ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Κατανεμημένα Συστήματα

Καπερώνη Φρειδερίκη	03116685
Πιτσόλης Γιώργος	03116441
Τριανταφυλλόπουλος Ηλίας	03116028

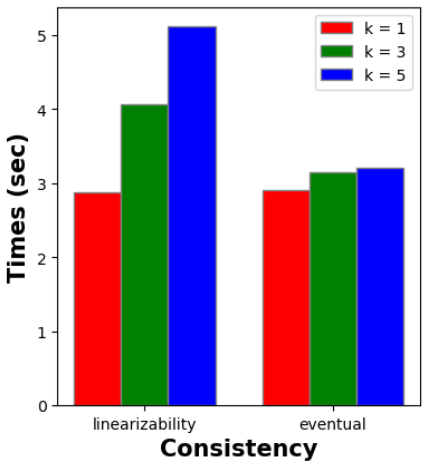
Εξαμηνιαία Εργασία κατασκευής ToyChord

Πειράματα

Δημιουργείται ένα DHT με 10 κόμβους, 2 ανά VM με port από 5000 έως 5009.

- Στο **πρώτο μέρος** της εργασίας ζητείται να υπολογιστούν οι χρόνοι που προκύπτουν μετά την περάτωση 6 διαφορετικών πειραμάτων με διαφορετικούς συντελεστές. Οι χρόνοι αυτοί μετρούνται έχοντας ως αρχείο εισόδου το insert.txt με σκοπό την καταγραφή του write throughput του συστήματος για linearizability και eventual consistency για τιμές replications k = 1, 3 και 5. Αναλυτικά τα αποτελέσματα είναι τα ακόλουθα:

total time for insert with k=1 - linearizability:	2.881353378295898
total time for insert with k=1 - eventual:	2.911711931228637
total time for insert with k=3 - linearizability:	4.059234142303467
total time for insert with k=3 - eventual:	3.149774551391601
total time for insert with k=5 - linearizability:	5.124131679534912
total time for insert with k=5 - eventual:	3.214448690414429



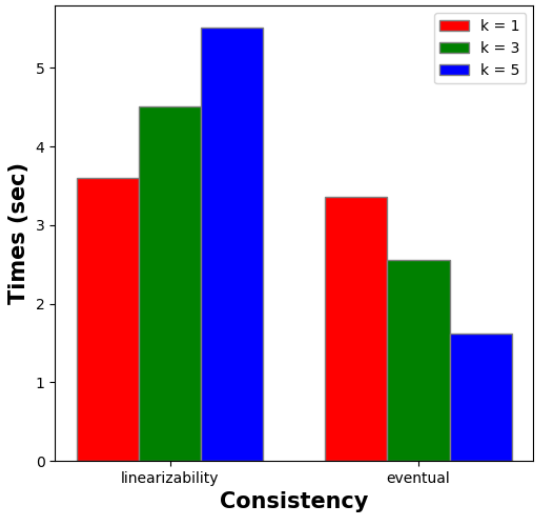
Όπως ήταν αναμενόμενο, στην περίπτωση που έχουμε μόνο ένα αντίγραφο και οι δύο τρόποι consistency έχουν τον ίδιο χρόνο throughput καθώς οι δύο περιπτώσεις μπορεί να θεωρηθούν ότι ταυτίζονται. **Απεναντίας, στις επόμενες δύο περιπτώσεις με τρία και πέντε αντίγραφα παρατηρείται ότι υπάρχει απόκλιση ενός και δύο δευτερολέπτων αντίστοιχα.** Αναμενόμενο συμπέρασμα καθώς με την μέθοδο του linearizability το πρόγραμμα περιμένει να γίνει η εισαγωγή όλων των αντιγράφων στον δακτύλιο για να συνεχίσει ενώ με το eventual αρκεί να δημιουργηθεί μόνο το πρώτο αντίγραφο προκειμένου να συνεχίσει στην επόμενη εντολή. Υπάρχει δηλαδή κάποια μορφή επικάλυψης στις εντολές που όμως δεν παρουσιάζουν κάποιο πρόβλημα στην συγκεκριμένη περίπτωση που έχουμε μόνο inserts στον δακτύλιο μας. Επίσης ο χρόνος του linearizability αυξάνεται όσο μεγαλώνει και ο αριθμός των αντιγράφων, γεγονός που αποτελεί συνέπεια του αλγορίθμου αφού για να μπορέσει να επιστραφεί μήνυμα επιτυχούς εκτέλεσης του insert απαιτείται να γίνει η εισαγωγή και των k αντιγράφων που ζητούνται. Μικρότερη αύξηση παρουσιάζεται και στην περίπτωση του eventual consistency. Αυτή η μικρή αύξηση οφείλεται και εδώ στην αύξηση του k, αφού μπορεί να επιστρέφει μετά την εισαγωγή του πρώτου αντιγράφου, αλλά το σύστημα συνεχίζει να βάζει τα αντίγραφα και έτσι καταναλώνεται περισσότερος χρόνος.

- Στο **δεύτερο μέρος** ζητείται για τις 6 παραπάνω περιπτώσεις μετά το insert να εκτελεστούν και τα queries που βρίσκονται στο αρχείο query.txt με καταγραφή του read throughput. Τα αποτελέσματα είναι τα ακόλουθα:

total time for queries with k=1 - linearizability: 3.594913482666016
total time for queries with k=1 - eventual: 3.353199720382691

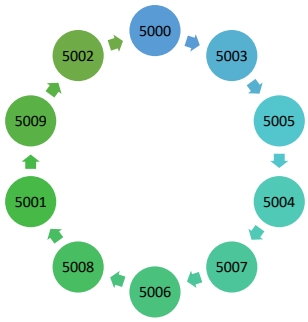
total time for queries with k=3 - linearizability: 4.507219076156616
total time for queries with k=3 - eventual: 2.553696155548096

total time for queries with k=5 - linearizability: 5.512279510498047
total time for queries with k=5 - eventual: 1.624677658081055



Παρόμοια αποτελέσματα παρουσιάζονται και στο δεύτερο κομμάτι της άσκησης όπου εκτός από την εκτέλεση του insert γίνεται και μία ακολουθία από queries για τους δύο τρόπους συνέπειας που προαναφέρθηκαν. **Στην περίπτωση του linearizability αναζητούνται όλα τα αντίγραφα που ικανοποιούν το συγκεκριμένο query** απαιτώντας κάθε φορά που αυξάνεται το k περισσότερο χρόνο καθώς αυξάνονται τα αντίγραφα που καλείται να εντοπίσει. Από την άλλη μεριά **στο eventual consistency απαιτείται ο εντοπισμός μόνο ενός εκ των αντιγράφων** για να επιστραφεί η απαιτούμενη τιμή. Απόρροια αυτού αποτελεί το γεγονός πως ο χρόνος μειώνεται όσο αυξάνεται ο αριθμός των αντιγράφων αφού υπάρχουν περισσότερα αντικείμενα που μπορεί να εντοπίσει. **Έτσι το χάσμα μεταξύ linearizability και eventual consistency αυξάνεται με την παράλληλη άνοδο του k.**

- Στο **τρίτο μέρος** για την μεμονωμένη περίπτωση των τριών αντιγράφων ζητείται η παρατήρηση των αποτελεσμάτων σχετικά με το ποια μέθοδος συνέπειας φέρει τα πιο ορθά αποτελέσματα, τα αποτελέσματα δηλαδή με τις πιο πρόσφατες τιμές για τα αντίστοιχα κλειδιά. Αφότου εκτελέστηκαν και οι δύο παραπάνω μέθοδοι παρατηρήθηκε πως **με linearizability consistency δεν εμφανίζονταν λάθη αναφορικά με τις τιμές που αποδίδονταν στα queries.** Κάτι τέτοιο αποτελεί λογικό συμπέρασμα της παρούσας μεθόδου καθώς κάθε φορά που γίνεται κάποιο insert το οποίο καλείται να ανανεώσει παλιές τιμές αναζητούνται και τροποποιούνται όλα τα αντίγραφα που οδηγούν σε αυτό. Έπειτα στέλνεται μήνυμα ολοκλήρωσης της διαδικασίας εισαγωγής και το πρόγραμμα συνεχίζει στην επόμενη εντολή που μπορεί να αποτελεί ένα ενδεχόμενο query στην πρόσφατα τροποποιημένη τιμή. Το αποτέλεσμα με αυτό τον τρόπο διασφαλίζεται ως ορθό αφού σε καμία περίπτωση δεν παρατηρείται κάποιου είδους επικάλυψη.
  Αντιθέτως, με eventual consistency παρατηρήθηκε όπως φαίνεται και παρακάτω κάποιου είδους τροποποίηση των αποτελεσμάτων σε συγκεκριμένες περιπτώσεις.



Σε μια συγκεκριμένη περίπτωση ζητείται να γίνει insert μια καινούργια τιμή στην περίπτωση του <key, value> = <Hey Jude, 597> με το <Hey Jude, 598> και αμέσως μετά ένα query πάνω στο value που έχει το key = Hey Jude.

query, Hey Jude

insert, Hey Jude, 598

query, Hey Jude

Μέσω του master έχουμε τα μηνύματα που αναφέρουν που θα γίνουν τα δύο queries και το insert.

```
query will start from port =  5009
query completed
insert will start from port =  5006
insert completed
query will start from port =  5009
query completed
```

Το insert για την τιμή 598 ξεκινάει από το κόμβο 5006, ο οποίος διαδοχικά το στέλνει στον 5008, που είναι υπεύθυνος για το συγκεκριμένο ζεύγος key-value. Οπότε μετά την αλλαγή του πρώτου αντιγράφου του ζευγαριού από τον 5008 το πρόγραμμα θεωρεί ότι μπορεί να συνεχίσει χωρίς να έχει προλάβει ακόμη να κάνει την τροποποίηση των αντιγράφων που βρίσκονται στους κόμβους 5001 και 5009 αφήνοντας τους για κάποιο μικρό διάστημα με την προηγούμενη τιμή την 597. Έπειτα ζητείται να γίνει το query που παρατηρούμε να τυχαίνει να ξεκινάει από τον κόμβο 5009 που περιέχει ένα παλιό αντίγραφο του συγκεκριμένου key.

```
I am port =  5009 with id =  b0129a178fcde3256ce78dc8a12de759485c1fc2 and the key =   Hey Jude has been found by node with id =  b0
t =  5009 and it is =   597
I am port =  5009 with id =  b0129a178fcde3256ce78dc8a12de759485c1fc2 and the key =   Hey Jude has been found by node with id =  b0
t =  5009 and it is =   597
I am port =  5009 with id =  b0129a178fcde3256ce78dc8a12de759485c1fc2 and I have inserted the key =   Hey Jude with value =   598
```

Από το στιγμιότυπο αυτό παρατηρούμε ότι τυπώνεται και για τα δύο παραπάνω queries η ίδια τιμή 597 αντί για 597 στο πρώτο που είναι πριν το insert και 598 στο δεύτερο που το ακολουθεί. Όπως φαίνεται, το δεύτερο query ολοκληρώνεται πριν από το insert και έτσι επιστρέφεται η παλιά τιμή. Γενικεύοντας το συμπέρασμα μας, η περίπτωση το πρόγραμμα να αποτύχει να επιστρέψει την σωστή τιμή του ζευγαριού είναι όταν το query ξεκινάει από κόμβο που περιέχει αντίγραφο που δεν έχει προλάβει να ενημερωθεί. Στην συγκεκριμένη περίπτωση θα εμφανιζόταν στο αποτέλεσμα πάλι λανθασμένη τιμή αν το query ξεκινούσε από τον 5001 ή τον 5009 ενώ σε όλες τις άλλες περιπτώσεις θα εντόπιζε το σωστό value (εκτός αν το insert κινούνταν γρήγορα και είχε προλάβει να ανανεώσει και την τιμή κάποιου αντιγράφου).