ΕΞΑΜΗΝΙΑΙΑ ΕΡΓΑΣΙΑ ΣΤΑ ΚΑΤΑΝΕΜΗΜΕΝΑ ΣΥΣΤΗΜΑΤΑ
ΑΝΑΦΟΡΑ
Ακ. έτος 2020-2021, 9ο Εξάμηνο, Σχολή ΗΜΜΥ

Παναγιώτης Ταμπάκης: 03116749
Νικόλαος Σπύρου: 03116201

Παρουσίαση & Σχολιασμός Πειραμάτων

Πείραμα 1: *Write Throughput*

Για την καταγραφή του write throughput του συστήματος (DHT 10 κόμβων) εισάγαμε σειριακά τα κλειδιά που περιέχονται στο αρχείο **inserts.txt,** επιλέγοντας κάθε φορά τυχαία έναν από τους 10 κόμβους για την εξυπηρέτηση του εκάστοτε αιτήματος. Η μέτρηση του write throughput πραγματοποιήθηκε για τις περιπτώσεις συνέπειας ***linearizability***(*chain replication*) και ***eventual consistency,*** καθώς και για διαφορετικές τιμές του αριθμού αντιγράφων στο σύστημα: **k=1, k=3, k=5.**

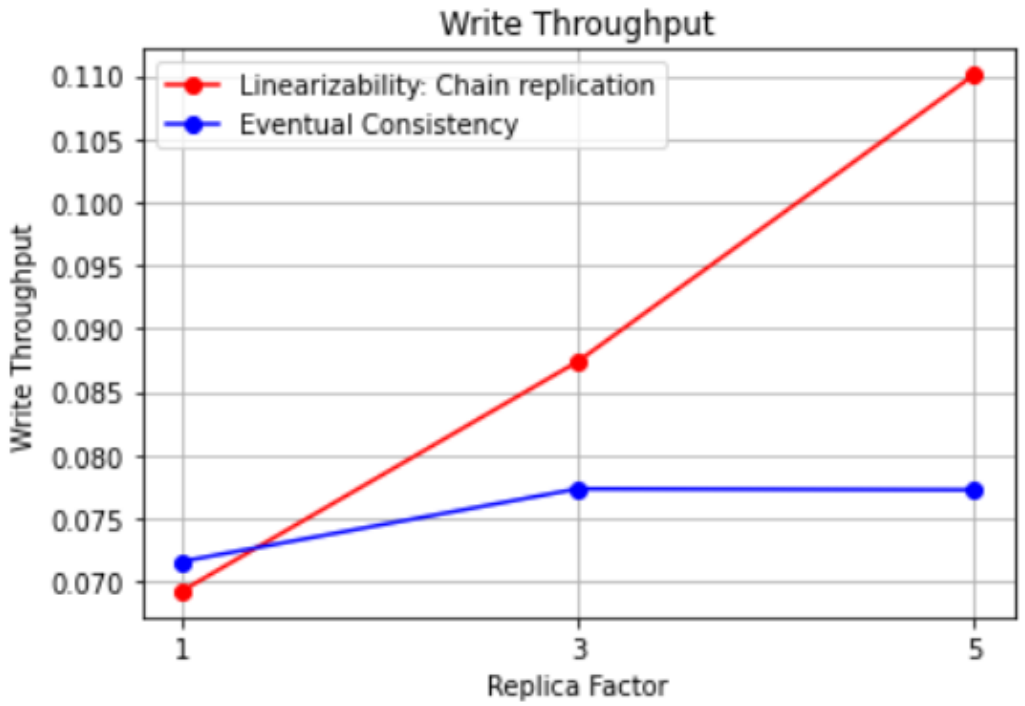
Παραθέτουμε τις τιμές που προέκυψαν από τα παραπάνω 6 setups, καθώς και το αντίστοιχο διάγραμμα.

Linearizability: Chain Replication

k = 1	0.06924084138870239
k = 3	0.08739981937408448
k = 5	0.11005661392211914

Eventual Consistency

k = 1	0.07159079360961915
k = 3	0.07733038806915284
k = 5	0.0772453727722168



Στην περίπτωση του chain replication παρατηρούμε ότι υπάρχει μία γραμμική εξάρτηση του write throughput από το replica factor. Με την αύξηση του αριθμού των αντιγράφων διακρίνεται και ανάλογη συμπεριφορά στο write throughput(αύξηση). Εξ' ορισμού στο chain replication η διαδικασία write επιστρέφει το αποτέλεσμα της μετά και την εισαγωγή αντιγράφου στο τελευταίο κόμβο της "αλυσίδας" που δημιουργείται, γεγονός που καθιστά το συγκεκριμένο πρωτόκολλο πιο χρονοβόρο στα writes σε σχέση με άλλα, προσφέροντας όμως μία ισχυρή μορφή συνέπειας. Συνεπώς η αύξηση που παρατηρείται στο throughput, καθώς μεγαλώνει ο αριθμός των αντιγράφων είναι αναμενόμενη.

Στον αντίποδα, για την περίπτωση του eventual consistency το αποτέλεσμα επιστρέφεται μόλις το κλειδί εγγραφεί στον primary κόμβο της αλυσίδας(λιγότερο αυστηρή μορφή συνέπειας). Πιο συγκεκριμένα, λαμβάνουμε αρκετά κοντινές τιμές write throughput για τις διαφορετικές τιμές του k στο eventual consistency, συμπεριφορά αναμενόμενη βάσει του πρωτοκόλλου.

Πείραμα 2: *Read Throughput*

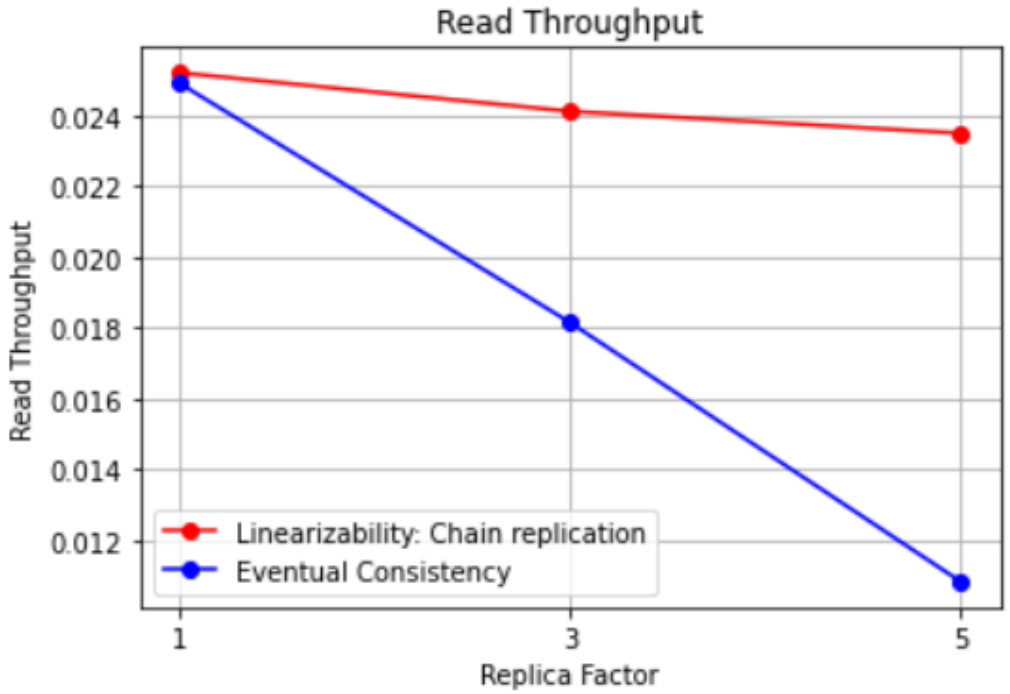
Στο δεύτερο πείραμα κάνουμε χρήση του αρχείου query.txt και καταγράψαμε το read throughput. Το συγκεκριμένο αρχείο περιέχει αιτήματα read τα οποία ξεκινούν από τυχαίο κόμβο του DHT κάθε φορά. Για τα παραπάνω 6 setups παρουσιάζουμε τα αποτελέσματα που προέκυψαν.

Linearizability: Chain Replication

k = 1	0.02522079038619995
k = 3	0.024129820823669434
k = 5	0.023511315345764162

Eventual Consistency

k = 1	0.024932424068450927
k = 3	0.018161864280700685
k = 5	0.010832189083099365



Στην περίπτωση του chain replication η επεξεργασία των αιτημάτων για read γίνεται από τον server που αποτελεί την ουρά της αλυσίδας των αντιγράφων και είναι αυτός που απαντάει πίσω στον client που έθεσε το query. Επομένως, η ουρά περιέχει πάντα την πιο πρόσφατη τιμή που γράφτηκε στο σύστημα για το δεδομένο κλειδί και θα είναι αυτή η τιμή που θα επιστρέφεται σε κάθε αίτημα read στον client. Έτσι, αποφεύγεται το διάβασμα stale δεδομένων. Από το διάγραμμα, παρατηρούμε χαμηλότερες τιμές σε σχέση με το write throughput, αρκετά παραπλήσιες μεταξύ τους. Με την αύξηση του replica factor βλέπουμε ότι υπάρχει κάποια μικρή μείωση στο read throughput, καθώς η ύπαρξη περισσότερων αντιγράφων στο σύστημα αυξάνει την πιθανότητα να σταλεί αίτημα query σε κάποιον κόμβο που έχει ήδη το αντίγραφο και έτσι να βρεθεί πιο γρήγορα ο server ουρά που έχει την πιό πρόσφατη τιμή. Επίσης, επειδή στο chain replication όλα τα read requests πρέπει να ικανοποιηθούν από την ουρά της αλυσίδας δεν μπορεί να διαμορφωθεί κάποια εμφανή γραμμική εξάρτηση σε σχέση με το μέγεθος της αλυσίδας (replica factor). Αυτο επιβεβαιώνεται και από το διάγραμμα.

Όσον αφορά το eventual consistency τα reads requests επιστρέφουν την τιμή του πρώτου server που θα βρεθεί και θα έχει το κλειδί, χωρίς να λαμβάνεται υπόψην να επιστρέφεται πάντα η πιό πρόσφατη τιμή. Συνεπώς, θα είναι συχνό το φαινόμενο κάποιος να client να λαμβάνει stale τιμες. Από το διάγραμμα, παρατηρείται μία αρκετά εμφανής γραμμική μείωση του read throughput κατά την αύξηση του replication factor. Μπορούμε να πούμε ότι αυτή η συμπεριφορά είναι λογική, καθώς το read επιστρέφει την τιμή του πρώτου κόμβου που θα έχει το κλειδί, επομένως όσο περισσότερα είναι τα αντίγραφα για κάποιο κλειδί στο σύστημα, τόσο μεγαλύτερη είναι η πιθανότητα το αίτημα να σταλεί σε έναν server που θα έχει το κλειδί και να επιστρέψει αμέσως.

Πείραμα 3: *Freshness αποτελεσμάτων query στις 2 περιπτώσεις consistency*

Στο συγκεκριμένο πείραμα έγινε χρήση του αρχείου requests.txt, το οποίο περιέχει αιτήματα insert και query, τα οποία κάθε φορά ανατίθενται σε έναν τυχαίο κόμβο. Το replication factor ήταν ίσο με 3. Έγινε καταγραφή σε αρχείο των απαντήσεων των queries για τις 2 μορφές συνέπειας.

Ενδεικτικά σας παραθέτουμε ένα κομμάτι από το αρχείο requests.txt :

```
 96  insert, Like a Rolling Stone, 520
 97  query, Hey Jude
 98  query, Like a Rolling Stone
 99  query, Like a Rolling Stone
100  query, Respect
101  insert, What's Going On, 521
102  query, Like a Rolling Stone
103  query, Hey Jude
104  query, Respect
105  query, Hey Jude
106  insert, Respect, 522
107  query, Respect
108  query, What's Going On
109  query, Hey Jude
110  query, What's Going On
111  insert, Hey Jude, 523
112  query, Like a Rolling Stone
```

Στην γραμμή 96 βλέπουμε ότι γίνεται insert με key : Like a Rolling Stone και value : 520 , ενώ στην γραμμή 112 γίνεται αίτημα query για το συγκεκριμένο κλειδί.

Στην περίπτωση του linearizability(chain replication) παρατηρούμε ότι λαμβάνουμε για κάθε query τις πιό fresh τιμές και δεν διαβάζονται stale δεδομένα. Όπως αναφέρθηκε και παραπάνω η συμπεριφορά αυτή είναι αναμενόμενη, καθώς τα αιτήματα query ικανοποιούνται από τον server ουρά της αλυσίδας.

```
Request:112 returned:"520:1"^M
```

Βλέπουμε ότι επιστρέφεται η πιο fresh τιμή για το κλειδί αυτό.

Στην περίπτωση του eventual consistency βλέπουμε ότι στο log αρχείο υπάρχουν και stale τιμές στις απαντήσεις των αιτημάτων query. Αυτό είναι κάτι που περιμέναμε,

καθώς όπως έχει αναφερθεί στον eventual consistency τα αιτήματα read ικανοποιούνται από τον πρώτο κόμβο που έχει διαθέσιμο αντίγραφο του κλειδιού.

Από το log αρχείο φαίνεται ότι επιστρέφεται μια παλαιότερη τιμή για το συγκεκριμένο κλειδί (504), η οποία έχει εισαχθεί νωρίτερα στο σύστημα.

Επομένως, προφανώς η μορφή συνέπειας **linearizability** προσφέρει τις πιό πρόσφατες (fresh) τιμές, όπως και αναμέναμε.