Αναφορά Εξαμηνιαίας Εργασίας στο μάθημα «Κατανεμημένα Συστήματα»

Ομάδα:

Ευάγγελος Μεκλής 03116644

Χρήστος Συρμακέζης 03115182

Παναγιώτης Χαρατσάρης 03116024

Δομή Εργασίας: Η εργασία εκπονήθηκε με χρήση της γλώσσας javascript και συγκεκριμένα με την χρήση της node.js. Τα σημαντικότερα αρχεία βρίσκονται στον φάκελο src. Ο φάκελος src είναι δομημένος ως εξής:

- Config : περιέχει το αρχείο Message Strings που καθορίζει τα μηνύματα που μπορούν να σταλθούν μεταξύ των κόμβων του δικτύου.
- ConsoleCommands: ο φάκελος αυτός περιέχει αρχεία που εκτελούνται μετά από μία εντολή του χρήστη.
- Messages: περιέχει τον κώδικα για όλα τα μηνύματα που ανταλλάζονται μεταξύ των κόμβων του δικτύου.
- Utils: περιέχει συναρτήσεις που χρησιμοποιούνται στο project, αλλά και το αρχείο SocketClient που καθορίζει τις παραμέτρους που θα δέχονται τα μηνύματα.

Θεωρητικό Μέρος: Στα αποτελέσματα του πειραματικού μέρους περιμένουμε τα παρακάτω:

Στο πλαίσιο της εργασίας κληθήκαμε να σχεδιάσουμε το ToyChord, μια απλοποιημένη εκδοχή του Chord. Η εφαρμογή που αναπτύξαμε είναι ένα application με πολλαπλούς κατανεμημένους κόμβους DHT, ενώ παράλληλα υλοποιεί μια απλοποιημένη εκδοχή του πρωτοκόλλου του Chord. Στην υλοποιήση αυτή, καλύφθηκαν όλα τα ζητούμενα της εργασίας, ενώ στο linearizability επιλέχθηκε το chain replication.

Κατέβασμα και τρέξιμο του application: Η εφαρμογή είναι διαθέσιμη στο github αλλά και στο tarball που θα υποβληθεί. Ο τρόπος που τρέχουμε την εφαρμογή είναι ο παρακάτω:

1. Σε ένα cmd κατευθυνόμαστε στον φάκελο Chord-DHT.
2. Στον φάκελο αυτό τρέχουμε την εντολή node index.js replication <X> type <Y>
    a. Οι τιμές που λαμβάνει το X είναι φυσικοι αριθμοί 1 έως n και αφορά το πόσα αντίγραφα θέλουμε να φτιάχνονται
    b. Οι τιμές που λαμβάνει το Y είναι 2, “chain-replication” και “eventual-consistency”, ανάλογα με τη μορφή συνέπειας που θέλουμε να επιλέξουμε
3. Για να εισάγουμε έναν νέο κόμβο στο δίκτυο ανοίγουμε ένα νέο cmd στον φάκελο Chord-DHT και τρέχουμε την εντολή: node index.js ip_address:port . Στα πεδία ip address και port εισάγουμε το ip address και το port του κόμβου που δημιουργείται στο βήμα 2, ο οποίος θα είναι και ο bootstrap κόμβος.
4. Μετά το τρέξιμο της εντολής 3 συνδέεται ο κόμβος στο δίκτυο του Chord και λαμβάνει ένα port , ένα id και προφανώς την ip που ήδη έχει. Σε τοπικό δίκτυο, αντί για ip address μπορεί ο χρήστης να γράφει localhost.

Πειραματικό Μέρος

Πειράματα 1 και 2:

Στο linearizability περιμένουμε όσο το k αυξάνεται τα inserts να αυξάνονται σε χρόνο καθώς θα χρειάζεται επιπλέον χρόνος για να δημιουργηθούν τα αντίγραφα. Παράλληλα, τα queries θα αυξάνονται σε χρόνο, καθώς θα απαιτείται να φτάσουν στο τελευταίο replica και άρα θα πρέπει να διασχίζουν μεγαλύτερη απόσταση στον δακτύλιο. Στο eventual consistency περιμένουμε τα inserts να μην μεταβάλλονται σε χρόνο όσο αυξάνεται το k, καθώς από τα inserts το ack γυρνάει μετά την πρώτη εισαγωγή, ενώ τα queries πιθανώς θα μειώνονται σε χρόνο, καθώς επιστρέφουν την πρώτη τιμή που θα βρουν και όχι την τιμή του τελευταίου replica. Συνεπώς το eventual consistency περιμένουμε να είναι «γρηγορότερο» από το chain-replication που επιλέξαμε για το linearizability.

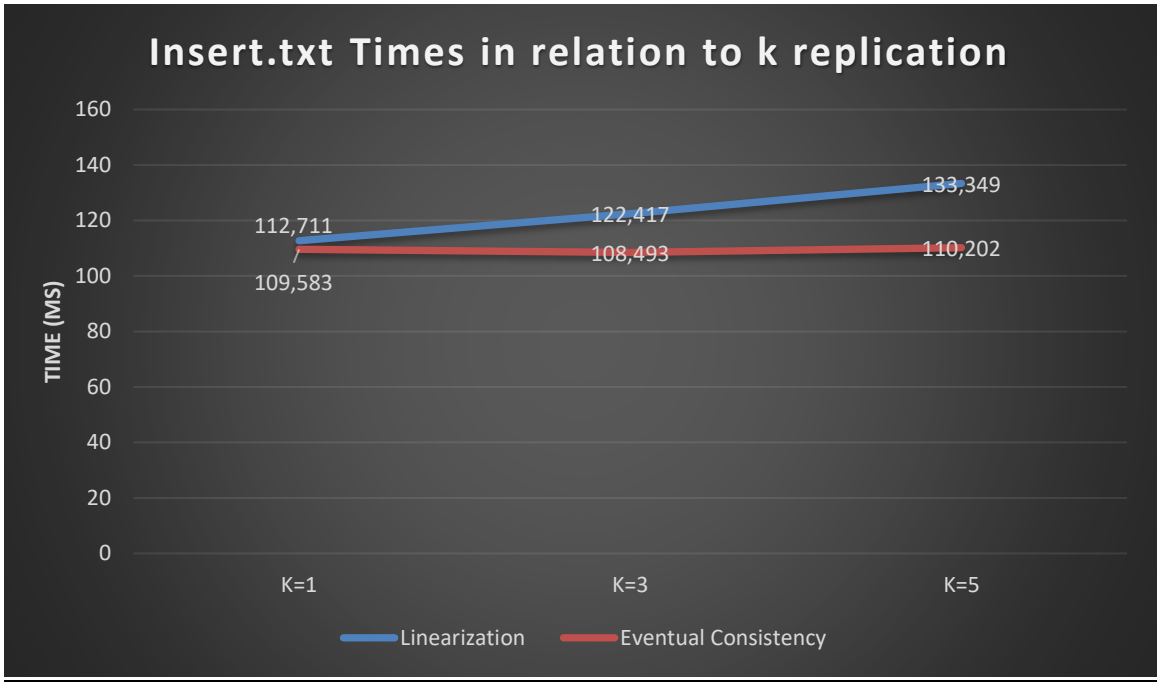
Αποτελέσματα πειραμάτων 1 και 2:

Time (ms)	**Insert.txt**		**Query.txt**	
	Linearization	**Eventual Cons**	**Linearization**	**Eventual Cons**
K=1	112.711	109.583	74.511	108.301
K=3	122.417	108.493	94.572	92.477
K=5	133.349	110.202	97.054	73.210

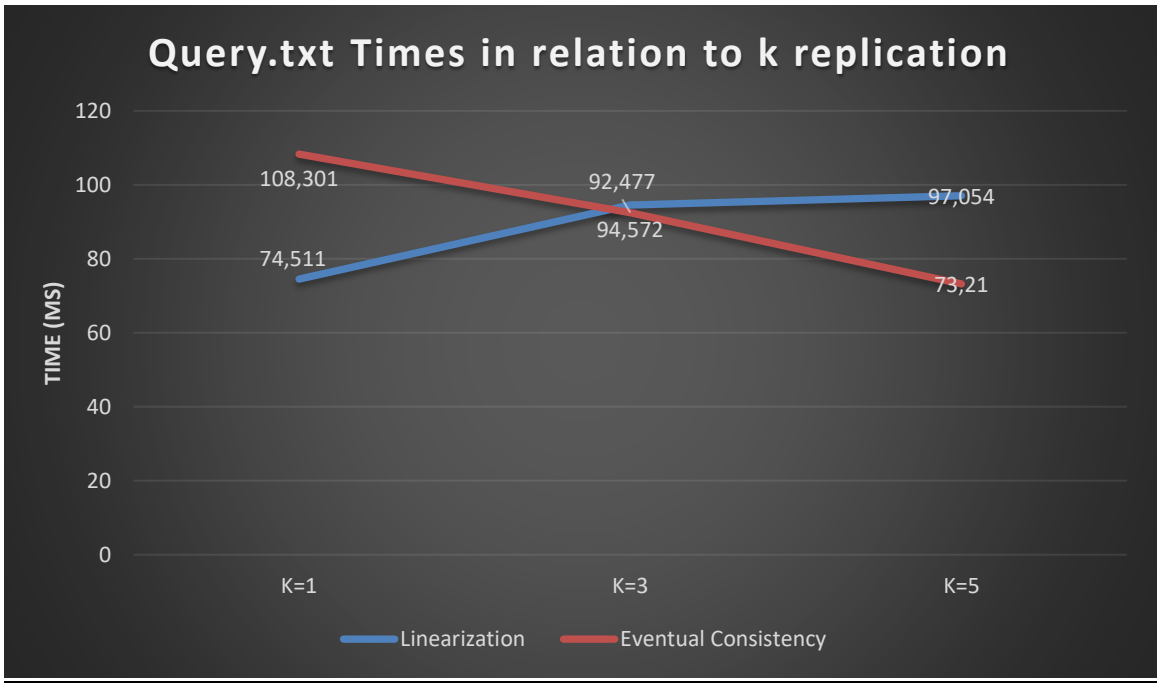
Πίνακας 1.1: Αποτελέσματα πειραμάτων 1 και 2 (καταγραφές χρόνων)

Throughput (Time/number of commands)	**Insert.txt**		**Query.txt**	
	Linearization	**Eventual Cons**	**Linearization**	**Eventual Cons**
K=1	0.225	0.219	0.149	0.146
K=3	0.225	0.217	0.189	0.185
K=5	0.267	0.220	0.194	0.217

Πίνακας 1.2: Αποτελέσματα πειραμάτων 1 και 2 (καταγραφή Throughput)



Γράφημα 1.1: Χρόνοι για insert.txt αναφορικά με τον αριθμό k replication



Γράφημα 1.2: Χρόνοι για query.txt αναφορικά με τον αριθμό k replication

Πείραμα 3:

Στο πείραμα αυτό, περιμένουμε πως το eventual consistency θα μας δίνει τιμές που ορισμένες φορές θα είναι stale, καθώς δεν θα έχουν προλάβει να διαδοθούν οι αλλαγές σε όλα τα replicas. Συνεπώς είναι πολύ πιθανό να επιστραφεί κάποιο αποτέλεσμα με διαφορετική τιμή από την καινούρια. Αντίθετα στο linearization πάντα θα επιστρέφεται η ίδια τιμή στα queries καθώς θα επιστρέφεται η τιμή από τον τελευταίο κόμβο στην αλυσίδα των replicas και άρα θα εξασφαλίζουμε πως οποιαδήποτε τιμή θα έχει διαδοθεί σε όλα τα replicas.

Request	**Answer**	
insert, Hey Jude, 501 query, Hey Jude query, Hey Jude query, Hey Jude	**Linearization**	Στον τρόπο αυτό επιστράφηκε η τιμή 501 σε όλους τους κόμβους.
	Eventual Consistency	Στον τρόπο αυτό επιστράφηκε στον έναν κόμβο η τιμή 500 και στους άλλους δύο κόμβους η τιμή 501

Σχολιασμός Αποτελέσματος: Πριν το insert η τιμή που υπήρχε ήταν η 500. Παρατηρήσαμε λοιπόν ότι επιστράφηκε στο eventual consistency μια stale τιμή, καθώς δεν είχε προλάβει να διαδοθεί η αλλαγή. Αντίθετα στο linearization, η απάντηση ήρθε παντού με την νέα τιμή, καθώς δεν υπάρχει περίπτωση να υπάρξουν stale τιμές.

Συμπέρασματα: Παρατηρούμε πως υπάρχει ένα tradeoff μεταξύ της ταχύτητας εκτέλεσης ερωτημάτων στο linearization και στο eventual consistency. Το linearization παρότι πιο αργό στην ταχύτητα εκτέλεσης, επιτυγχάνει να επιστρέφει κάθε φορά την καινούρια τιμή και εξασφαλίζει ότι δεν θα επιστρέφει stale τιμές.