Κατανεμημένα Συστήματα

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, ΕΜΠ

Εξαμηνιαία εργασία στα Κατανεμημένα Συστήματα - ToyChord

Δημήτρης Γιάγκος – 03116302
Κωνσταντίνος Σταυρακάκης – 03116155
Ραφαήλ Δάσκος – 03116049
9ο εξάμηνο, 2020-2021

Εισαγωγή

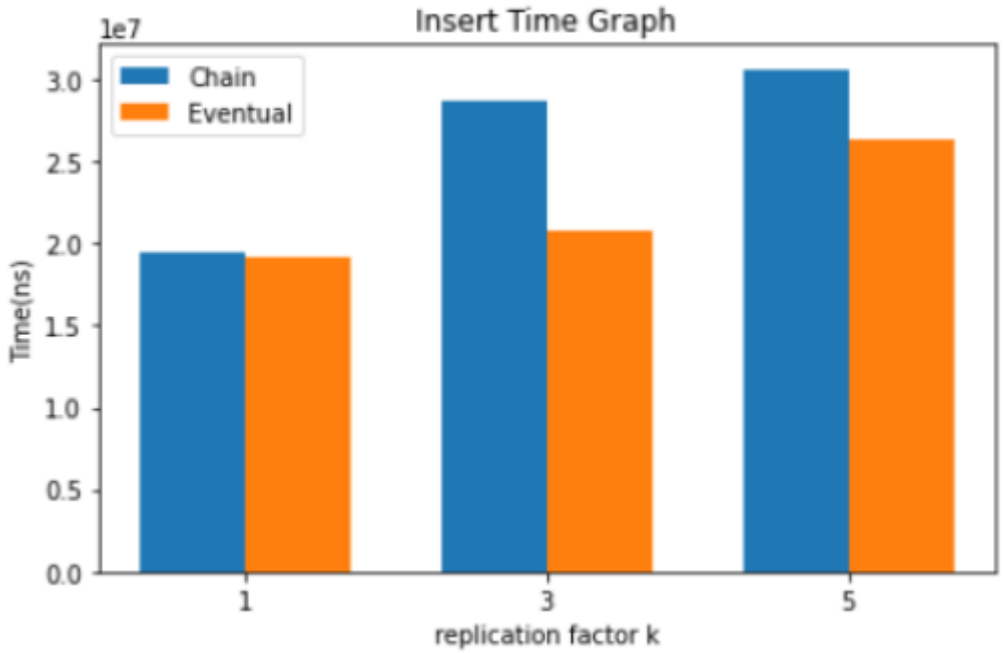
Η υλοποίηση του ToyChord έγινε σε java με τη βοήθεια του rmiregistry που μας παρέχει εύκολη και γρήγορη πρόσβαση σε απομακρυσμένους κόμβους με τη χρήση διαφορετικών VMs στον ~okeanos. Στην υλοποίησή μας χρησιμοποιούμε και finger tables για τη δρομολόγηση ώστε να έχουμε ακόμα καλύτερη απόδοση και να μη χρειάζεται ο κάθε κόμβος να δρομολογεί μηνύματα στον αμέσως επόμενο του στην αλυσίδα αλλά σε αυτόν που θεωρεί πιο κατάλληλο (με βάση τα entries του finger table) για την επεξεργασία του κάθε μηνύματος.

Τα ακόλουθα πειράματα έγιναν με τη χρήση 5 VMs στα οποία έγιναν deploy 10 κόμβοι του δικτύου και ένας γενικός υπεύθυνος για την ομαλή λειτουργία του συστήματος. Τα requests δρομολογούνται από τον server μας (bootstrap node) στους υπόλοιπους κόμβους (peers), με τυχαίο αρχικό κάθε φορά.

Στη συνέχεια θα παρουσιάσουμε τα αποτελέσματα που λαμβάνουμε όταν τρέξουμε το insert.txt, query.txt και requests.txt.

inserts.txt

Από την εισαγωγή ζευγών key-values στους κόμβους του δικτύου μας λαμβάνουμε τους ακόλουθους μέσους χρόνους εκτέλεσης των εντολών μας από τους οποίους μπορούμε να αποφανθούμε και για την απόδοση (throughput) του συστήματος. Τα πειράματα που πραγματοποιήθηκαν είναι με τη χρηση linearizability, και συγκεκριμένα chain replication, και eventual consistency για τη μετάδοση των replicas που θα ελεγχθούν οι τιμές 1, 3, 5 για κάθε περίπτωση. Έτσι λαμβάνουμε τα ακόλουθα αποτελέσματα:

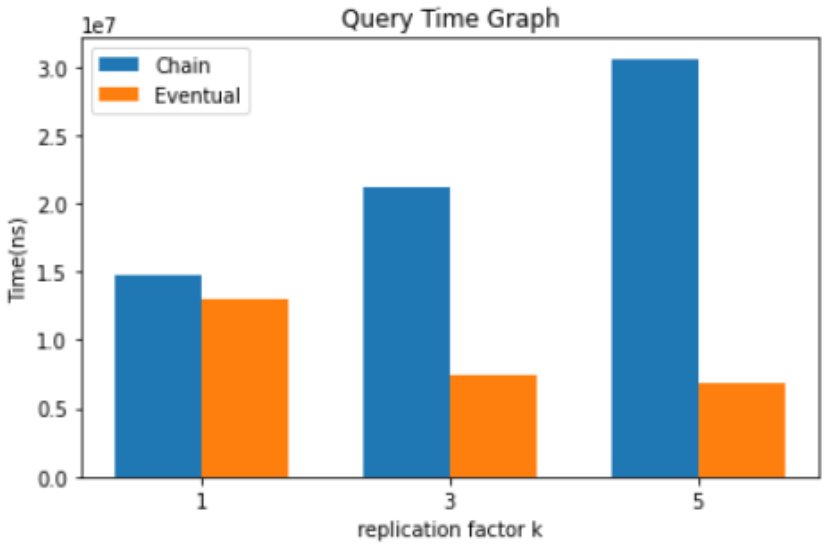


Συμπεράσματα:

Αρχικά, όπως μπορούμε να δούμε, όταν έχουμε replication factor 1 δεν έχει διαφορά ποια λογική χρησιμοποιούμε για τη δημιουργία των αντιγράφων δεδομένου ότι δε δημιουργούνται άλλα. Αυτό οδηγεί στο να έχουμε όμοια αποτελέσματα για τα δύο πειράματά μας. Όταν, όμως, αυξάνουμε τον αριθμό των αντιγράφων που θέλουμε να διατηρούνται στην αλυσίδα μας τότε, στην περίπτωση του linearizability (chain replication), αρχίζουμε και βλέπουμε αύξηση του χρόνου απόκρισης του συστήματος που σημαίνει μείωση του throughput. Αυτό γίνεται δεδομένου ότι χρειάζεται να τελειώσει η εγγραφή σε όλους τους κόμβους που πρέπει προτού επιστρέψει το αποτέλεσμα ο αρχικός κόμβος που ζητήθηκε η δρομολόγηση του request. Στην αντίθετη περίπτωση, όταν έχουμε δηλαδή eventual consistency το σύστημα δε νοιάζεται για το τι θα κάνουν οι υπόλοιποι κόμβοι αλλά μόλις γίνει η εγγραφή στον πρώτο κόμβο στέλνει μήνυμα ότι έγινε για να συνεχίσει τις υπόλοιπες λειτουργίες του. Αυτό οδηγεί σε παρεμφερή αποτελέσματα στις διαφορετικές τιμές αντιγράφων. Οι διαφορές που παρατηρούνται είναι εξαιτίας των ανόμοιων chords αλλά και random αρχικών κόμβων στα οποία ζητούνται τα inserts. Έτσι, μπορεί σε ένα πείραμα να είμαστε κάπως πιο τυχεροί ή άτυχοι είτε με την τοπολογία των 10 κόμβων του δικτύου μας είτε με την επιλογή των κόμβων που ζητάται να αρχίσουν την εισαγωγή του καθενός δεδομένου. Βέβαια, μια αύξηση στον χρόνο απόκρισης είναι λογικό να εμφανιστεί με την αύξηση των αντιγράφων δεδομένου ότι κάθε κόμβος δεν μπορεί να αναλάβει ταυτόχρονα δύο inserts οπότε, και αφού γίνονται πολλές παράλληλες εγγραφές στο Chord εμφανίζονται παύσεις όταν επιλεγεί κόμβος που ήδη γράφεται κάτι σε αυτόν, εξ ου και η μείωση του throughput. Αυτό το φαινόμενο είναι πιο εύκολο παρατηρήσιμο για k = 5 δεδομένου των πολλών αντιγράφων που επιθυμούμε να δημιουργήσουμε. Οι παύσεις του σύστήματός μας, δηλαδή, είναι αποτέλεσμα της αύξησης του traffic που εμφανίζεται το οποίο και αναπόφευκτα οδηγεί σε αύξηση του latency.

query.txt

Στην περίπτωση της αναζήτησης στοιχείων στο Chord έχουμε τα ακόλουθα δεδομένα. Στην περίπτωση chain replication πρέπει να φτάσουμε στο τελευταίο αντίγραφο μιας αλυσίδας για να αποφανθούμε για το read ενώ στην περίπτωση του eventual consistency αν βρεθεί αντίγραφο σταματάμε την αναζήτηση και επιστρέφουμε τιμή. Και στις δύο περιπτώσεις βέβαια ο τρόπος για να δούμε ότι μια τιμή δεν υπάρχει στην αλυσίδα είναι να φτάσουμε στον κόμβο ο οποίος με βάση το id του θα ήταν υπεύθυνος για να έχει το κλειδί αν δεν είχαμε καθόλου replication.



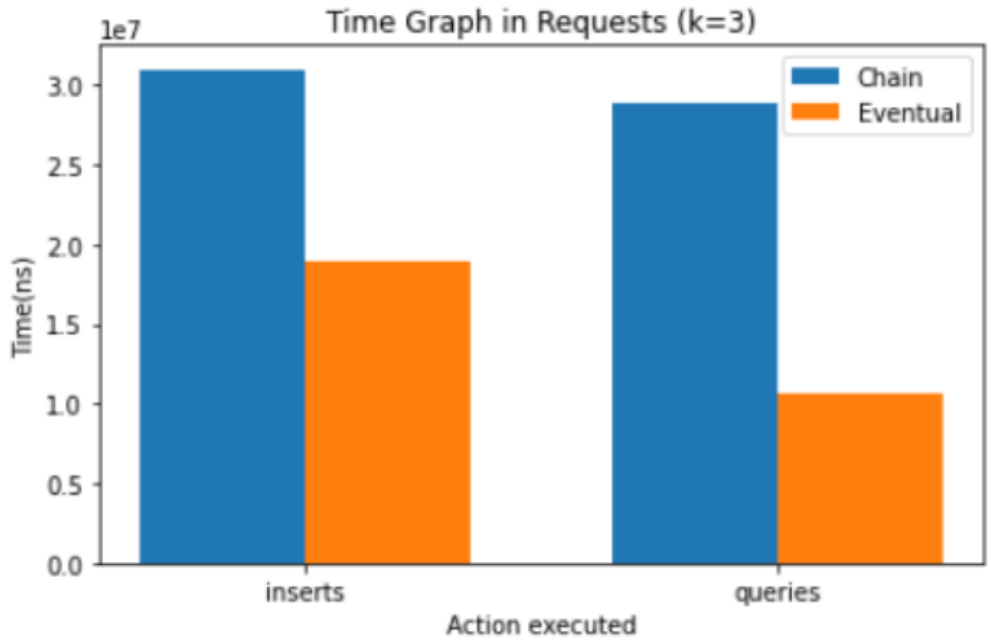
Συμπεράσματα:

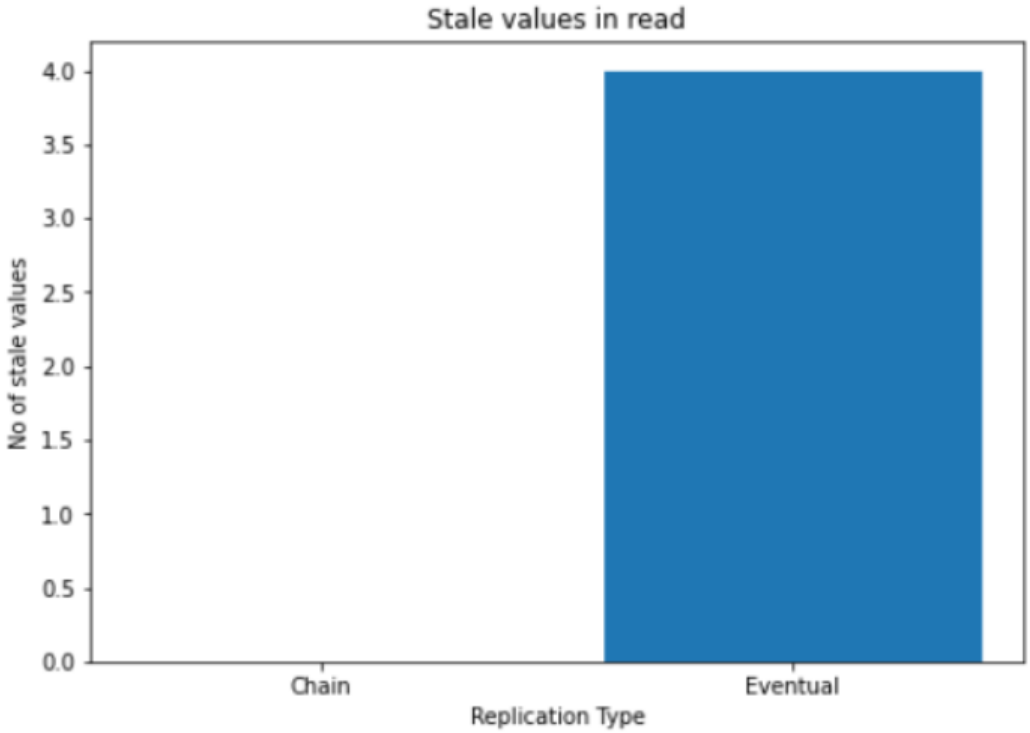
Στην περίπτωση της αναζήτησης αλλάζουν λίγο τα αποτελέσματα σε σχέση με την εισαγωγή. Βέβαια, και πάλι μπορούμε να δούμε αρκετά παρεμφερείς τιμές για την περίπτωση του replication factor ίσο με 1 δεδομένου ότι και στις δύο αυτές υλοποιήσεις θα αναζητηθεί το στοιχείο από την αρχή της αλυσίδας, τον κόμβο δηλαδή που είναι υπεύθυνος για το κλειδί που αναζητούμε. Όσο, όμως, αυξάνεται ο αριθμός των αντιγράφων που έχουμε στο δίκτυο μας παρατηρούμε αντίθετη φορά στην απόδοση. Μπορούμε να αποφανθούμε ότι στην περίπτωση του linearizability, δεδομένου ότι πρέπει να φτάσουμε στο τέλος της αλυσίδας για να πάρουμε το αποτέλεσμα, θα κάνουμε επιπλέον βήματα ακόμα και αν αρχίσουμε από κόμβο που με την τοπολογία που έχουμε διαθέτει replica του κλειδιού που αναζητούμε. Από την άλλη, όταν έχουμε eventual consistency μπορούμε να παρατηρήσουμε πως όσο περισσότερα αντίγραφα έχουμε τόσο μικρότεροι είναι οι χρόνοι απόκρισης που συνεπάγεται σε μεγαλύτερο throughput. Αυτό συμβαίνει δεδομένου ότι έχουμε περισσότερα αντίγραφα άρα και περισσότερες πιθανότητες με μια απλή επιλογή κόμβου, παρόλο που είναι τυχαία, να επιλέξουμε έναν που έχει το κλειδί που αναζητούμε. Ακόμα, αφού δε μας νοιάζει από ποιον κόμβο θα διαβάσουμε σημαίνει ότι θα γυρίσουμε αμέσως με το που βρούμε ένα οποιοδήποτε αντίγραφο και άρα το δίκτυο έχει πιο γρήγορες απαντήσεις στις αναζητήσεις μας.

Οπότε γενικότερα μπορούμε να συμπεράνουμε ότι όταν θέλουμε να είμαστε σίγουροι για την ακρίβεια των αντιγράφων (linearizability) χάνουμε χρόνο που κερδίζεται αν απλά μας ενδιαφέρει να βρούμε αντίγραφο, πράγμα που υλοποιείται για να έχουμε eventual consistency.

requests.txt

Το τελευταίο πείραμα είναι να τρέξουμε για k = 3 το αρχείο που έχει συνδυασμό τόσο inserts όσο και queries ώστε να δούμε κατά πόσο υπάρχει consistency των διαβασμάτων ωστε να δούμε κατά πόσο έχουμε ίδια αποτελέσματα. Έτσι παίρνουμε:





<u>Συμπεράσματα:</u>

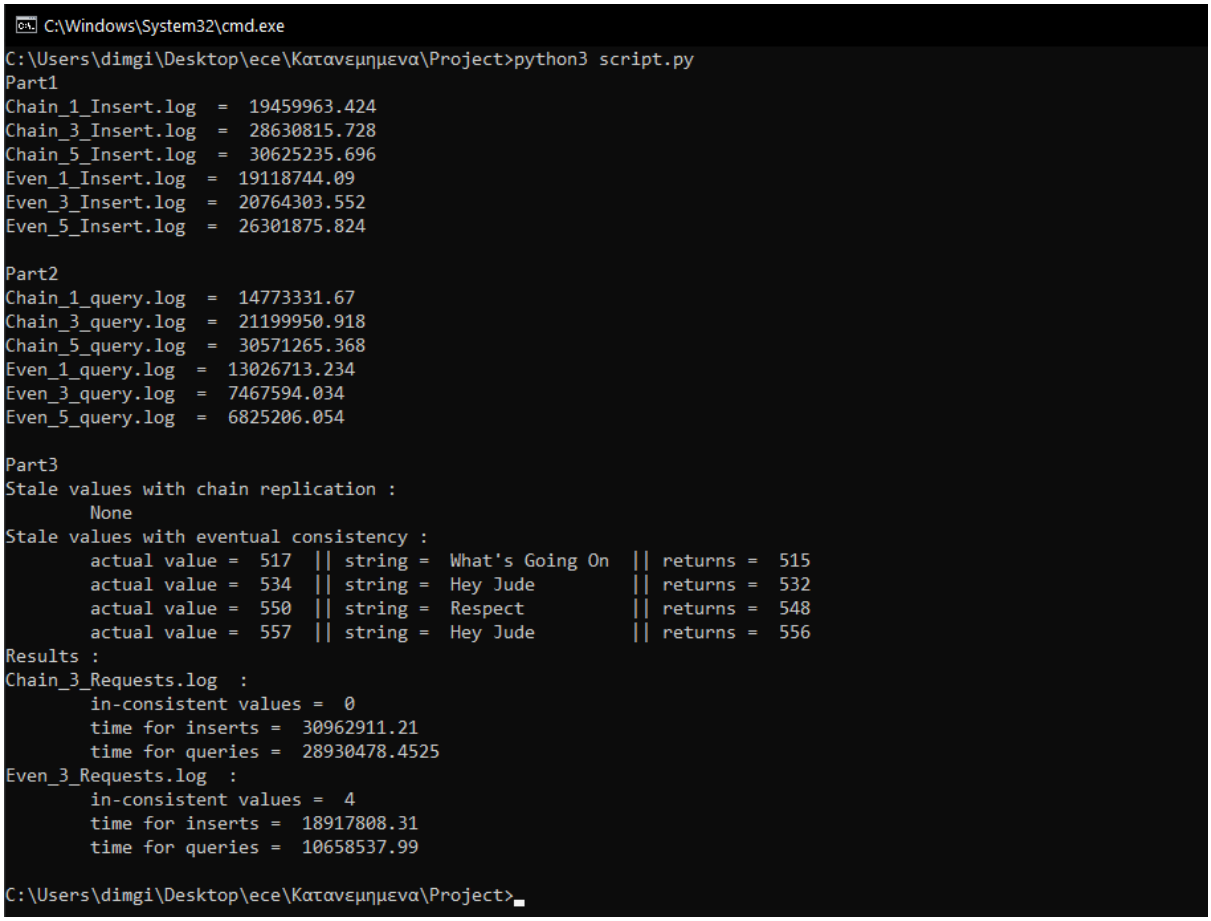
Έχουμε για το συγκεκριμένο πείραμα 2 γραφικές. Η 1η αφορά το χρόνο που χρειάστηκαν για να τρέξουν οι εντολές σε κάθε περίπτωση replication και η 2η τις stale τιμές που εμφανίζονται κατά την εκτέλεση.

Όσον αφορά τους χρόνους βλέπουμε τα ίδια αποτελέσματα που εξηγήσαμε και στα πρώτα δύο πειράματα που πραγματοποιήσαμε όσον αφορά τα 3 αντίγραφα. Αυτό που μπορούμε να παρατηρήσουμε επιπλέον όσων έχουμε ήδη πει είναι ότι είναι πιο γρήγορη η αναζήτηση από την εισαγωγή. Αυτό είναι λογικό δεδομένου ότι στην εισαγωγή μπορείς στην περίπτωση του linearizability να ξεκινήσεις από ενδιάμεσο κόμβο της αλυσίδας και άρα να πας πιο γρήγορα στο τέλος ενώ στην εισαγωγή πρέπει

πάντα να φτάσεις στον αρχικό κόμβο για να ξεκινήσει. Στην περίπτωση του eventual consistency μπορείς να πάρεις αποτέλεσμα από ενδιάμεσο κόμβο ενώ πάλι χρειάζεσαι την αρχή της αλυσίδας για να γίνει η εισαγωγή.
Όσον αφορά τις τιμές που διαβάζονται από τις αναζητήσεις μας μπορούμε να δούμε και από τα αποτελέσματα αλλά και μέσω ενός απλού parsing των δεδομένων ότι στην περίπτωση που έχουμε chain replication, μιας και περιμένουμε να γίνει εγγραφή σε όλη την αλυσίδα και μετά επιστρέφεται αποτέλεσμα, έχουμε πάντα τις πιο πρόσφατες τιμές στα δεδομένα μας. Από την άλλη όμως, όταν έχουμε eventual consistency δεδομένου ότι ο πρώτος κόμβος γυρίζει και εκκινεί την εγγραφή στους επόμενους παρατηρούμε σε κάποιες περιπτώσεις να έχουμε διαφορετική τιμή από την τελευταία που έχει γίνει insert στο Chord. Συγκεκριμένα στο παράδειγμα που τρέξαμε εμφανίζονται 4 stale τιμές κατά τα reads που σημαίνει ότι σε 4 περιπτώσεις διαβάσαμε την αμέσως προηγούμενη τιμή που είχε το συγκεκριμένο κλειδί. Αυτό γίνεται δεδομένου ότι το μήνυμα για την επόμενη εντολή που είναι το διάβασμα έρχεται προτού προλάβει να ανανεωθεί ολόκληρη η αλυσίδα με αποτέλεσμα να μη βλέπουμε τη νέα τιμή όπως θα θέλαμε. Αξίζει να αναφέρουμε πως αυτό είναι θέμα και τύχης μιας και όπως έχουμε ήδη εξηγήσει ο αρχικός κόμβος κάθε μηνύματος γίνεται με τυχαία επιλογή από τον bootstrap οπότε σημαίνει ότι μπορεί να πάμε είτε προς τον τελευταίο κόμβο που δεν έχει γίνει η ανανέωση, όντας άτυχοι, είτε προς τους αρχικούς κόμβους και να δούμε τις νέες τιμές. Γι' αυτό και δεν εμφανίζεται αυτό το "πρόβλημα" κάθε φορά που γίνεται κάποιο insert αλλά σε ορισμένες από τις περιπτώσεις.

Έξτρα

Στη συνέχεια θα παραθέσουμε τις τιμές όπως μας εμφανίστηκαν από το parsing των log αρχείων για να είναι πιο εύκολη η αντίληψη των αποτελεσμάτων. Να δούμε, δηλαδή, πιο εύκολα τα ακριβή νούμερα. Πρέπει να πούμε ότι όλες οι τιμές χρόνων είναι σε ns γι' αυτό και είναι τόσο μικρές και δεν είναι εξαιτίας καθυστέρησης του δικτύου για δρομολόγηση των διαφόρων αιτημάτων. Γενικότερα, δηλαδή, οι τιμές που βλέπουμε είναι της τάξης των λίγων ms απλά για να μην έχουμε θέματα με τιμές μικρότερες του ενός ms που υπήρχαν σε ορισμένες περιπτώσεις. Γι' αυτό και αποφασίσαμε να κοιτάξουμε τα ns τα οποία είναι εύκολα να τα δούμε με απλές εντολές της Java.
Part1 είναι τα inserts, Part2 τα queries και Part3 τα requests. Για να είναι πιο ευνόητα τα ακόλουθα αποτελέσματα, στα πρώτα δύο έχουμε εξόδους το μέσο χρόνο απόκρισης στο εκάστοτε πείραμα, ενώ στο τελευταίο αρχικά εμφανίζουμε τις περιπτώσεις stale τιμών στα reads, δείχνοντας τι τιμή εμφανίζεται σε αντίθεση με το ποιά θα έπρεπε να εμφανιστεί, και στο τέλος στα δύο αυτά πειράματα πόσες τέτοιες inconsistent εμφανίσεις έχουμε αλλά και τους χρόνους για το καθένα είδος request που βλέπουμε.


C:\Windows\System32\cmd.exe
C:\Users\dimgi\Desktop\ece\Κατανεμημενα\Project>python3 script.py
Part1
Chain_1_Insert.log = 19459963.424
Chain_3_Insert.log = 28630815.728
Chain_5_Insert.log = 30625235.696
Even_1_Insert.log = 19118744.09
Even_3_Insert.log = 20764303.552
Even_5_Insert.log = 26301875.824
Part2
Chain_1_query.log = 14773331.67
Chain_3_query.log = 21199950.918
Chain_5_query.log = 30571265.368
Even_1_query.log = 13026713.234
Even_3_query.log = 7467594.034
Even_5_query.log = 6825206.054
Part3
Stale values with chain replication :
None
Stale values with eventual consistency :
actual value = 517 || string = What's Going On || returns = 515
actual value = 534 || string = Hey Jude || returns = 532
actual value = 550 || string = Respect || returns = 548
actual value = 557 || string = Hey Jude || returns = 556
Results :
Chain_3_Requests.log :
in-consistent values = 0
time for inserts = 30962911.21
time for queries = 28930478.4525
Even_3_Requests.log :
in-consistent values = 4
time for inserts = 18917808.31
time for queries = 10658537.99
C:\Users\dimgi\Desktop\ece\Κατανεμημενα\Project>