

# ΕΞΑΜΗΝΙΑΙΑ ΕΡΓΑΣΙΑ ΣΤΑ ΚΑΤΑΝΕΜΗΜΕΝΑ ΣΥΣΤΗΜΑΤΑ

Ακ. έτος 2020-2021, 9ο Εξάμηνο, Σχολή ΗΜ&ΜΥ

ΑΝΑΣΤΑΣΑΚΗΣ ΖΑΧΑΡΙΑΣ

- AM: 03116675
- Email: [zaxarisanastasakis@gmail.com](mailto:zaxarisanastasakis@gmail.com)

ΑΝΤΩΝΙΟΥ ΚΩΝΣΤΑΝΤΙΝΟΣ

- AM: 03116169
- Email: [kostas.an2016@gmail.com](mailto:kostas.an2016@gmail.com)

ΒΑΚΗΣ ΜΙΧΑΗΛ

- AM: 03113137
- Email: [mmvaki@hotmail.com](mailto:mmvaki@hotmail.com)

Στην εργασία αυτή θα υλοποιήσουμε το ToyChord, μια απλοποιημένη έκδοση του Chord. Η εφαρμογή που θα δημιουργήσουμε είναι ένα file sharing application με πολλαπλούς κατανεμημένους κόμβους DHT.

Το σύστημα μας χειρίζεται εισαγωγές νέων κόμβων `join(nodeID)` και αποχωρήσεις κόμβων `depart(nodeID)`, για αυτό το λόγο ορίζουμε έναν κόμβο, ο οποίος δέχεται όλα τα αιτήματα `join` (Bootstrap Node), του οποίου η IP είναι γνωστή σε κάθε άλλο κόμβο. Κάθε κόμβος υλοποιεί τις λειτουργίες `insert(key, value)`, `query(key)` και `delete(key)` για `<key, value>` ζεύγη. Ακόμη, κάθε `key` θα πρέπει να περάσει από hash function πριν χρησιμοποιηθεί για οποιαδήποτε από τις παραπάνω λειτουργίες ώστε να βρεθεί η σωστή θέση στον δακτύλιο.

Τα αιτήματα για την εκτέλεση των παραπάνω λειτουργιών δίνονται σ' έναν CLI client, το οποίο δίνει την δυνατότητα στον χρήστη να εκτελέσει τις παρακάτω λειτουργίες:

- `insert<key> <value><node>`: Εισαγωγή δεδομένων.
- `delete<key><node>`: Διαγραφή δεδομένων.
- `query<key><node>`: Αναζήτηση δεδομένων
- `depart<node>`: Αποχώρηση κόμβου.
- `overlay<node>`: Εκτύπωση της τοπολογίας του δικτύου
- `help`: Επεξήγηση των παραπάνω εντολών.
- `readfile<filename>` : Εκτέλεση ενός από τα 3 αρχεία που μας δόθηκαν

### Σημείωση:

Σε όλες τις λειτουργίες του CLI (εκτός του `help`) έχει προστεθεί μία παραπάνω παράμετρος σε σχέση με τα ζητούμενα της εκφώνησης. Η παράμετρος αυτή είναι η `<node>` την οποία χρησιμοποιούμε ως εξής:

- `IP:Port` : Έχουμε την δυνατότητα να δώσουμε συγκεκριμένο IP και Port για να επιλέξουμε τον κόμβο στον οποίον θα σταλεί το αίτημα. Η λειτουργία αυτή βοηθάει στην αποσφαλμάτωση του κώδικα.
- `"random"` : Δίνεται επίσης η δυνατότητα τυχαίας επιλογής κόμβου που θα αποσταλεί το αίτημα.

Επίσης προστέθηκε και η λειτουργία `readfile<filename>` η οποία παίρνει ένα από τα 3 αρχεία που μας δίνονται και αποστέλλει τα κατάλληλα αιτήματα προς τους κόμβους.

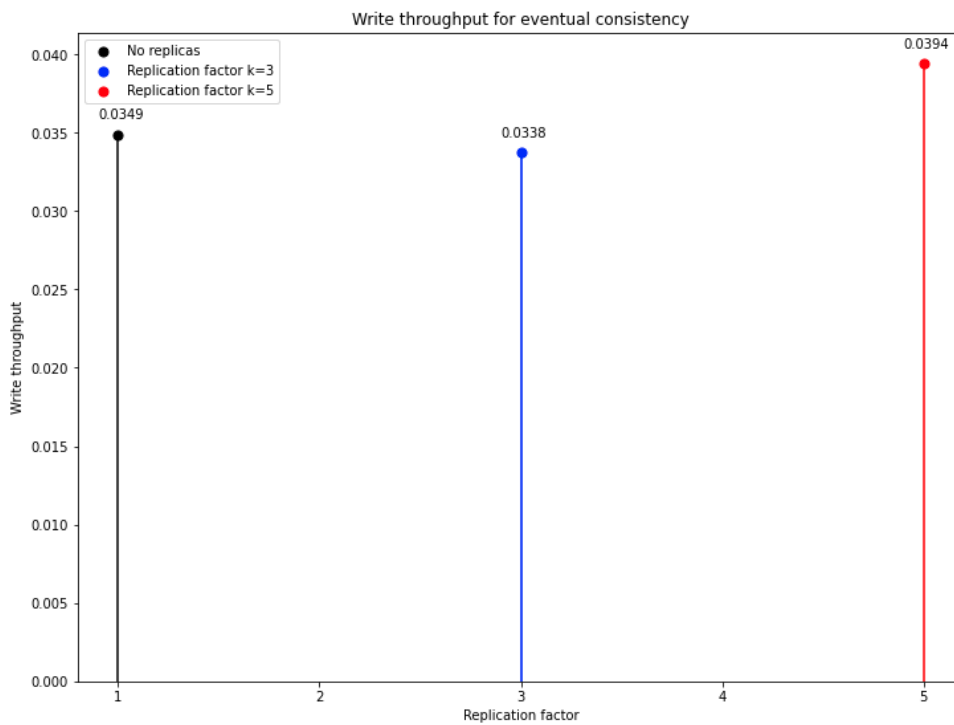
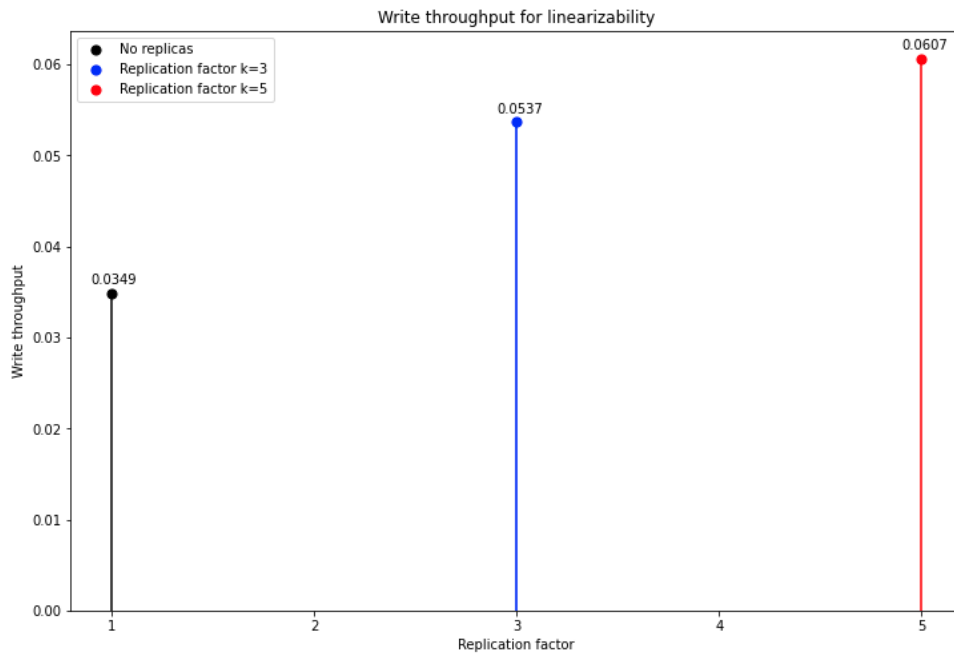
## Πειράματα:

Καλούμαστε να εκτελέσουμε τα παρακάτω πειράματα:

### Πείραμα1 :

Εισάγουμε σε ένα DHTμε 10 κόμβους όλα τα κλειδιά που βρίσκονται στο αρχείο `insert.tx` με  $k=1$  (χωρίς replication),  $k=3$  και  $k=5$  και με linearizability και eventual consistency και καταγράφουμε το write throughput του συστήματος.

Αρχικά θα παρουσιάσουμε τις γραφικές παραστάσεις για τα 6 setups που μας ζητούνται. Στα παρακάτω γραφήματα αναγράφεται ο χρόνος που πήρε η εισαγωγή των κλειδιών προς τον αριθμό τους σε κάθε έναν από τους διαφορετικούς τύπους consistencies:



### Συμπεράσματα:

Αρχικά αξίζει να σημειώσουμε ότι πρακτικά τα πειράματα ήταν 5 σε αριθμό. Αυτό συμβαίνει διότι για  $k = 1$  δεν διαμοιράζονται αντίγραφα και επομένως δεν έχει νόημα να συζητάμε για consistencies.

Όσον αφορά το linearizability παρατηρούμε ότι καθώς αυξάνουμε το  $k$  έχουμε αύξηση και το write throughput. Αυτό συμβαίνει διότι όσο αυξάνεται το  $k$ , αυξάνεται και ο αριθμός των αντιγράφων που πρέπει ο κάθε κόμβος να μεταφέρει στους  $k-1$  successors του. Έχοντας chain replication linearizability τα αιτήματα του κόμβου προς τους successors του γίνονται σειριακά, με αποτέλεσμα όσο αυτά αυξάνονται να αυξάνεται και ο χρόνος διεκπεραίωσης του αρχικού αιτήματος.

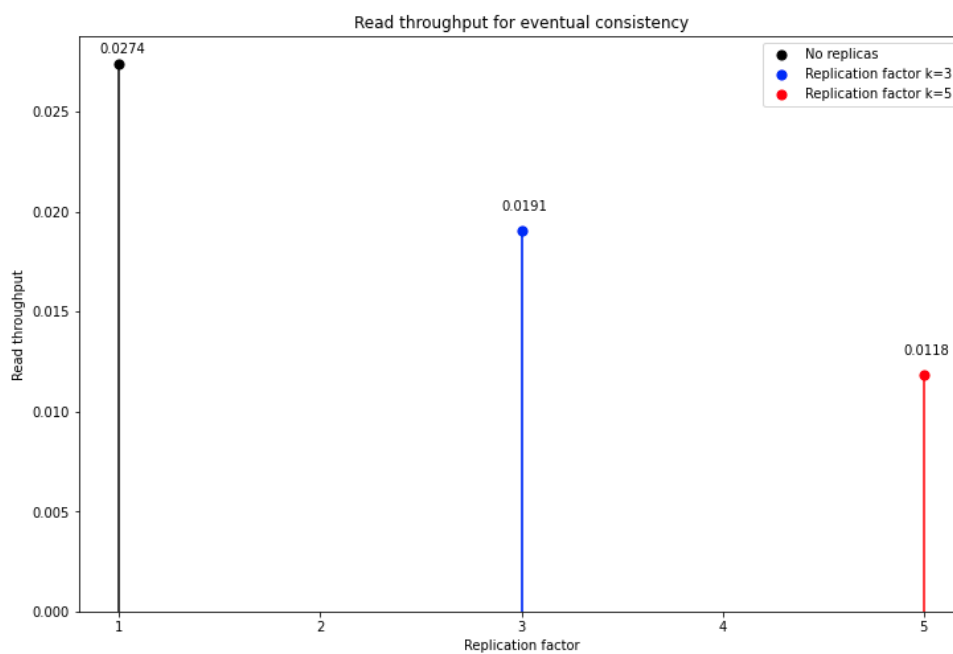
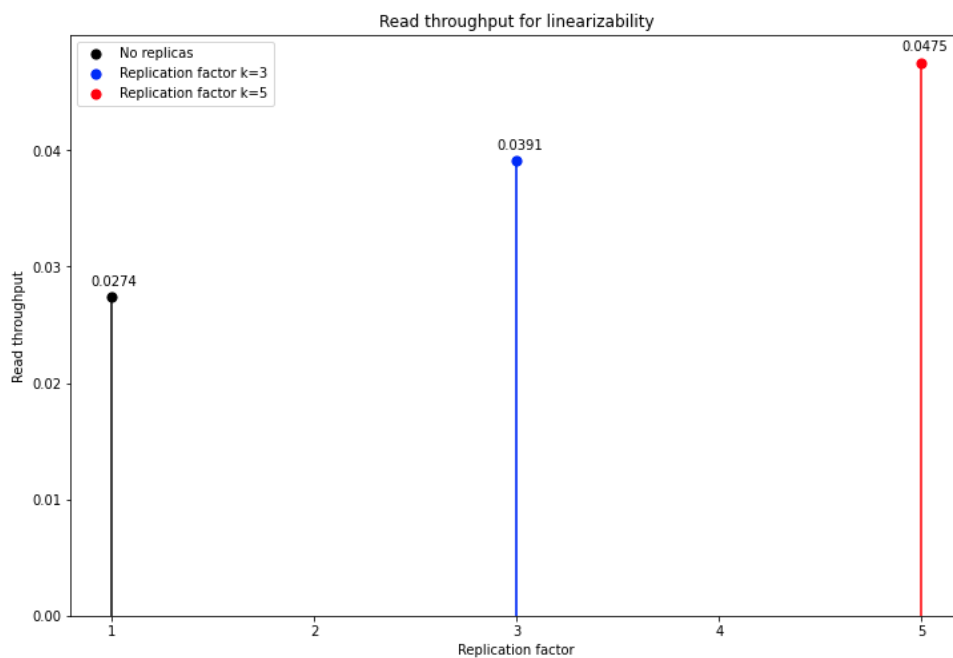
Από την άλλη μεριά για την περίπτωση του eventual consistency παρατηρούμε ότι η αύξηση στο write throughput όσο αυξάνεται το  $k$  είναι αρκετά μικρή. Αυτό οφείλεται στο ότι το eventual consistency λειτουργεί πολύ διαφορετικά από το linearizability. Συγκεκριμένα ο κόμβος που είναι υπεύθυνος για κάποιο κλειδί θα επιστρέφει το αποτέλεσμα του write, χωρίς να περιμένει την μεταβίβαση των replicas στους  $k-1$  successors του, όπως γίνεται στην περίπτωση του linearizability. Από τα παραπάνω και σε συνδυασμό με την χρήση threads για την παράλληλη αποστολή των αιτημάτων για write οι χρόνοι που παρατηρούμε είναι σημαντικά μειωμένοι σε σχέση με την προηγούμενη περίπτωση.

Συμπερασματικά όσον αφορά το write throughput το eventual consistency δίνει αρκετά μικρότερους χρόνους σε σχέση με το chain replication linearizability καθώς το eventual consistency δίνει την δυνατότητα για παράλληλη εκτέλεση των requests.

### Πείραμα2 :

Για τα 6 διαφορετικά setups του προηγούμενου ερωτήματος, διαβάζουμε όλα τα keys που βρίσκονται στο αρχείο query.txt και καταγράφουμε το read throughput.

Αρχικά θα παρουσιάσουμε τις γραφικές παραστάσεις για τα 6 setups που μας ζητούνται. Στα παρακάτω γραφήματα αναγράφεται ο χρόνος που πήρε το διάβασμα των κλειδιών προς τον αριθμό τους σε κάθε έναν από τους διαφορετικούς τύπους consistencies:



### Συμπεράσματα:

Ομοίως με την προηγούμενη περίπτωση πρακτικά τα πειράματα ήταν 5 σε αριθμό, καθώς για  $k = 1$  δεν διαμοιράζονται αντίγραφα και επομένως δεν έχει νόημα να συζητάμε για consistencies.

Όσον αφορά το linearizability παρατηρούμε ότι όσο αυξάνεται το  $k$  αυξάνεται και η τιμή του του read throughput. Αυτό συμβαίνει διότι όσο αυξάνεται η τιμή του  $k$  αυξάνεται και ο αριθμός των αντιγράφων που πρέπει να διαμοιράσει ο κάθε κόμβος στους  $k - 1$  successors του, και κατά συνέπεια επειδή η διαδικασία του read ολοκληρώνεται όταν εντοπίσουμε το αντίγραφο στον  $k - 1$  successor, όσο μεγαλύτερη είναι η τιμή του  $k$  τόσο περισσότερο χρόνο μας παίρνει για να φτάσουμε στον  $k - 1$  successor.

Όσον αφορά την περίπτωση του eventual consistency παρατηρούμε ότι όσο αυξάνεται η τιμή του  $k$  μειώνετε το read throughput. Αυτό συμβαίνει διότι η διαδικασία του read, σε αντίθεση με το linearizability, ολοκληρώνεται μόλις βρούμε το key που αναζητούμε ανεξαρτήτως σε ποιον κόμβο βρισκόμαστε, είτε είμαστε στον υπεύθυνο, για αυτό το κλειδί, κόμβο είτε είμαστε σε κάποιο κόμβο ο οποίος έχει αντίγραφο του κλειδιού που αναζητούμε.

Συμπερασματικά, όσον αφορά το read throughput το eventual consistency δίνει αρκετά μικρότερους χρόνους σε σχέση με το chain replication linearizability.

Γενικότερα, παρατηρούμε ότι και στα δύο παραπάνω πειράματα, οι χρόνοι που παίρνουμε από το eventual consistency είναι σταθερά μικρότεροι από τους αντίστοιχους χρόνους που λαμβάνουμε από το chain replication linearizability. Αξίζει να σημειώσουμε ότι η περίπτωση του eventual consistency, εγκυμονεί κινδύνους όσον αφορά το read throughput καθώς υπάρχει πιθανότητα να διαβάσουμε stale τιμές.

### Πείραμα3 :

Για DHT με 10 κόμβους και  $k=3$ , εκτελούμε τα requests του αρχείου requests.txt και καταγράφουμε τις απαντήσεις των queries σε περίπτωση linearization και eventual consistency.

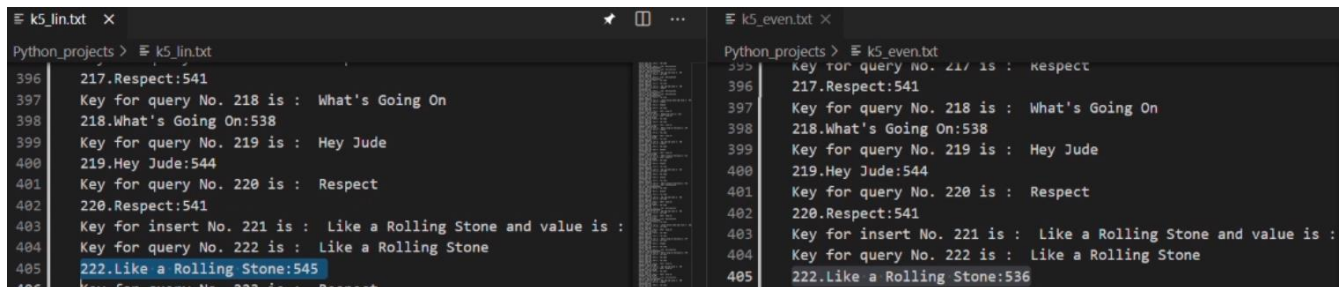
Γενικά, στο eventual consistency υπάρχει κίνδυνος τα δεδομένα που διαβάζουμε από κάποιον κόμβο να είναι stale τιμές, ενώ αντιθέτως στο chain replication linearizability εξασφαλίζεται πάντα η ανάγνωση fresh τιμών, με κόστος μεγαλύτερου χρόνου για τα read και write throughput.

Στο συγκεκριμένο πείραμα περιμέναμε να παρατηρήσουμε διαφορές μεταξύ των αποτελεσμάτων που έδωσαν τα queries του αρχείου requests.txt . Παρόλα αυτά πιθανώς λόγω του ότι έχουμε  $k = 3$  και 10 κόμβους έχουμε 70% πιθανότητα το query να απευθυνθεί σε κόμβο ο οποίος δεν έχει το αρχείο που αναζητούμε, δίνοντας έτσι παραπάνω χρόνο στα threads να διαδώσουν τα replicas. Χρησιμοποιώντας ένα replication factor με μεγαλύτερη τιμή, αυξάνουμε την πιθανότητα να διαβάσουμε κάποια stale τιμή. Μία άλλη εναλλακτική θα ήταν να προσθέταμε ένα μικρό delay στην διάδοση των replicas από τα threads, αυξάνοντας με αυτόν τον τρόπο το χρονικό παράθυρο στο οποίο μπορεί να γίνει ανάγνωση κάποιας stale τιμής.

Για του λόγου το αληθές αποφασίσαμε να τρέξουμε το ίδιο πείραμα για  $k = 5$  και για  $k = 7$ .

- **$k = 5$ :** Για  $k = 5$  οι διαφορές μεταξύ των 2 αρχείων φαίνονται στις παρακάτω φωτογραφίες:

```
Not the same Files
Differences:
['222', 'Like a Rolling Stone', '536']
```



```
Python_projects > k5_lin.txt
396 217.Respect:541
397 Key for query No. 218 is : What's Going On
398 218.What's Going On:538
399 Key for query No. 219 is : Hey Jude
400 219.Hey Jude:544
401 Key for query No. 220 is : Respect
402 220.Respect:541
403 Key for insert No. 221 is : Like a Rolling Stone and value is :
404 Key for query No. 222 is : Like a Rolling Stone
405 222.Like a Rolling Stone:545
406 Key for query No. 223 is : Respect

Python_projects > k5_even.txt
key for query no. 217 is : respect
396 217.Respect:541
397 Key for query No. 218 is : What's Going On
398 218.What's Going On:538
399 Key for query No. 219 is : Hey Jude
400 219.Hey Jude:544
401 Key for query No. 220 is : Respect
402 220.Respect:541
403 Key for insert No. 221 is : Like a Rolling Stone and value is :
404 Key for query No. 222 is : Like a Rolling Stone
405 222.Like a Rolling Stone:536
```

Παρατηρούμε ότι ακόμα και για  $k = 5$  οι διαφορές μεταξύ των αρχείων είναι ελάχιστες και συγκεκριμένα μία. Σημαντική είναι και η επίδραση του παράγοντα τύχης καθώς τα requests γίνονται κάθε φορά σε τυχαίο κόμβο.

- **$k = 7$ :** Για  $k = 7$  οι διαφορές μεταξύ των 2 αρχείων φαίνονται στην παρακάτω φωτογραφία:

```
Not the same Files
Differences:
['64', 'Hey Jude', '511']
['87', 'Hey Jude', '513']
['107', 'Respect', '516']
['117', 'Hey Jude', '523']
['118', 'Hey Jude', '523']
['167', 'Hey Jude', '532']
['172', 'Hey Jude', '534']
['193', 'Hey Jude', '535']
['198', 'Hey Jude', '539']
['202', 'Respect', '528']
['439', 'Hey Jude', '586']
['468', 'Hey Jude', '591']
['473', 'Hey Jude', '594']
['475', 'Hey Jude', '594']
['482', 'Hey Jude', '596']
```

Παρατηρούμε ότι οι διαφορές στην συγκεκριμένη περίπτωση είναι περισσότερες σε σχέση με τις 2 προηγούμενες περιπτώσεις. Πλέον ο παράγοντας τύχη παίζει αρκετά μικρότερο ρόλο καθώς η μεταφορά των replicas παίρνει περισσότερο χρόνο, ενώ τα read requests αποστέλλονται με την ίδια συχνότητα. Επομένως επειδή το διάβασμα γίνεται από οποιονδήποτε κόμβο (στο eventual consistency) είναι πολύ πιθανό να διαβάσουμε κάποια τιμή η οποία δεν έχει προλάβει να ανανεωθεί.