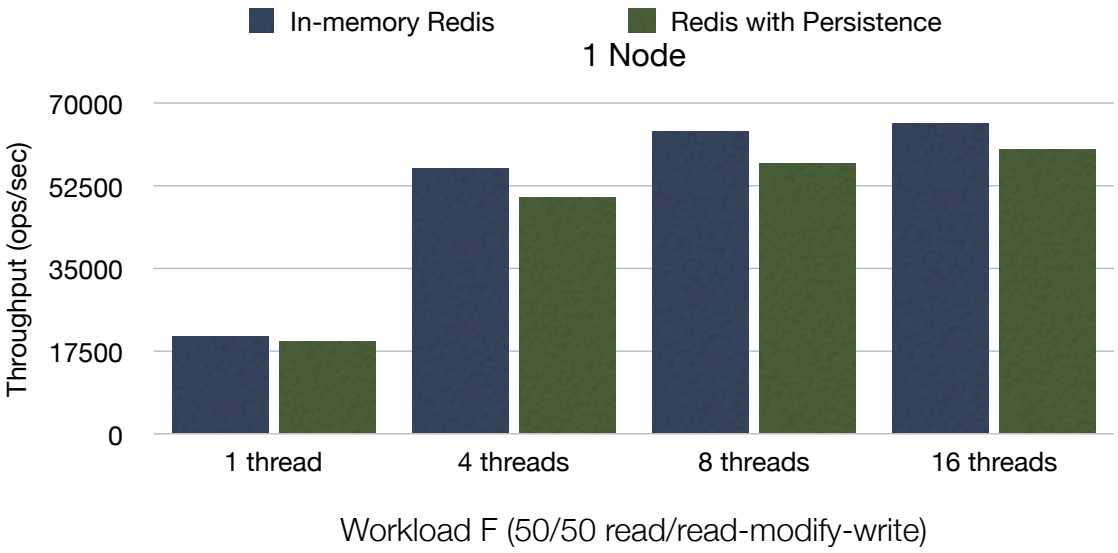
Διάγραμμα 78: In memory VS Persistent Redis WorkloadB Throughput



Διάγραμμα 79: In memory VS Persistent Redis WorkloadC Throughput

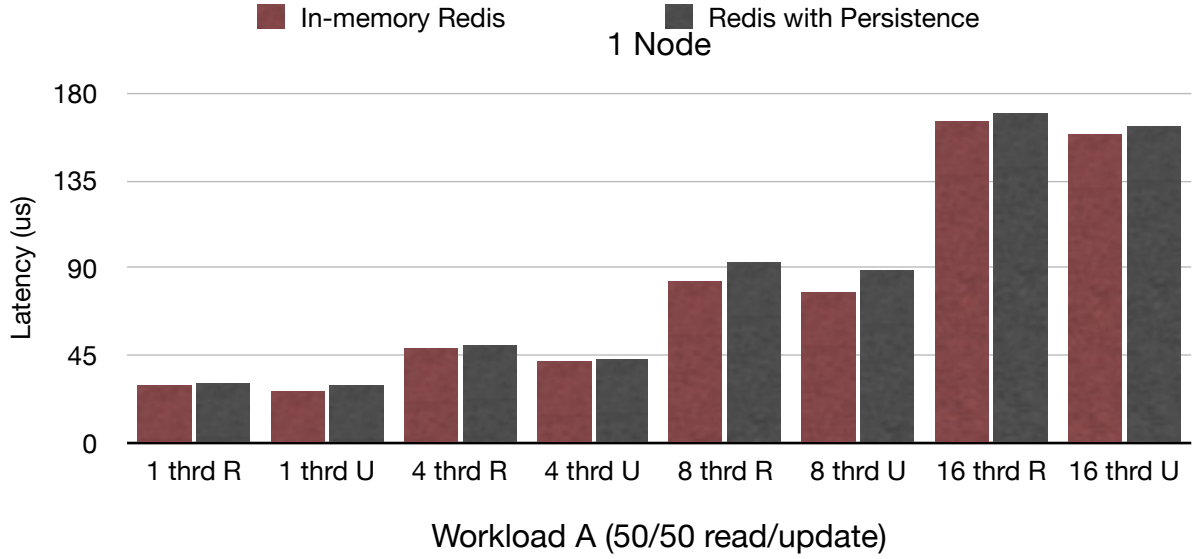


Διάγραμμα 80: In memory VS Persistent Redis WorkloadD Throughput

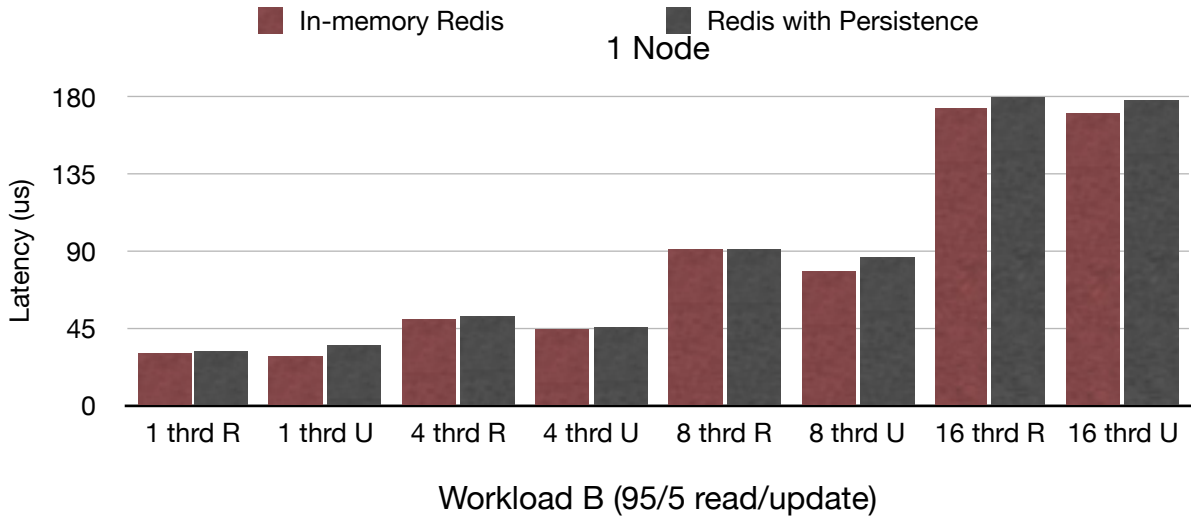


Διάγραμμα 81: In memory VS Persistent Redis WorkloadF Throughput

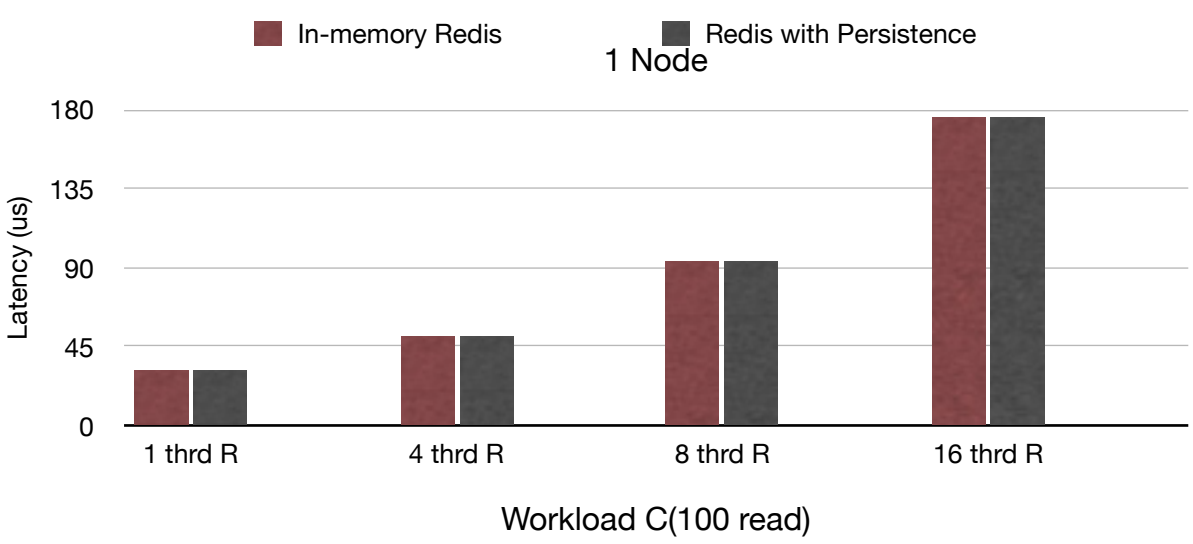
και για την χρόνο καθυστέρησης εκτέλεσης της κάθε εντολής (latency) έχουμε:



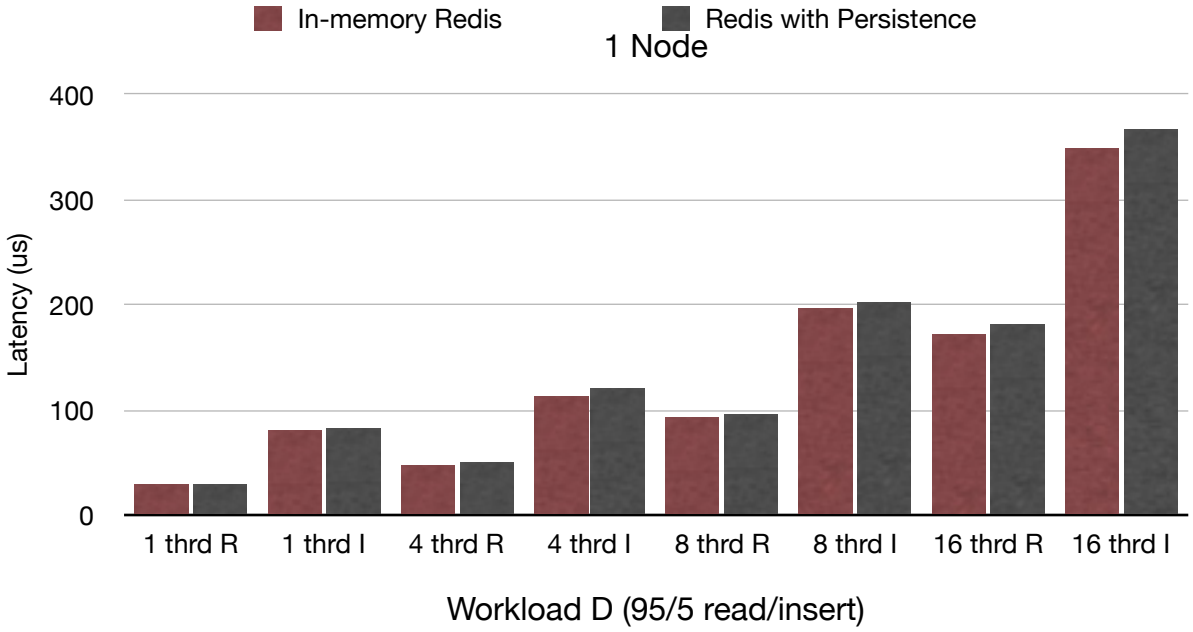
Διάγραμμα 82: In memory VS Persistent Redis WorkloadA Latency



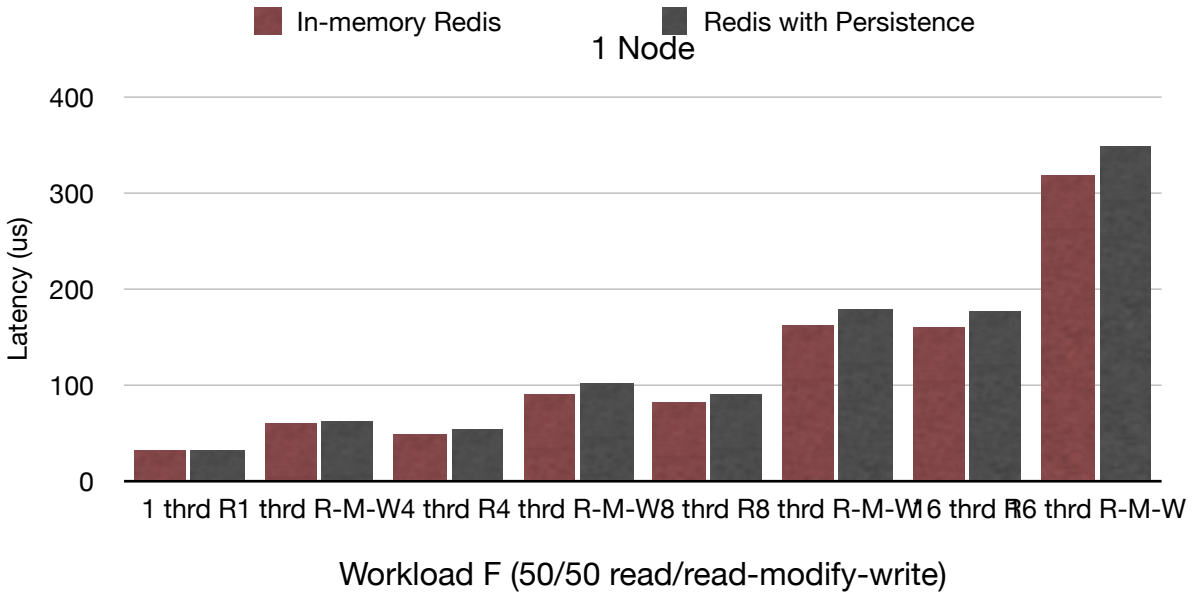
Διάγραμμα 83: In memory VS Persistent Redis WorkloadB Latency



Διάγραμμα 84: In memory VS Persistent Redis WorkloadC Latency



Διάγραμμα 85: In memory VS Persistent Redis WorkloadD Latency



Διάγραμμα 86: In memory VS Persistent Redis WorkloadF Latency

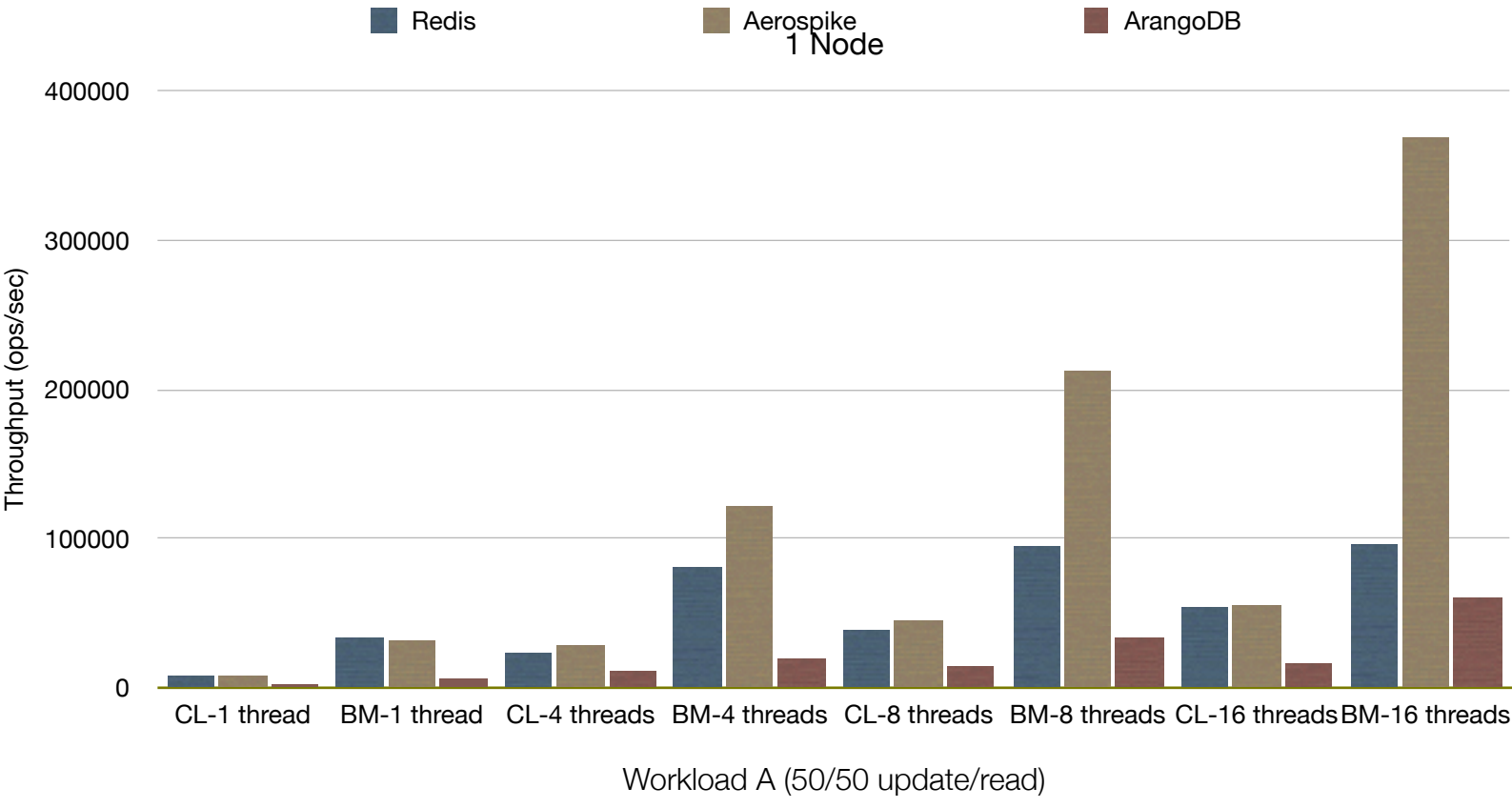
Από τα παραπάνω διαγράμματα παρατηρούμε ότι σε κάθε περίπτωση η In-memory Redis είναι πιο γρήγορη από την Redis με Persistence εκτός από τον φόρτο εργασίας όπου οι δύο παραμετροποιήσεις αυτές δίνουν το ίδιο αποτέλεσμα. Συγκεκριμένα παρατηρούμε ότι στο φόρτο εργασίας A (workload A) η In-memory Redis φτάνει εώς και 96,571 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 165 us και 158 us για τις εντολές ανάγνωσης (read) και τις εντολές ανανέωσης (update). Η Redis με Persistence φτάνει εώς και 88,621 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 170 us και 63 us για τις εντολές ανάγνωσης (read) και τις εντολές ανανέωσης (update). Για το φόρτο εργασίας B (workload B) η In-memory Redis φτάνει εώς και 90,514 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 173 us και 70 us για τις εντολές ανάγνωσης (read) και τις εντολές ανανέωσης (update). Η Redis με Persistence φτάνει εώς και 84,527 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 180 us και 178 us για τις εντολές ανάγνωσης (read) και τις εντολές ανανέωσης (update). Για το φόρτο εργασίας C (workload C) η In-memory Redis φτάνει εώς και 89,887 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 175 us για τις εντολές ανάγνωσης (read). Η Redis με Persistence φτάνει εώς και 89,590 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 175 us για τις εντολές ανάγνωσης (read). Για το φόρτο εργασίας D (workload D) η In-memory Redis φτάνει εώς και 86,873 εντολές/ δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 171 us και 348 us για τις εντολές ανάγνωσης (read) και τις εντολές εισαγωγής (insert). Η Redis με Persistence φτάνει εώς και 82,277 εντολές/ δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 181 us και 367 us για τις εντολές ανάγνωσης (read) και τις εντολές εισαγωγής (insert). Για το φόρτο εργασίας F (workload F) η In-memory Redis φτάνει εώς και 66,155 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 160 us και 317 us για τις εντολές ανάγνωσης (read) και τις εντολές ανάγνωσης-τροποποίησης-εγγραφής(read-modify-write). Η Redis με Persistence φτάνει εώς και 60,346 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 177 us και 349 us για τις εντολές ανάγνωσης (read) και τις εντολές ανάγνωσης-τροποποίησης-εγγραφής(read-modify-write). Τα παραπάνω αποτελέσματα είναι αναμενόμενα καθώς η όταν η Redis εκτελείται στη μνήμη είναι ταχύτερη από μια αντιστοιχη Redis η οποία έχει ενεργοποιημένη την αποθήκευση δεδομένων στο δίσκο. Βέβαια όπως παρατηρούμε η διαφορές δεν είναι μεγάλες το οποίο αναδεικνύει την εξαιρετική απόδοση της Redis τόσο σαν in-memory βάση δεδομένων όσο και σαν βάση δεδομένων που αποθηκεύει τα δεδομένα στο δίσκο.

5.2.4 Πειραματικά Αποτελέσματα σε Εικονικό Περιβάλλον - Throughput-Latency

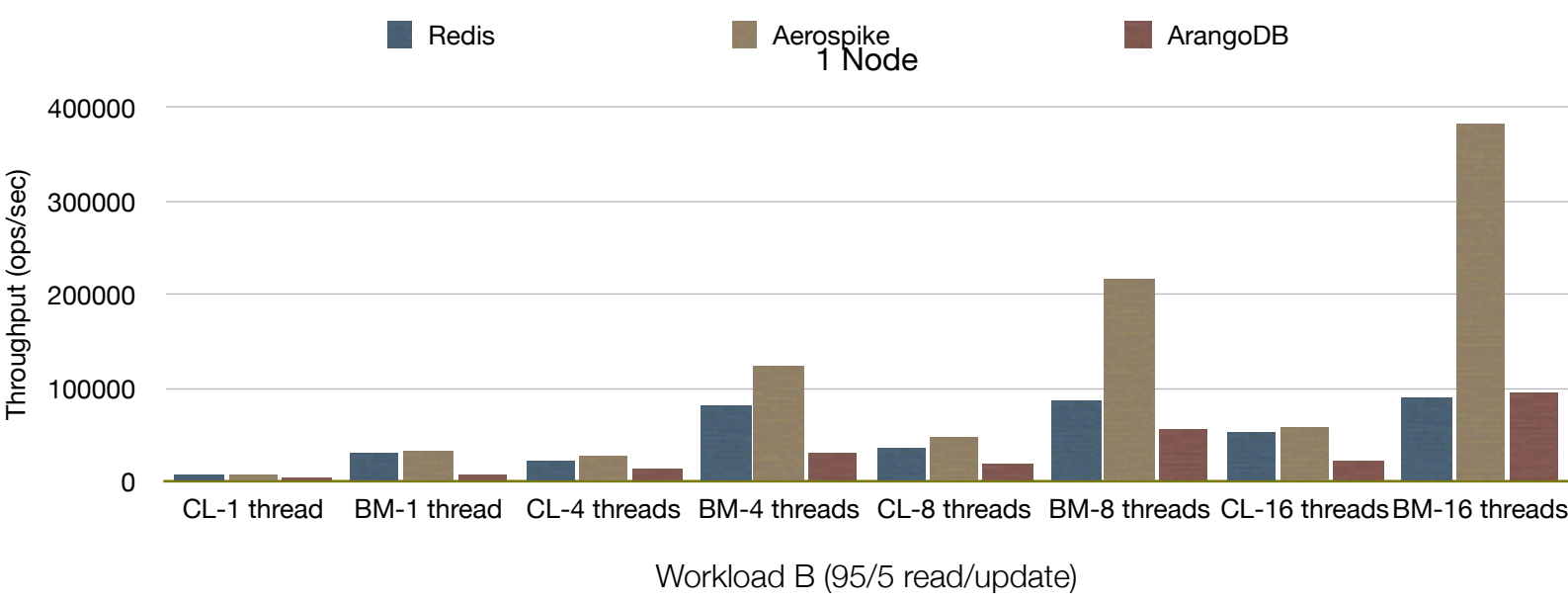
Τέλος, προχωρούμε με τα τεστ στο εικονικό περιβάλλον. Έχοντας ολοκληρώσει την παραμετροποίηση συνδεόμαστε στον κόμβο που περιέχει το εργαλείο YCSB φορτώνουμε τα δεδομένα με τις εξής εντολές για κάθε βάση αντίστοιχα:

```
$ cd ycsb-0.17.0/
$ ./bin/ycsb load redis -s -P workloads/workloada -p “redis.host=node_ip”
-p "redis.port=6379" -threads 16 > outputLoad.txt
$ ./bin/ycsb load aerospike -s -P workloads/workloada -p as.host=node_ip
-p as.namespace=test -threads 16 > outputLoad.txt
$ ./bin/ycsb run arangodb -s -P workloads/workloada -p arangodb.ip=node_ip
-p arangodb.port=8529 -threads 16 > outputLoad.txt
```

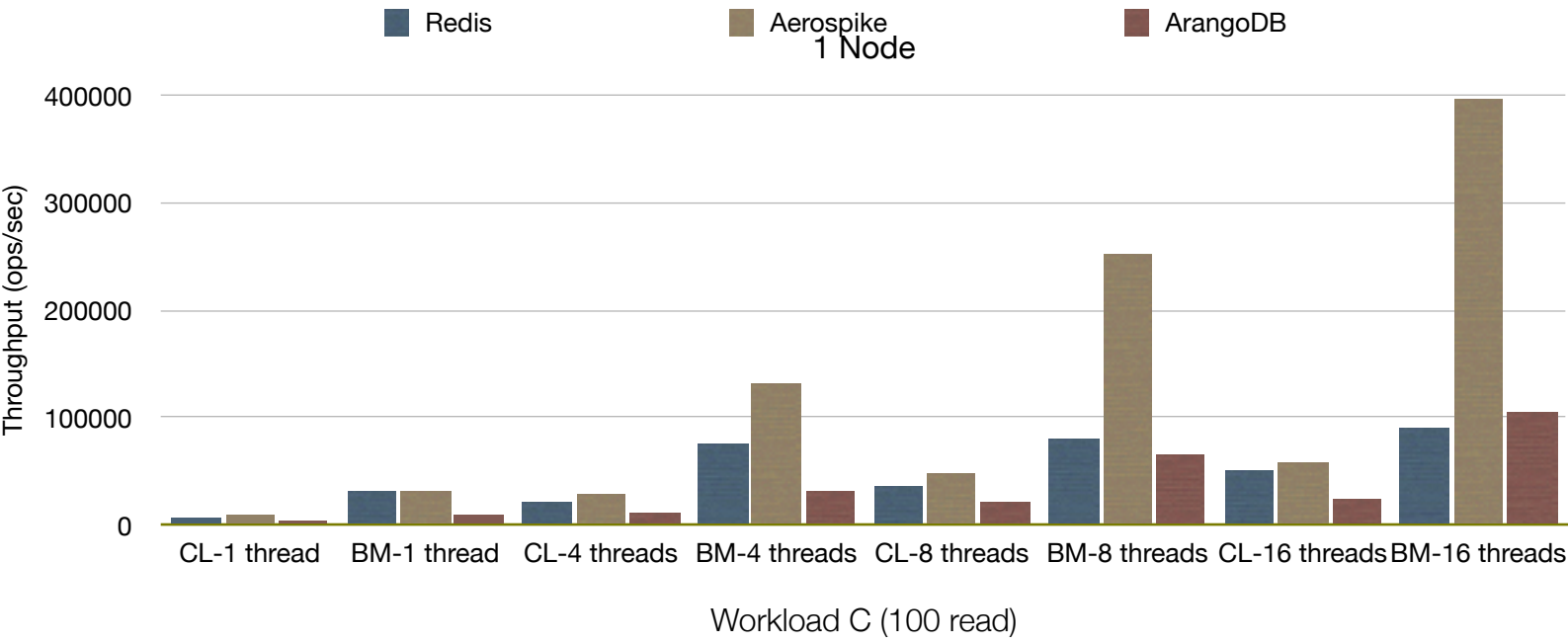
Τα αποτελέσματα που παίρνουμε εκτελώντας κάθε βάση σε λειτουργία μονού κόμβου με τους αντίστοιχους φόρτους εργασίας και με τη σειρά που αναφέραμε σε προηγούμενα πειράματα, είναι τα εξής για την διεκπεραιωτική ικανότητα:



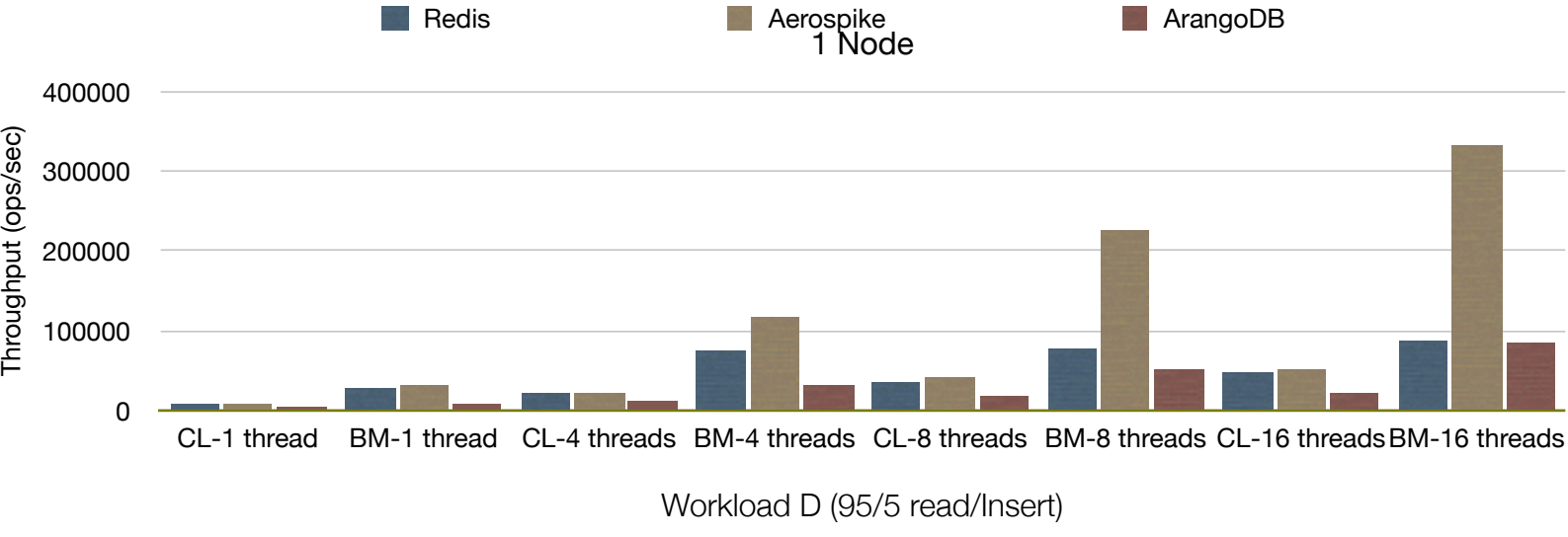
Διάγραμμα 87: Cloud vs Bare Metal Throughput WorkloadA 1 Node



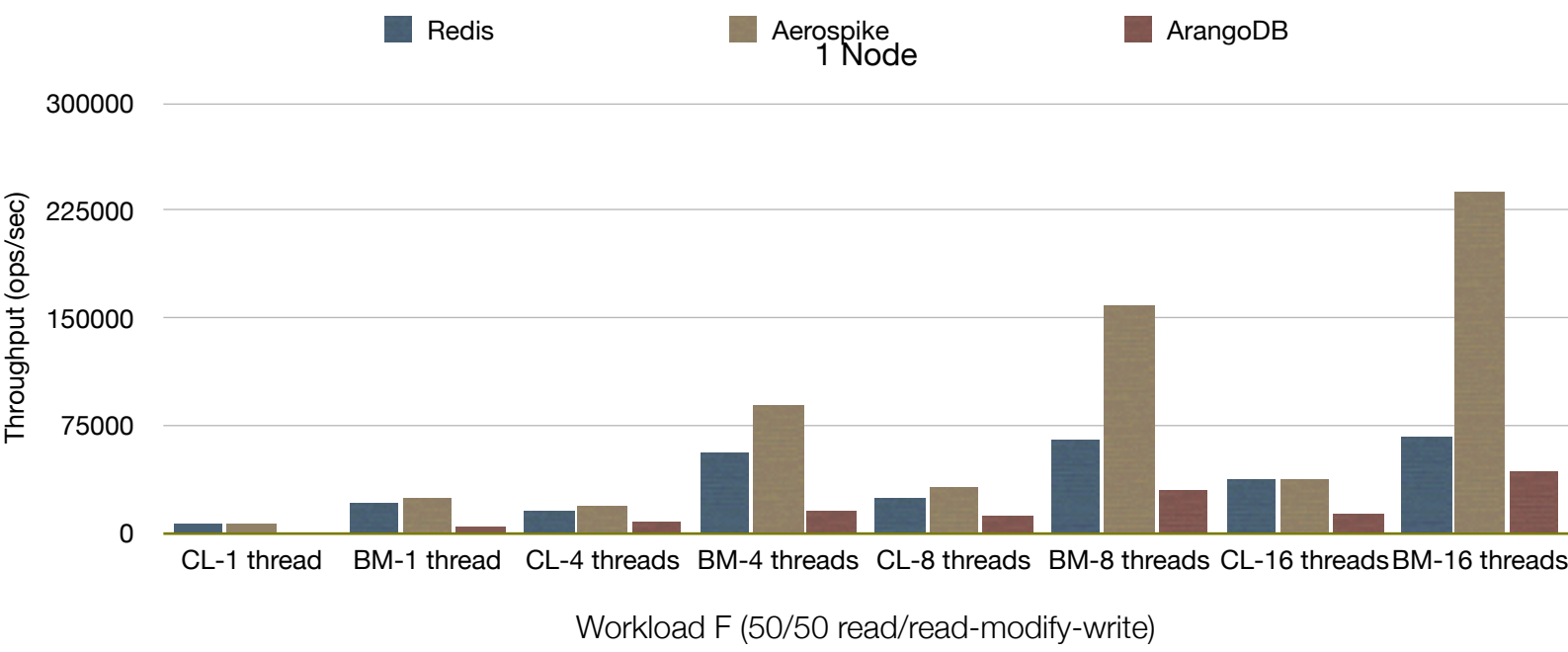
Διάγραμμα 88: Cloud vs Bare Metal Throughput WorkloadB 1 Node



Διάγραμμα 89: Cloud vs Bare Metal Throughput WorkloadC 1 Node

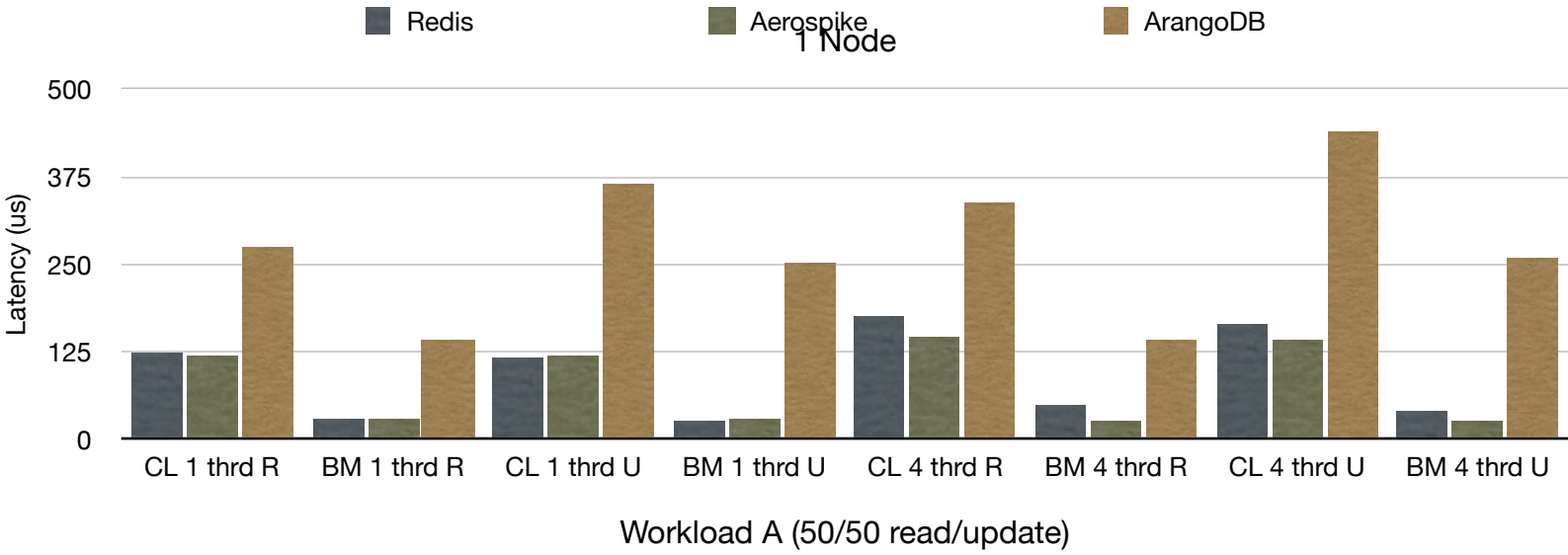


Διάγραμμα 90: Cloud vs Bare Metal Throughput WorkloadD 1 Node

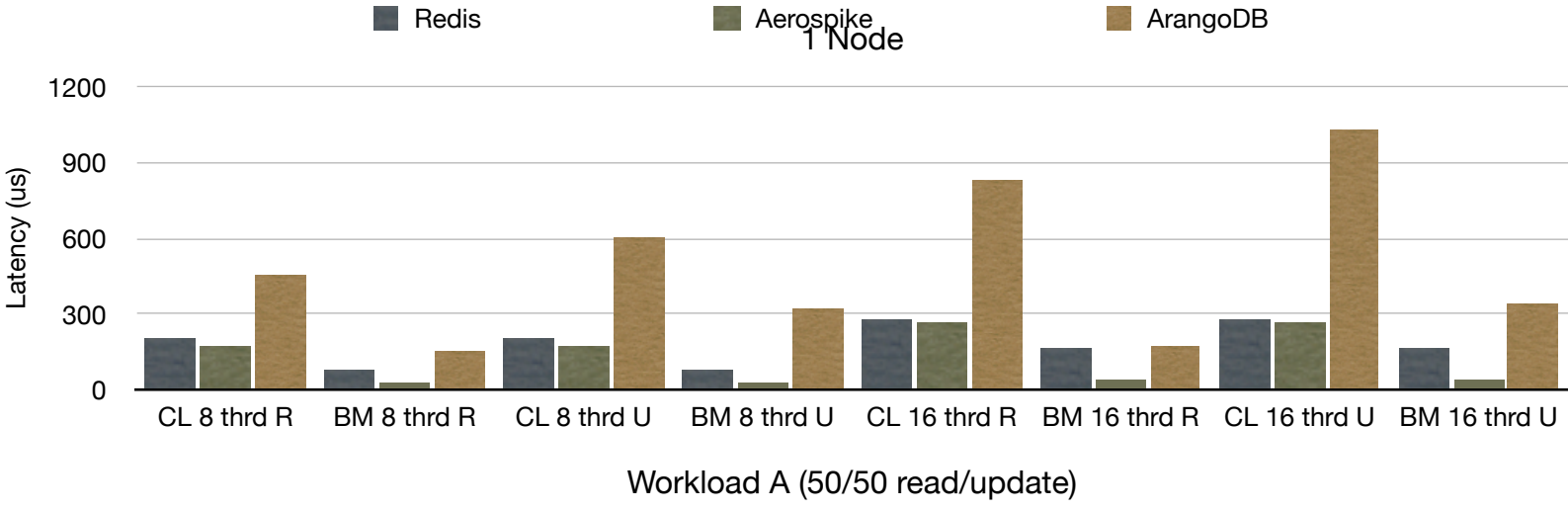


Διάγραμμα 91: Cloud vs Bare Metal Throughput WorkloadF 1 Node

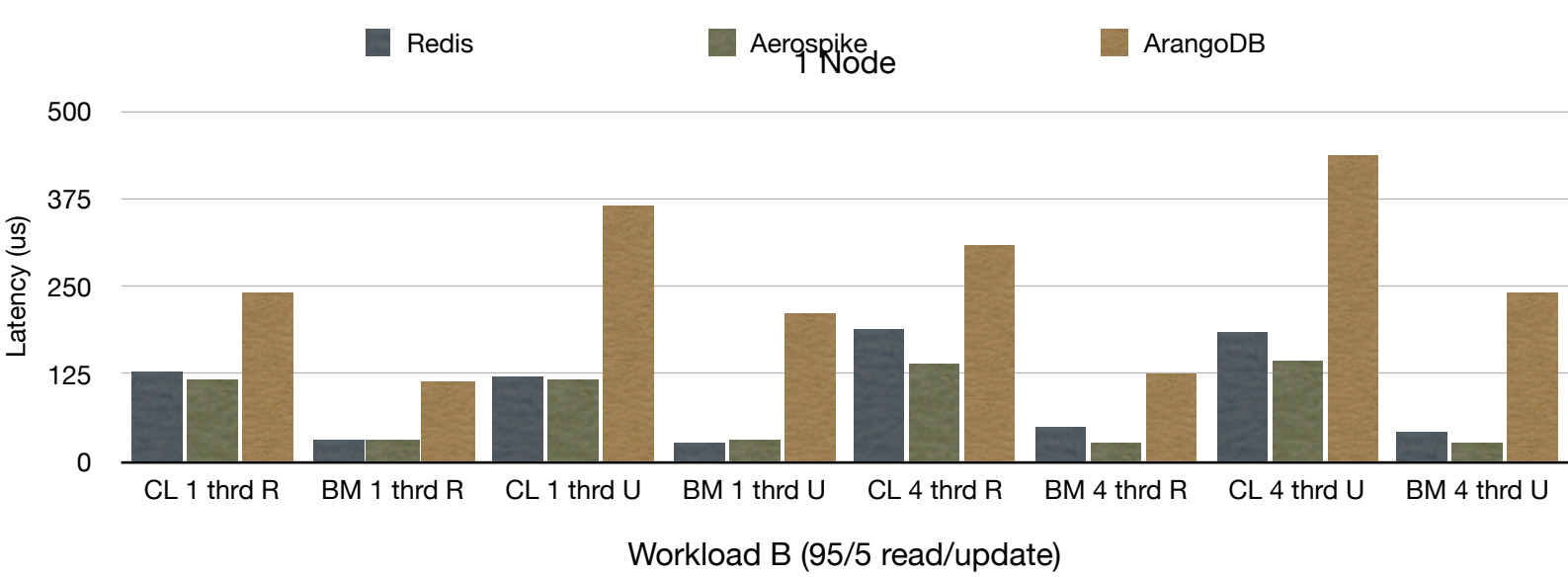
και για την χρόνο καθυστέρησης εκτέλεσης της κάθε εντολής (latency) έχουμε:



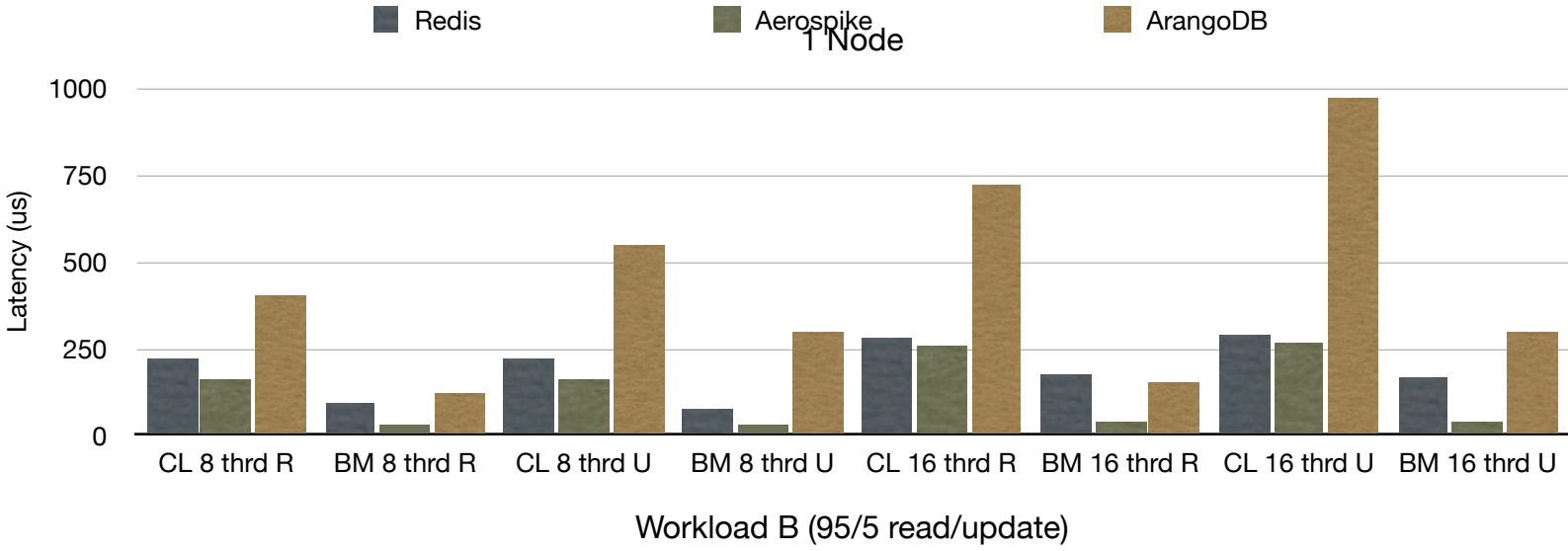
Διάγραμμα 92: Cloud vs Bare Metal Latency WorkloadA 1 Node 1-4 Threads



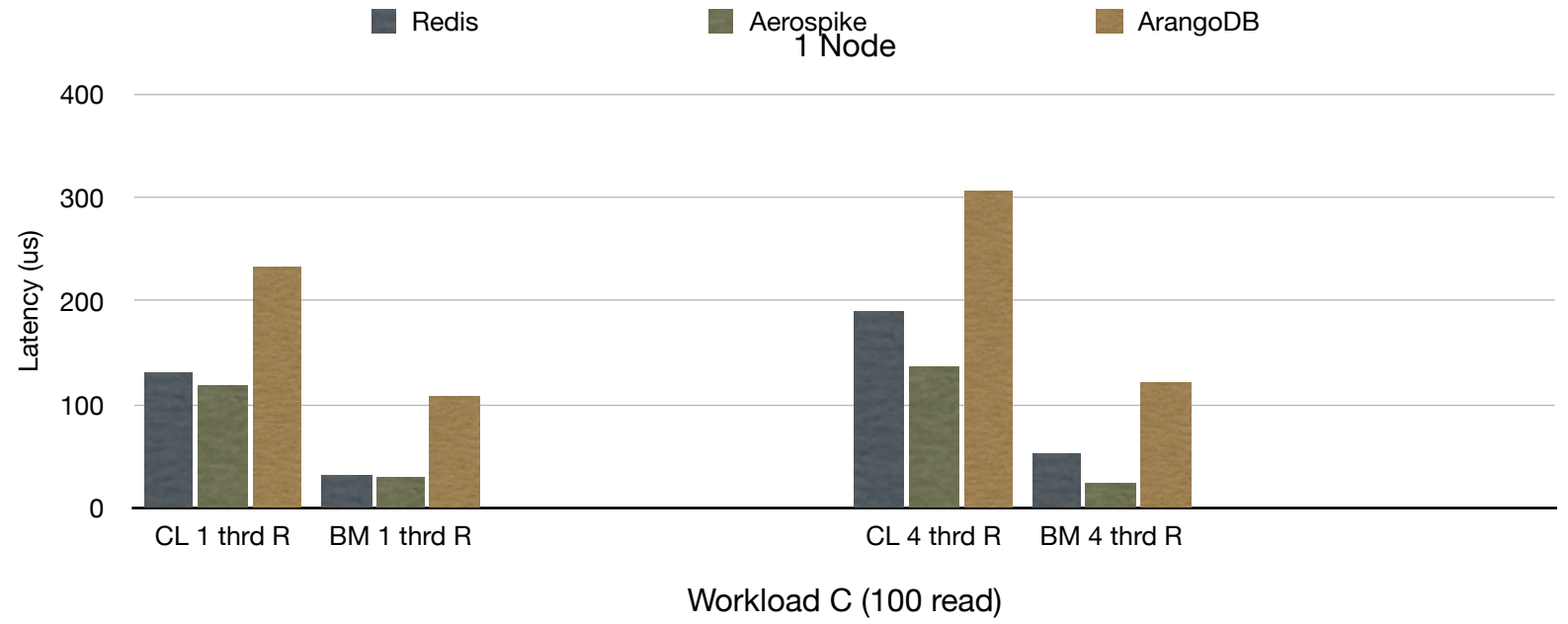
Διάγραμμα 93: Cloud vs Bare Metal Latency WorkloadA 1 Node 8-16 Threads



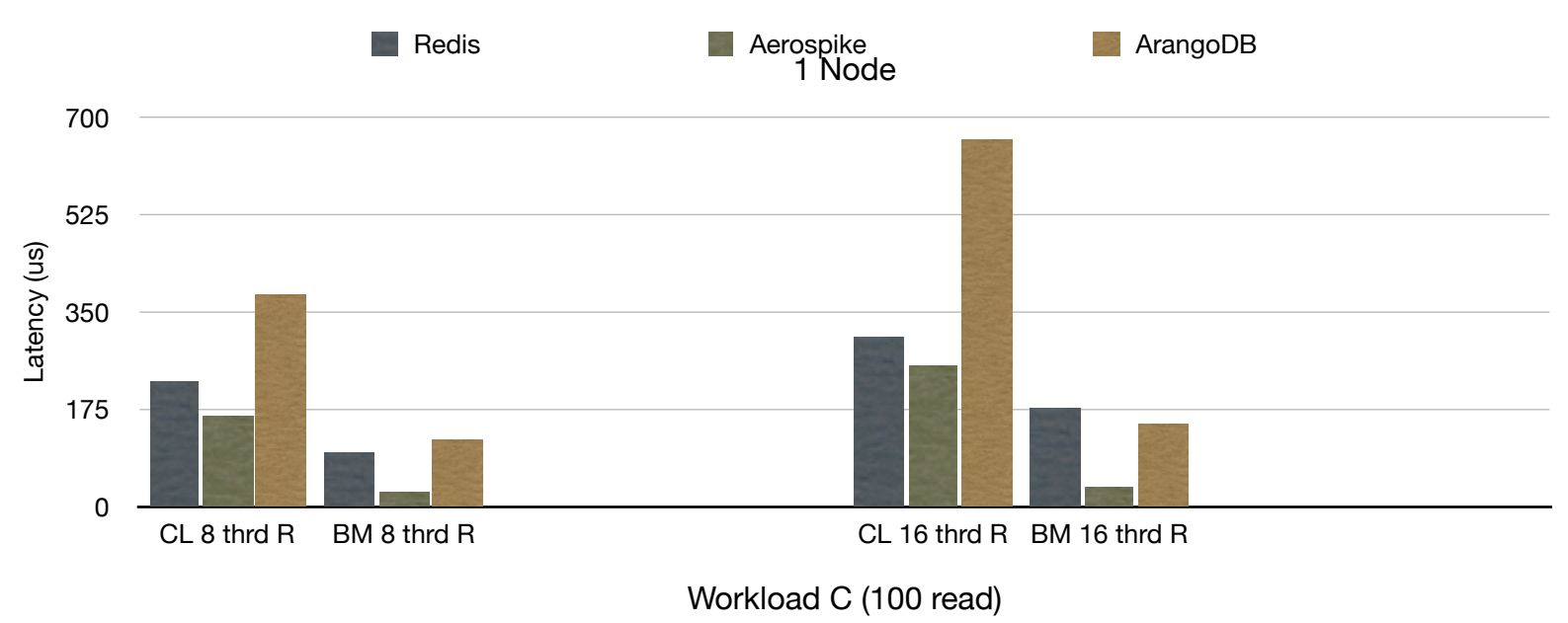
Διάγραμμα 94: Cloud vs Bare Metal Latency WorkloadB 1 Node 1-4 Threads



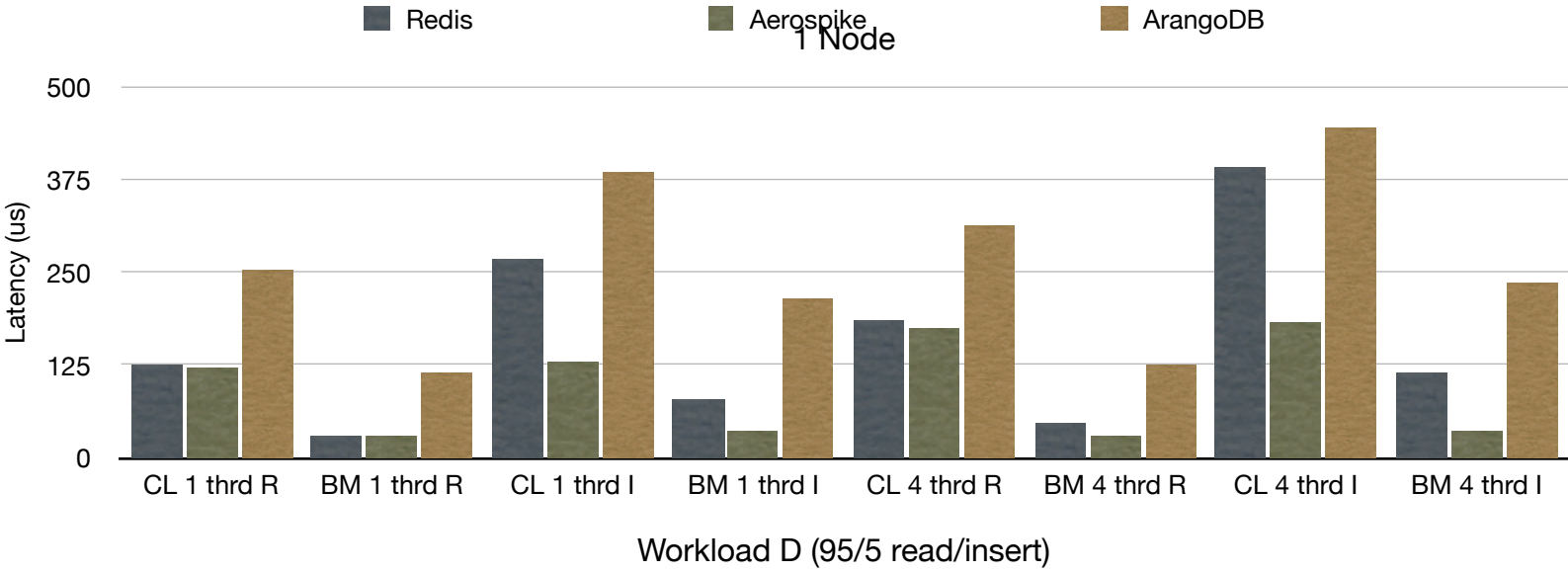
Διάγραμμα 95: Cloud vs Bare Metal Latency WorkloadB 1 Node 8-16 Threads



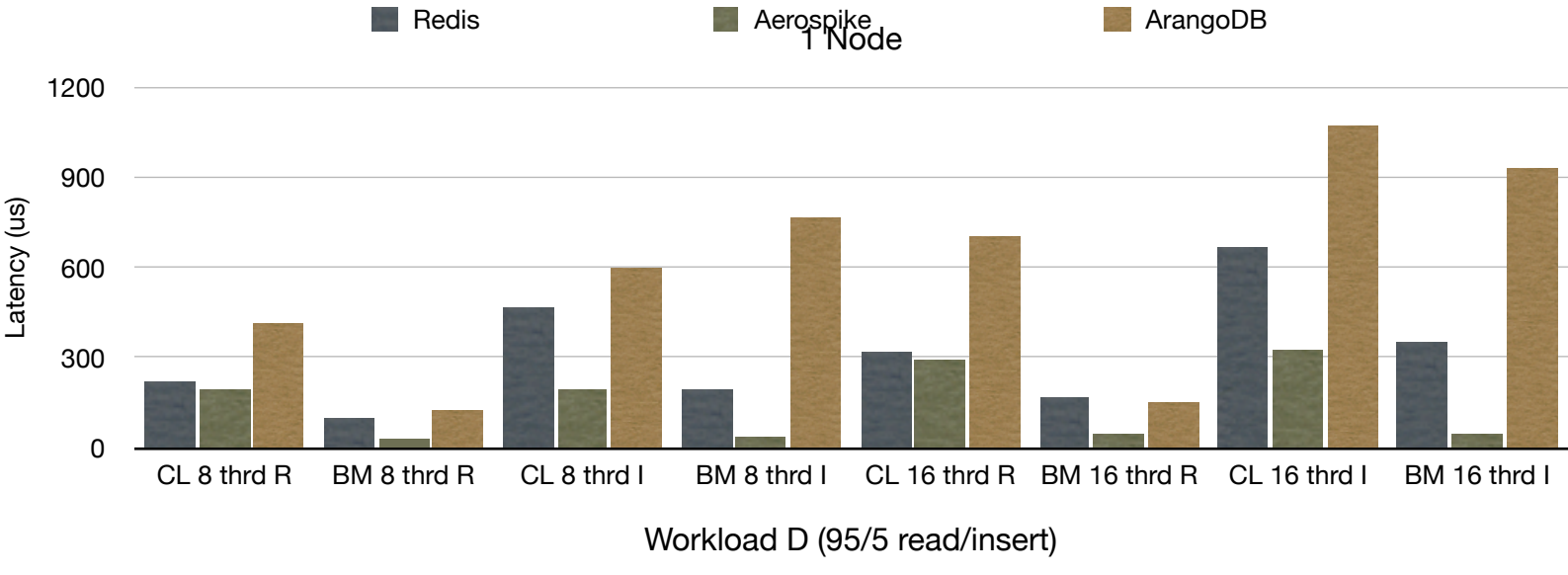
Διάγραμμα 96: Cloud vs Bare Metal Latency WorkloadC 1 Node 1-4 Threads



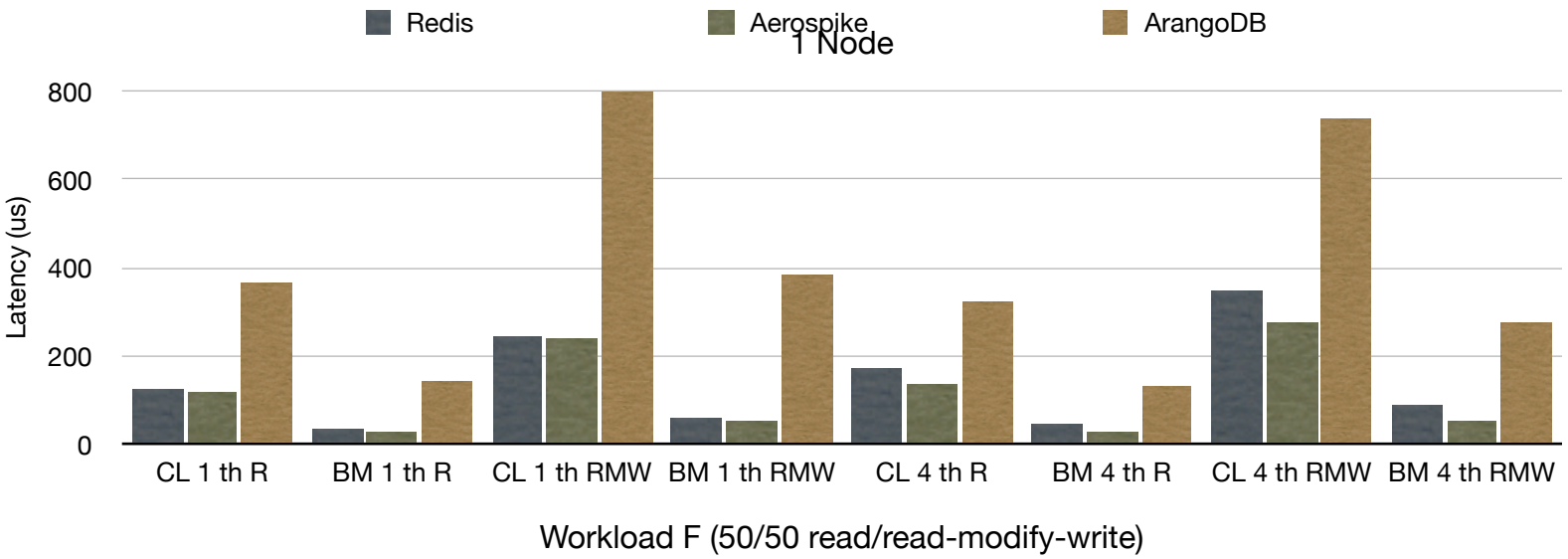
Διάγραμμα 97: Cloud vs Bare Metal Latency WorkloadC 1 Node 8-16 Threads



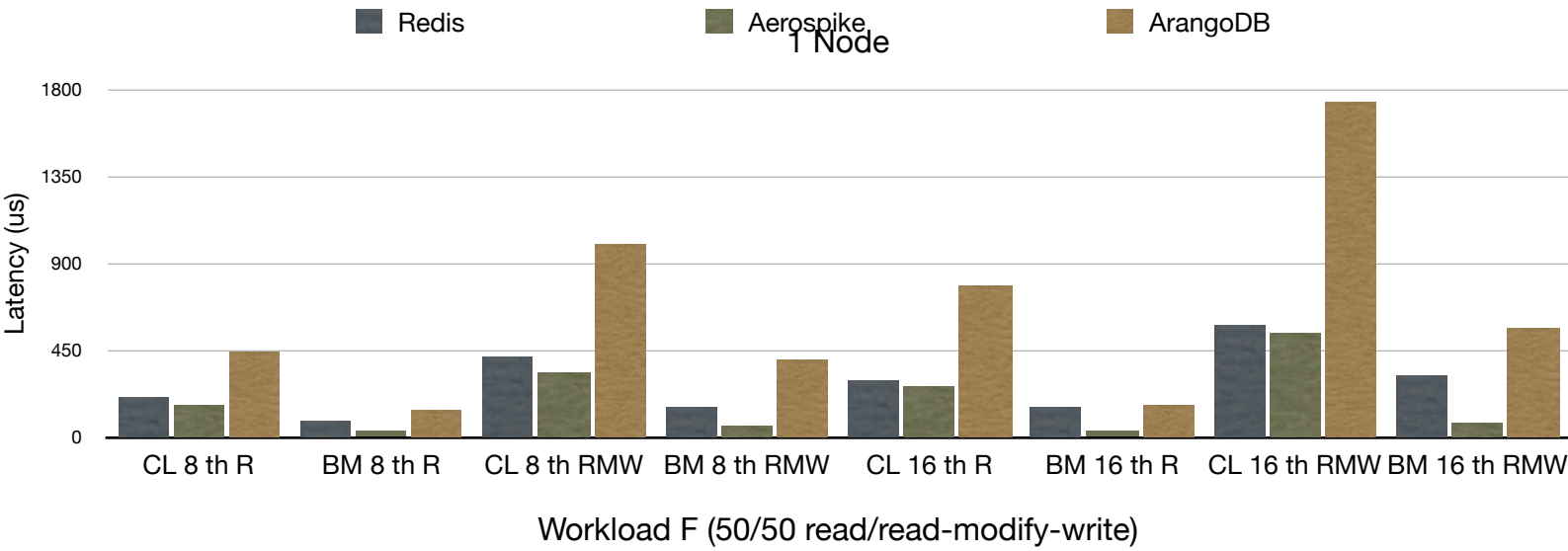
Διάγραμμα 98: Cloud vs Bare Metal Latency WorkloadD 1 Node 1-4 Threads



Διάγραμμα 99: Cloud vs Bare Metal Latency WorkloadD 1 Node 8-16 Threads



Διάγραμμα 100: Cloud vs Bare Metal Latency WorkloadF 1 Node 1-4 Threads



Διάγραμμα 101: Cloud vs Bare Metal Latency WorkloadF 1 Node 8-16 Threads

Από τα παραπάνω αποτελέσματα παρατηρούμε ότι η Redis και η Aerospike είναι πολύ παρόμοιες στην απόδοσή τους με την Aerospike να είναι πιο γρήγορη σε όλες τις περιπτώσεις. Αυτές οι δύο βάσεις είναι αναμενόμενο να είναι αρκετα πιο γρήγορες από την ArangoDB καθώς η ArangoDB αποθηκεύει τα δεδομένα στο δίσκο κάτι το οποίο είναι απενεργοποιημένο στις άλλες δύο βάσεις. Για την ακρίβεια η ArangoDB δεν υποστηρίζει λειτουργία μη αποθήκευσης στο δίσκο και η Aerospike υποστηρίζει αυτή τη λειτουργία μόνο σε περίπτωση που ο σκληρός δίσκος είναι τύπου SSD. Στη δική μας περίπτωση όλοι οι σκληροί δίσκοι είναι τύπου HDD. Επίσης παρατηρούμε ότι όλες οι βάσεις κλιμακώνονται γραμμικά όσο αυξάνουμε τον αριθμό των νημάτων εκτέλεσης του YCSB. Πιο συγκεκριμένα παρατηρούμε ότι στο φόρτο εργασίας A (workload A) η Redis φτάνει εώς και 53,562 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 279 us και 277 us για τις εντολές ανάγνωσης (read) και τις εντολές ανανέωσης (update). Η Aerospike φτάνει εώς και 55,157 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 263 us και 264 us για τις εντολές ανάγνωσης (read) και τις εντολές ανανέωσης (update). Η ArangoDB φτάνει εώς και 16,768 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 829 us και 1032 us για τις εντολές ανάγνωσης

(read) και τις εντολές ανανέωσης (update). Για το φόρτο εργασίας B (workload B) η Redis φτάνει εώς και 52,934 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 288 us και 284 us για τις εντολές ανάγνωσης (read) και τις εντολές ανανέωσης (update). Η Aerospike φτάνει εώς και 58,390 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 256 us και 265 us για τις εντολές ανάγνωσης (read) και τις εντολές ανανέωσης (update). Η ArangoDB φτάνει εώς και 21,147 εντολές/ δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 720 us και 972 us για τις εντολές ανάγνωσης (read) και τις εντολές ανανέωσης (update). Για το φόρτο εργασίας C (workload C) η Redis φτάνει εώς και 52,347 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 289 us για τις εντολές ανάγνωσης (read). Η Aerospike φτάνει εώς και 59,171 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 253 us για τις εντολές ανάγνωσης (read). Η ArangoDB φτάνει εώς και 23,434 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 661 us για τις εντολές ανάγνωσης (read). Για το φόρτο εργασίας D (workload D) η Redis φτάνει εώς και 47,654 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 303 us και 670 us για τις εντολές ανάγνωσης (read) και τις εντολές εισαγωγής (insert). Η Aerospike φτάνει εώς και 50,479 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 288 us και 322 us για τις εντολές ανάγνωσης (read) και τις εντολές εισαγωγής (insert). Η ArangoDB φτάνει εώς και 21,467 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 700 us και 1075 us για τις εντολές ανάγνωσης (read) και τις εντολές εισαγωγής (insert). Για το φόρτο εργασίας F (workload F) η Redis φτάνει εώς και 36,189 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 285 us , 573 us και 283 us για τις εντολές ανάγνωσης (read), τις εντολές ανάγνωσης-τροποποίησης-εγγραφής(read-modify-write) και τις εντολές ανανέωσης (update) . Η Aerospike φτάνει εώς και 37,412 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 263 us , 530 us και 264 us για τις εντολές ανάγνωσης (read), τις εντολές ανάγνωσης-τροποποίησης-εγγραφής(read-modify-write) και τις εντολές ανανέωσης (update) Η ArangoDB φτάνει εώς και 12,397 εντολές/δευτερόλεπτο με καθυστέρηση εκτέλεσης εντολών 779 us , 1744 us και 955 us για τις εντολές ανάγνωσης (read), τις εντολές ανάγνωσης-τροποποίησης-εγγραφής(read-modify-write) και τις εντολές ανανέωσης (update). Μια παρατήρηση που μπορούμε να κάνουμε είναι για την απόδοση όλων των βάσεων τόσο σε διεκπεραιωτική όσο και σε καθυστέρηση εκτέλεσης των εντολών. Είναι αρκετά χαμηλότερα (και ψηλότερα αντίστοιχα για την καθυστέρηση εκτέλεσης των εντολών) σε σχέση με τα αποτελέσματα που συλλέξαμε από το πραγματικό μηχάνημα. Αυτό συμβαίνει για δύο λόγους. Αρχικά τα μηχανήματα αυτά επικοινωνούν με το εργαλείο YCSB μέσω διαδικτύου καθώς βρίσκονται σε διαφορετικούς κόμβους. Έτσι σε περιόδους μεγάλου φόρτου του διαδικτύου τα αποτελέσματα είαπότιναι πιο χαμηλά απ'ότι περιμένουμε, όπως συμβαίνει και σε αυτή την περίπτωση. Επίσης ένα ακόμα εμπόδιο στα σωστά αυτά αποτελέσματα αποτελεί το ίδιο το cloud καθώς όλοι αυτοί οι κόμβοι είναι εγκατεστημένοι σε εικονικές μηχανές που σημαίνει ότι έχουν μικρότερη υπολογιστική ισχύ σε σχέση με ένα πραγματικό μηχάνημα με ακριβώς τα ίδια χαρακτηριστικά [34].

6. Επίλογος

6.1 Σύνοψη και συμπεράσματα

Πριν αναλύσουμε τα συμπεράσματα μας οφείλουμε να κάνουμε μια διευκρίνιση. Η διευκρίνιση αυτή αφορά την επίδοση της Redis. Η απόδοση που περιμέναμε θεωρητικά από την Redis ήταν να είναι η πιο γρηγορη βάση από τις υπόλοιπες καθώς είναι μια βάση η οποία έχει σχεδιαστεί καθαρά για επιδόσεις επειδή είναι μια βάση η οποία η κυρια λειτουργία είναι να αποθηκεύει τα δεδομένα της στη μνήμη Ram για γρήγορη διαχείριση δεδομένων. Επίσης όπως αναλύσαμε προηγουμένως η αναμενόμενη συμπεριφορά είναι ότι οι εντολές ανάγνωσης είναι πιο γρήγορες από τις εντολές εγγραφής. Η βιβλιογραφία της Redis [30] καλύπτει αυτό το γεγονός. Ο λόγος που η επίδοση της δεν είναι η αναμενόμενη είναι επειδή δε χρησιμοποιείται η τεχνική Pipelining [32]. Η τεχνική αυτή επιτρέπει καλύτερη εκμετάλλευση των πόρων του συστήματος καθώς εκτελούνται παράλληλα περισσότερες εντολές. Άλλο ένα πλεονέκτημα αυτής είναι η μείωση του ***RRT*** χρόνου, ο οποίος είναι ο χρόνος επικοινωνίας της βάσης με τον πελάτη καθώς δε χρειάζεται σε κάθε εντολή επικοινωνία με τον πελάτη παρά μόνο στην αρχή και στο τέλος ενός πακέτου εντολών. Αυτή η τεχνική σύμφωνα με τη βιβλιογραφία της Redis επιτρέπει εώς και δεκαπλάσιες επιδόσεις σε σύγκριση με τη μη χρησιμοποίηση αυτής της τεχνικής. Η τεχνική αυτή δε χρησιμοποιείται στην υλοποίηση του εργαλείου YCSB που χρησιμοποιούμε. Επειδή όμως θέλαμε να επιβεβαιώσουμε αυτή τη συμπεριφορά δοκιμάσαμε ένα διαφορετικό εργαλείο ανεπτυγμένο από τους δημιουργούς της Redis, το ***redis-benchmark*** . Το εργαλείο αυτό δε χρειάζεται εγκατάσταση καθώς υπάρχει στο ίδιο πακέτο εγκατάστασης της Redis. Προκειμένου να δημιουργήσουμε ένα αντίστοιχο φορτίο, χρησιμοποιούμε κλειδιά μεγέθους 1KB που χρησιμοποιεί και το YCSB, 16 νήματα εκτέλεσης για τον πελάτη και λειτουργία μονού κόμβου για τη Redis. Τα αποτελέσματα ήταν 993,049 ops/sec για ένα φορτίο σαν το C που έχει μόνο εντολές ανάγνωσης και 493,827ops/sec σε ένα φορτίο που έχει μόνο εντολές εγγραφής. Σε έναν μη pipelined client τα αντιστοιχα αποτελέσματα είναι 93,396 ops/sec και 89,887 ops/sec. Παρατηρούμε ότι σε σχέση με τα αποτελέσματα του YCSB όντως η απόδοση ενός pipelined πελάτη έχει σαν αποτέλεσμα απόδοση της Redis σχεδόν δεκαπλάσια στα Reads και σχεδόν 5πλάσια στα writes σε σύγκριση με ένα πελάτη που δε χρησιμοποιεί αυτή τη τεχνική.

Τα αποτελέσματα για τη Redis και την Aerospike είναι λογικό να είναι αρκετά καλύτερα από τα αποτελέσματα της ArangoDB καθώς οι δύο πρώτες βάσεις έχουν σχεδιαστεί προκειμένου να είναι πολύ γρήγορες. Επίσης τόσο η Redis όσο Aerospike έχουν σαν κύριο μοντέλο βάσης το μοντέλο κλειδιού-τιμής και απλά υποστηρίζουν μερικά δευτερεύοντα μοντέλα. Η ArangoDB έχει σαν κύρια μοντέλα βάσης ταυτόχρονα το μοντέλο κλειδιού-τιμής, το μοντέλο αποθήκευσης εγγράφων (document store), το μοντέλο βάσης γράφων (graph DBMS) και τέλος το μοντέλο μηχανής αναζήτησης (search engine). Οπότε οι δημιουργοί της Arango θυσίασαν ταχύτητα

προκειμένου να δημιουργήσουν μια βάση δεδομένων η οποία μπορεί να χρησιμοποιηθεί για πολλά είδη εφαρμογών. Επίσης στην ArangoDB η διαδικασία κλιμάκωσης αλλά και συρρίκνωσης ενός συμπλέγματος είναι μιά πολύ απλή διαδικασία καθώς απλά γίνεται προσθαφαίρεση κόμβων και το σύστημα αναλαμβάνει τις υπόλοιπες λειτουργίες όπως ο νέος θρυμματισμός δεδομένων κτλ. Ακόμα επειδή η ArangoDB έχει σχεδιαστεί ώστε να υποστηρίζει όλα αυτά τα μοντέλα απλοποιεί πολύ την επικοινωνία των μοντέλων αυτών καθώς σε διαφορετική περίπτωση κάθιε μοντέλο σημαίνει και διαφορετική βάση δεδομένων. Όποτε οι αλλαγές στην υλοποίηση, αλλά και οι ενημερώσεις απλοποιούνται δραστικά καθώς απλά πρέπει να γίνουν στο ίδιο σύστημα. Ένα ακόμη πλεονέκτημα της βάσης αυτής είναι ότι υποστηρίζει συναλλαγές (transactions). Αυτές ουσιαστικά είναι μικρά κομμάτια κώδικα που οφείλουν να ακολουθούν τις ιδιότητες ACID κάτι το οποίο είναι δυσεύρετο στις μη σχεσιακές βάσεις. Επίσης ένας άλλο παράγοντας που επηρεάζει την ταχύτητα της ArangoDB αρνητικά είναι ότι δε χρησιμοποιεί τη δική της μηχανή βάσης, αλλά χρησιμοποιεί σαν μηχανή την RocksDB που έχει φτιαχτεί από την εταιρία Facebook. Η Aerospike όπως είδαμε από τα πειραματικά αποτελέσματα είναι μια βάση δεδομένων η οποία έχει πολύ καλές επιδόσεις. Η δυνατότητα της να αποθηκεύει τα δεδομένα προσωρινά στη μνήμη έχει σαν αποτέλεσμα τη γρήγορη εκτέλεση εντολών. Επίσης η Aerospike έχει αναπτύξει δικούς της οδηγούς (drivers) προκειμένου να είναι όσο πιο αποδοτική γίνεται όταν αποθηκεύει δεδομένα σε σκληρούς δίσκους SSD. Η λειτουργία της μόνιμης αποθήκευσης όμως δε μπορεί να χρησιμοποιηθεί σε δίσκους τύπου HDD. Δυστυχώς στη δική μας περίπτωση έχουμε δίσκο HDD οπότε δε μπορούμε να εκτελέσουμε κάποιο τέτοιο τεστ οπως θα εκτελέσουμε με τη Redis στη συνέχεια. Επίσης άλλο ένα πλεονέκτημα της Aerospike είναι ότι η παραμετροποίηση της είναι πολύ εύκολη, όπως επίσης και το στήσιμο ενός συμπλέγματος καθώς το μόνο που πρέπει να κάνει ένας χρήστης είναι να προσθαφαιρέσει κόμβους. Τέλος, η Redis είναι μια βάση η οποία επιτυγχάνει πολύ μεγάλες επιδόσεις τόσο χρησιμοποιώντας pipelining όσο και χωρίς τη χρησιμοποίηση αυτού. Αυτό βέβαια έχει σαν κόστος τη σχετικά δύσκολη παραμετροποίηση της. Επίσης δύσκολο είναι και στο στήσιμο ενός συμπλέγματος καθώς δεν αρκεί απλά να προσθαφαιρέσεις κόμβους όπως γίνεται στις άλλες δύο βάσεις. Τα αρχεία που κρατούν την κατάσταση τον κόμβων πρέπει να διαγραφούν και να ξαναστηθεί από την αρχή το σύμπλεγμα με τις αντίστοιχες εντολές. Επίσης η Redis επειδή δεν υποστηρίζει ακριβώς μόνιμη εγγραφή στο δίσκο έγκειται σε κινδύνους απώλειας δεδομένων. Επίσης ένα άλλο μειονέκτημα της Redis είναι ότι επειδή είναι βάση η οποία στηρίζεται στη μνήμη ram, χρειάζεται αρκετά μεγάλη ram κάτι το οποίο ενδέχεται να είναι κοστοβόρο. Κάτι άλλο που παρατηρούμε για την Redis είναι ότι οι λειτουργίες ενημέρωσης είναι ταχύτερες από τις λειτουργίες εισαγωγής, σε αντίθεση με τις άλλες δύο βάσεις.

Επίσης παρατηρήσαμε ότι η Redis είναι αναμενόμενα ταχύτερη όταν χρησιμοποιείται για να αποθηκεύει τα δεδομένα στη μνήμη του υπολογιστή παρά όταν χρησιμοποιείται για να αποθηκεύει τα δεδομένα στο σκλήρό στο δίσκο. Βέβαια οι διαφορές δεν είναι μεγάλες που αποδυκνύει πόσο αποδοτική μπορεί να γίνει η Redis.

Ακόμα αναδείξαμε τη διαφορά που μπορεί να έχει το περιβάλλον στο οποίο εκτελούνται οι βάσεις και πόσο μπορεί να επηρεάσει την απόδοση αυτών των συστημάτων. Συγκεκριμένα όταν

εκτελέσαμε τους φόρτους εργασίας στο εικονικό περιβάλλον παρατηρήσαμε αρκετά μειωμένη διεκπεραιωτική ικανότητα και αυξημένους χρόνους καθυστέρησης εκτέλεσης εντολών σε σχέση με τους φόρτους εργασίας που εκτελέσαμε στο πραγματικό περιβάλλον.

Συνοψίζοντας αποδείξαμε πως κάθε βάση δεδομένων έχει τα πλεονεκτήματά και τα μειονεκτήματά της. Η επιλογή της βάσης δεδομένων πρέπει να γίνεται με τρόπο με τον οποίο να καλύπτονται οι ανάγκες για ένα συγκεκριμένο πρόβλημα που θέλουμε να λύσουμε. Συγκεκριμένα η Redis, ως η ταχύτερη βάση δεδομένων από τις υπόλοιπες δύο, ενδείκνυται για τη χρησιμοποίηση της ως προσωρινή μνήμη (cache) σε μια εφαρμογή [34], ώστε να μπορεί να γίνεται ταχύτατη ανάγνωση και εγγραφή των πιο σημαντικών δεδομένων. Επίσης η ατομικές εντολές που παρέχει διασφαλίζουν πάντα τη σωστή τιμή των δεδομένων και οι πολλοί τύποι δεδομένων που υποστηρίζει την κάνει πολλή ελαστική, ώστε να μπορεί να προσαρμοστεί σε κάθε περίπτωση. Επίσης όντας μια κατανεμημένη βάση δεδομένων μπορεί να χρησιμοποιηθεί πολύ αποδοτικά προκειμένου να μπορεί να δέχεται περισσότερα αιτήματα και να μπορεί να παρέχει περισσότερη ασφάλεια των δεδομενων σε περίπτωση που κάποιος κόμβος αποτύχει. Η Aerospike επίσης είναι μια γρήγορη βάση δεδομένων. Μπορεί και αυτή να χρησιμοποιηθεί ως προσωρινή μνήμη[35]. Σε σχέση με την Redis δε χρειάζεται τόσους κόμβους καθώς η πολυνηματική της αρχιτεκτονική της επιτρέπει να απαντά σε πολλαπλά αιτήματα των πελατών. Επίσης η διάταξη συμπλέγματός της, είναι πολύ απλή σε σχέση με την Redis και παρέχει πολλές αυτόματες διαδικασίες οι οποίες διευκολύνουν πολύ τον χρήστη. Επίσης μπορεί να χρησιμοποιηθεί και για να υποστηρίξει εξαιρετικά μηχανές συστάσεων καθώς η Aerospike είναι μια βάση δεδομένων που μπορεί να χειριστεί πολύ καλά μεγάλο όγκο δεδομένων λόγω των drivers που έχει αναπτύξει για τη ταχύτατη ανάκτηση και επεξεργασία δεδομένων. Τέλος η ArangoDB είναι μία βάση με αξιόλογες επιδόσεις. Τα πολλά κύρια μοντέλα της ArangoDB της επιτρέπουν να χρησιμοποιηθεί σε πληθώρα εφαρμογών[36]. Συγκεκριμένα το μοντέλο αυτό κάνει τη συντήρηση και την περαιτέρω ανάπτυξη μιας εφαρμογής επειδή η ίδια βάση μπορεί να χρησιμοποιηθεί αποδοτικά για πολλές χρήσεις. Αυτό έχει σαν αποτέλεσμα μειωμένο κόστος και μειωμένη πολυπλοκότητα σε ένα σύστημα. Επιπρόσθετα η λειτουργία συμπλέγματος της είναι αρκετά απλή που σημαίνει ότι μπορεί και κλιμακώνεται πολύ εύκολα.

6.2 Μελλοντικές επεκτάσεις

Υπάρχουν αρκετοί τρόποι με τους οποίους θα μπορούσε να επεκταθεί η εργασία. Αρχικά θα μπορούσαμε να διαφοροποιήσουμε τον αριθμό τον αριθμό τον κόμβων. Ακόμα θα μπορούσαμε να αυξήσουμε τον αριθμό των παράλληλων νημάτων-πελατών του YCSB που κάνουν αιτήματα ταυτόχρονα στις βάσεις. Επιπρόσθετα ένα άλλο κομμάτι στο οποίοι θα μπορούσε να επεκταθεί η συγκεκριμένη εργασία είναι να προστεθούν/διαλεχθούν και άλλες βάσεις δεδομένων τύπου κλειδιού τιμής. Μια ακόμη διάσταση που μπορεί να δωθεί σε τέτοιες εργασίες είναι η σύγκριση μη σχεσιακών βάσεων δεδομένων τύπου κλειδιού τιμής με άλλους τύπους μη σχεσιακών βάσεων δεδομένων ή και ακόμα και να γίνει σύγκριση με σχεσιακές βάσεις δεδομένων.

7. Βιβλιογραφία

[1] NoSQL Wikipedia Article, https://en.wikipedia.org/wiki/NoSQL

[2] RDBMS Wikipedia Article, https://searchdatamanagement.techtarget.com/definition/RDBMS-relational-database-management-system

[3] Cloud Computing Wikipedia Article, https://en.wikipedia.org/wiki/Cloud_computing

[4] What is a Key-Value store, https://www.aerospike.com/what-is-a-key-value-store/

[5] DB-Engines. DB-Engines Ranking, https://db-engines.com/en/ranking/key-value+store

[6] S. Sanfilippo, “Redis,” [Online]. Available: http://redis.io/

[7] Aerospike Official Site, https://www.aerospike.com/

[8] ArangoDB Official Site, https://www.arangodb.com/

[9] What are Virtual Machines, https://www.vmware.com/topics/glossary/content/virtual-machine

[10] Cooper, Brian F., et al. "Benchmarking cloud serving systems with YCSB." Proceedings of the 1st ACM symposium on Cloud computing. 2010.Cooper, Brian F., et al. "Benchmarking cloud serving systems with YCSB." Proceedings of the 1st ACM symposium on Cloud computing. 2010.

[11] Rabl, Tilmann & Sadoghi, Mohammad & Jacobsen, Hans-Arno & Gómez-Villamor, Sergio & Muntés-Mulero, Victor & Mankowskii, Serge. (2012). Solving Big Data Challenges for Enterprise Application Performance Management. Proc VLDB Endowment. 5. 10.14778/2367502.2367512.

[12] Project Voldemort Official Site, https://www.project-voldemort.com/voldemort/

[13] MySQL Cluster, https://www.mysql.com/products/cluster/

[14] VoltDB Official Site, https://www.voltdb.com/

[15] HBase Official Site, https://hbase.apache.org/

[16] Apache Cassandra Official Site, https://cassandra.apache.org/

[17] Ma, Wenlong & Zhu, Yuqing & Li, Cheng & Guo, Mengying & Bao, Yungang. (2019). BiloKey : A Scalable Bi-Index Locality-Aware In-Memory Key-Value Store. IEEE Transactions on Parallel and Distributed Systems. PP. 1-1. 10.1109/TPDS.2019.2891599.

[18] Memcached. A distributed memory object caching system, [Online]. Available: http://memcached.org/

[19] Anthonyaje.Github.Io, 2020, https://anthonyaje.github.io/file/An_empirical_evaluation_of_Memcached_Redis_and_Aerospike_kvstore_Anthony_Eswar.pdf.

[20] Research, Brian F. Cooper Yahoo!, et al. “PNUTS: Yahoo!'s Hosted Data Serving Platform.” Proceedings of the VLDB Endowment, 1 Aug. 2008, dl.acm.org/doi/10.14778/1454159.1454167.

[21] Lopes, Dayvson, D. S. (2015). Redis essentials: Harness the power of Redis to integrate and manage your projects efficiently. Packt Publishing.

[22] Carlson, J. L. (2013). Redis in action. Shelter Island: Manning.

[23] Redis Core Implementation, http://key-value-stories.blogspot.com/2015/01/redis-core-implementation.html

[24] Raft Consensus Algorithm, https://raft.github.io/

[25] Aerospike Wikipedia Article, https://en.wikipedia.org/wiki/Aerospike_(database)

[26] What are Sprigs, https://discuss.aerospike.com/t/faq-what-are-sprigs/4936

[27] ArangoDB Wikipedia Article, https://en.wikipedia.org/wiki/ArangoDB

[28] RocksDB Official Site, https://rocksdb.org/

[29] CAP Theorem, https://en.wikipedia.org/wiki/CAP_theorem

[30] Redis Pipelining, https://redis.io/topics/pipelining

[31] RocksDB Write Buffet Manager, https://github.com/facebook/rocksdb/wiki/Write-Buffer-Manager

[32] Pipelining Technique, https://en.wikipedia.org/wiki/Instruction_pipelining

[33] Fork Linux Commands, https://linux.die.net/man/2/fork

[34] Bare Metal VS VMs https://www.ibm.com/blogs/cloud-computing/2014/06/17/bare-metal-vs-virtual-machine-cloud-option-right/

[35] 15 Reasons to use Redis as a cache https://redislabs.com/wp-content/uploads/2016/03/15-Reasons-Caching-is-best-with-Redis-RedisLabs-1.pdf

[36] When to use Aerospike https://www.aerospike.com/solutions/technology/use-cases/

[37] When to use ArangoDB https://www.arangodb.com/why-arangodb/native-multi-model-database-advantages/