ΚΑΤΑΝΕΜΗΜΕΝΑ ΣΥΣΤΗΜΑΤΑ
Εξαμηνιαία Εργασία - Project

Μετζάκης Ιωάννης	03116202	johnmetzakis@gmail.com
Σκούφης Πέτρος	03116141	pskoufis13@gmail.com

Εισαγωγή

Στα πλαίσια της παρούσας εργασίας καλούμαστε να υλοποιήσουμε μία απλοποιημένη έκδοση του Chord DHT, το ToyChord.

Το περιβάλλον εργασίας ήταν οι 5 VMs που μας παρείχε ο okeanos, στους οποίους εγκαταστήσαμε το XFCE GUI για ευκολότερη χρήση και ομορφότερη παρουσίαση.

Σχεδιαστικές επιλογές

Επιλέξαμε να υλοποιήσουμε την εργασία σε γλώσσα Python, χρησιμοποιώντας κατά κύριο λόγο τη βιβλιοθήκη Flask, η οποία μας επέτρεψε να υλοποιήσουμε όλες τις λειτουργίες του Chord σε αρκετά υψηλό επίπεδο, χωρίς να χρειαστεί να ασχοληθούμε απευθείας με sockets.

Αντιθέτως, η εφαρμογή μας λειτουργεί με ασύγχρονη λογική, όπου μπορεί να συναντήσει κανείς δύο τύπους κόμβων:

- Bootstrap κόμβος: **μοναδικός** κόμβος με αυξημένες ευθύνες στο δίκτυο. Έχει όλες τις λειτουργίες ενός απλού κόμβου αλλά επιπλέον γνωρίζει όλες τις απαραίτητες πληροφορίες, ώστε να διαχειρίζεται την είσοδο (join) και αποχώρηση (depart) κόμβων από το δίκτυο.
- Απλός κόμβος: εισέρχεται στο δίκτυο εκτελώντας αυτόματα την συνάρτηση join και αποχωρεί με την depart. Μπορεί να εισάγει νέα δεδομένα για αποθήκευση στο δίκτυο με χρήση της insert και να διαγράφει υπάρχοντα, ενώ επίσης μπορεί και να αναζητεί τις τιμές κλειδιών που υπάρχουν στο δίκτυο με χρήση της query.

Κάθε κόμβος αποτελεί ουσιαστικά και έναν server, ο οποίος “ακούει” σε μια συγκεκριμένη διεύθυνση, στην οποία μπορεί να δεχτεί αιτήματα από τους υπόλοιπους αλλά και από τρίτους (clients) σε συγκεκριμένα endpoints. Για λόγους ευκολίας δημιουργήσαμε και επιπλέον endpoints από αυτά που μας προέτρεπε η εκφώνηση με σκοπό να διαχειριστούμε μέσω αυτών την εσωτερική επικοινωνία των κόμβων στο δίκτυο και την εκτέλεση ειδικών λειτουργιών του ToyChord (π.χ. ανανέωση successor, διάδοση replicas, query “*”).

Η επιλογή μας για την υλοποίηση του Linearizability ήταν μέσω χρήσης Chain Replication, ακριβώς με τον τρόπο που αναφέρεται στην εκφώνηση.

Ειδική μνεία θέλουμε να γίνει στον τρόπο που επιλέξαμε να υλοποιήσουμε τη μη γραμμικότητα στο eventual consistency, δηλαδή το γεγονός πως η συνάρτηση που διαχειρίζεται τις κλήσεις στα endpoints “/insert” και “/delete” πρέπει να συνεχίζει τη λειτουργία της μετά την επιστροφή (return) της, πράγμα που αντίκειται στο συνηθισμένο τρόπο λειτουργίας της γλώσσας Python.

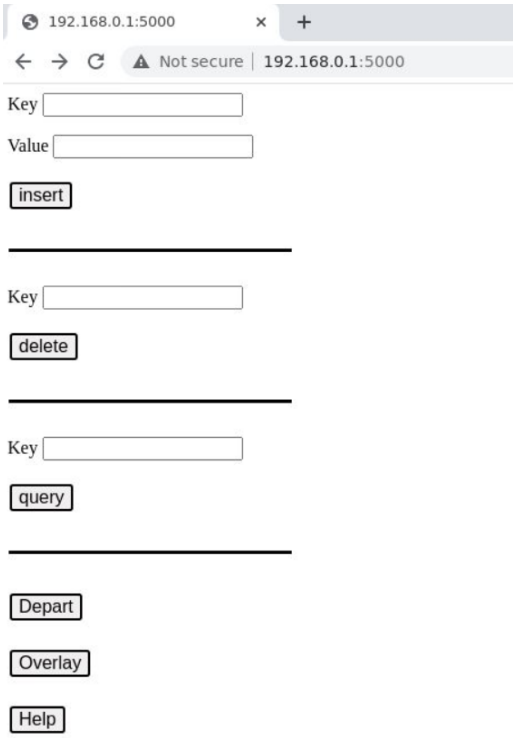
Για το λόγο αυτό χρησιμοποιήσαμε νήματα Threads, με σκοπό να διαχωρίσουμε τη διαδικασία της επιστροφής απάντησης από αυτή της ενημέρωσης των κόμβων replicas για την αλλαγή. Με το διαχωρισμό αυτό, η συνάρτηση που εξυπηρετεί το endpoint επιστρέφει άμεσα το αποτέλεσμα αμέσως μόλις γίνει η εγγραφή ή διαγραφή στον υπεύθυνο κόμβο και παράλληλα εκκινεί τα threads για την αποστολή αιτήματος ενημέρωσης στους κόμβους που έχουν τα replicas χωρίς όμως να χρειάζεται να περιμένει την επιστροφή αυτών των requests.

Σχετικά με την client διεπαφή με το χρήστη, επιλέξαμε αυτή να γίνεται άμεσα μέσω browser, χτυπώντας τη θύρα στην οποία τρέχει ο εκάστοτε κόμβος (GET request). Έχουμε αναπτύξει ένα βασικό κώδικα HTML, ο οποίος προβάλλει στον browser του χρήστη όλες τις δυνατότητες που έχει ως χρήστης του Chord. Σημειώνεται πως επιλέξαμε το join να γίνεται αυτόματα κατά την εκκίνηση ενός server. Παρακάτω παρουσιάζουμε τις δυνατότητες του html client που δημιουργήσαμε.

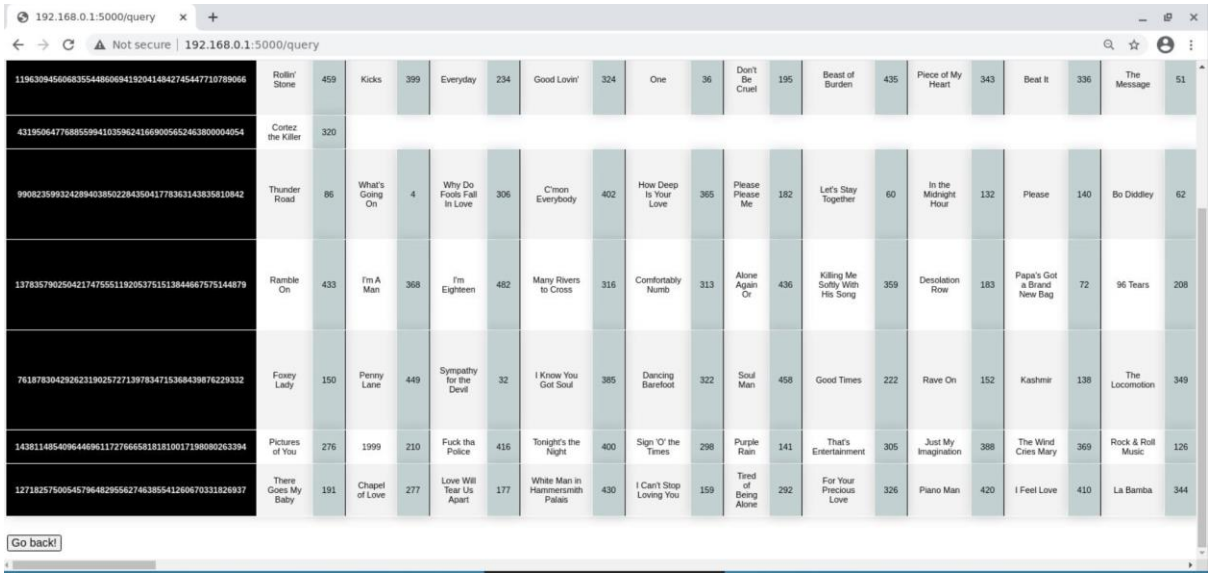
Παρουσίαση Λειτουργιών

Για την πρόσβασή του στον client του κάθε κόμβου αρκεί κανείς να προσπελάσει από τον browser του την διεύθυνση στην οποία “ακούει” ο εκάστοτε κόμβος. Τότε ο κόμβος αν δεν είναι ήδη μέρος του ToyChord ενεργοποιείται και στέλνει αίτημα για “**join**” στον bootstrap, τη διεύθυνση του οποίου γνωρίζει εξαρχής. Έπειτα αφού λάβει την απάντηση από τον bootstrap με τις πληροφορίες τις οποίες χρειάζεται για να αποτελεί μέρος του δικτύου, φορτώνει στον browser την παρακάτω html σελίδα **(Εικόνα 1)** που δίνει τις εξής επιλογές σε αναλογία με τις ζητούμενες λειτουργίες της εκφώνησης:

- **insert**: με τη συμπλήρωση των αντίστοιχων πεδίων “Key” και “Value” και το πάτημα του πλήκτρου insert ο χρήστης στέλνει ένα αίτημα στον κόμβο για εισαγωγή ενός νέου ζεύγους στο δίκτυο.
- **query**: με τη συμπλήρωση του πεδίου “Key” και το πάτημα του πλήκτρου query ο χρήστης στέλνει ένα αίτημα στον κόμβο για αναζήτηση της τιμής ενός κλειδιού στον κόμβο. Σε περίπτωση εύρεσής του επιστρέφει την τιμή (“Value”) του κλειδιού και τον κόμβο στον οποίο βρέθηκε, μέσω ανακατεύθυνσης σε ανάλογη σελίδα, ενώ αντιστοίχως λειτουργεί και στην περίπτωση μη εύρεσης.
  Ειδική μέριμνα έχει γίνει για το κλειδί “*”, για το οποίο η query επιστρέφει όλα τα ζεύγη <κλειδί,τιμή> που είναι αποθηκευμένα στο ToyChord ανά κόμβο, όπως φαίνεται στην παρακάτω εικόνα (**Εικόνα 2**).
- **delete**: με τη συμπλήρωση του πεδίου “Key” και το πάτημα του πλήκτρου delete ο χρήστης στέλνει ένα αίτημα στον κόμβο για διαγραφή ενός κλειδιού από το δίκτυο. Σε περίπτωση εύρεσής του το κλειδί διαγράφεται και ο χρήστης ανακατευθύνεται σε σελίδα με το ανάλογο μήνυμα, ενώ αντιστοίχως λειτουργεί και στην περίπτωση μη εύρεσης.
- **depart**: με το πάτημα του πλήκτρου depart ο κόμβος στέλνει αίτημα για αποχώρηση από το δίκτυο και εμφανίζει ανάλογο μήνυμα στην οθόνη του χρήστη.
- **overlay**: το πάτημα του πλήκτρου αυτού εμφανίζει στην οθόνη του χρήστη την τοπολογία του δικτύου, όπως φαίνεται στο στιγμιότυπο παρακάτω **(Εικόνα 3)**.
- **help**: εμφανίζεται στο χρήστη μία οθόνη όπου περιγράφονται οι λειτουργίες της εφαρμογής **(Εικόνα 4)**.



Εικόνα 1: Κεντρικό μενού του html client.



Εικόνα 2: Παράδειγμα της οθόνης απάντησης από την κλήση της query με όρισμα "".*

Connected nodes:

No	Node ID	IP Address
1.	203930399152259103538200775058720099807039272584	http://192.168.0.1:5000
2.	420355033397155728927117253942481438948818115541	http://192.168.0.5:5000
3.	431950647768855994103596241669005652463800004054	http://192.168.0.4:5000
4.	761878304292623190257271397834715368439876229332	http://192.168.0.3:5000
5.	990823599324289403850228435041778363143835810842	http://192.168.0.1:5001
6.	119630945606835544860694192041484274544771078906 6	http://192.168.0.4:5001
7.	127182575005457964829556274638554126067033182693 7	http://192.168.0.2:5001
8.	137835790250421747555119205375151384466757514487 9	http://192.168.0.5:5001
9.	138566631849213202482586715997670463325117422497 5	http://192.168.0.3:5001
10.	143811485409644696117276665818181001719808026339 4	http://192.168.0.2:5000

Go back!

Εικόνα 3: Στιγμιότυπο του αποτελέσματος της εκτέλεσης της "overlay" σε υπάρχον δίκτυο

Command	Attributes	Action
insert	key,value	Insert a (key,value) pair in the responsible node, where key is a song's name and value is a random string
delete	key	Delete the (key,value) pair from the responsible node, given the key
query	key	Search for given key, and return its value and the responsible node. If given key is "*", return all (key,value) pairs for every node
depart	-	Gracefully depart from the DHT Chord
overlay	-	Print network topology: The nodes of the chord are printed in order

Go back!

Εικόνα 4: Στιγμιότυπο της οθόνης "help".

Για όλες τις παραπάνω εντολές, πλην της depart, στην οθόνη απάντησης υπάρχει ένα πλήκτρο "Go Back!" που δίνει στο χρήστη τη δυνατότητα να επιστρέψει στο αρχικό μενού επιλογών.

Πειραματικό μέρος

Εισαγωγή - Αναμενόμενα Αποτελέσματα

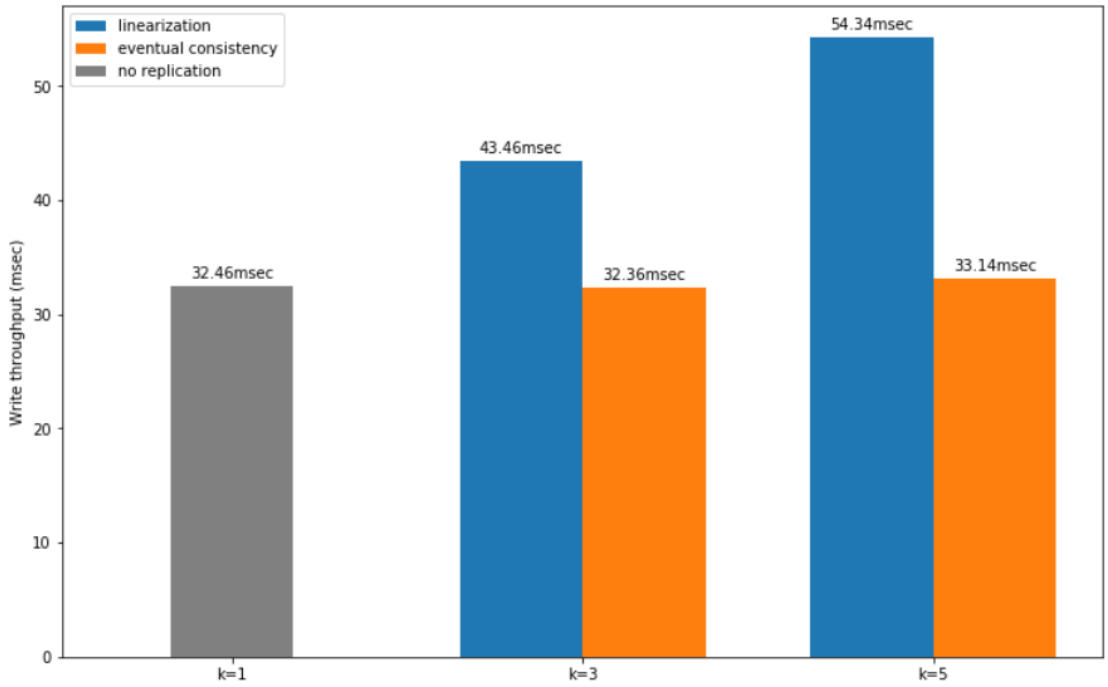
Στο πειραματικό αυτό μέρος της εργασίας έχουμε σκοπό να ελέγξουμε τη λειτουργία του δικτύου, ως προς την ταχύτητα εξυπηρέτησης αιτημάτων και την ορθότητα των απαντήσεων, όπως αυτή αντικατοπτρίζεται από την επιστροφή της πιο πρόσφατης τιμής κατά την ανάγνωση. Σε γενικές γραμμές αναμένουμε η “eventual consistency” να προσφέρει μεγαλύτερη ταχύτητα στο δίκτυο υπό το κόστος βέβαια κάποιων λανθασμένων- παλιών τιμών κατά την ανάγνωση, ως χαλαρή μορφή συνέπειας που είναι. Αντιθέτως, η “linearizability” ως αυστηρή μορφή συνέπειας, αναμένουμε να επιστρέφει πάντα τη σωστή - πιο πρόσφατη τιμή, επιβαρύνοντας όμως χρονικά τη λειτουργία του δικτύου.

Στην παρακάτω σειρά πειραμάτων αναδεικνύεται ο διαφορετικός τρόπος λειτουργίας του δικτύου με τα δύο πρωτόκολλα συνέπειας, τα πλεονεκτήματα και οι αδυναμίες τους.

1ο Πείραμα

Στο πείραμα αυτό, καλούμαστε να εισάγουμε σε ένα DHT δίκτυο με 10 κόμβους όλα τα κλειδιά που βρίσκονται στο αρχείο *insert.txt* για k=1 (χωρίς replication), καθώς και για k=3 και k=5, ελέγχοντας και τα δύο είδη συνέπειας (linearizability και eventual consistency), και να καταγράψουμε το write throughput του συστήματος.

Τα συγκεντρωτικά αποτελέσματα και από τα 5 πειράματα συνοψίζονται στο παρακάτω γράφημα:



όπου το write throughput προκύπτει από τη διαίρεση του συνολικού χρόνου εισαγωγής των κλειδιών με τον αριθμό τους, όπου στα συγκεκριμένα πειράματα είναι ίσος με 500.

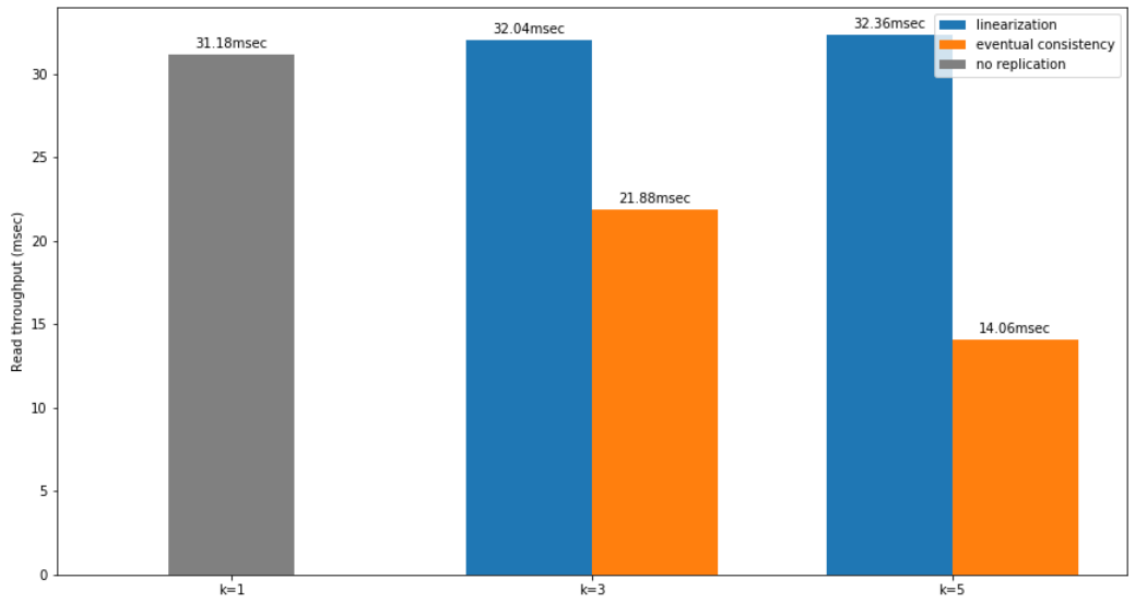
Όπως βλέπουμε, το write throughput για linearizability αυξάνεται σημαντικά και ανάλογα με τον αριθμό των replicas, γεγονός το οποίο οφείλεται στο ότι σε κάθε insert προστίθεται ο χρόνος εισαγωγής των απαιτούμενων κάθε φορά replicas, αφού κατά το chain replication ο τελευταίος κόμβος στη σειρά επιστρέφει το αποτέλεσμα του write.
Συγκεκριμένα, η διαφορά μεταξύ του write throughput για k=1 και k=3 είναι 11msec, ενώ η αντίστοιχη διαφορά για k=3 και k=5 είναι 10.98sec. Συνεπώς, παρατηρούμε μια αύξηση περίπου 5.5msec ανά k.

Αντιθέτως, στην περίπτωση του eventual consistency δεν παρατηρείται αύξηση (καθόλου έως ελάχιστη) του write throughput όσο αυξάνεται το k, καθώς το σύστημα μόλις κάνει insert ένα κλειδί επιστρέφει το αποτέλεσμα του write και στη συνέχεια αποστέλλει τα αιτήματα για εισαγωγή των replicas, συνεχίζοντας όμως παράλληλα την εισαγωγή νέων κλειδιών.

2ο Πείραμα

Στο πείραμα αυτό, καλούμαστε να κάνουμε την αντίστοιχη με την προηγούμενη διαδικασία και για τα 5 setups, αλλά αντί για εισαγωγή των κλειδιών από το αρχείο *insert.txt*, να διαβάσουμε τα κλειδιά που βρίσκονται στο αρχείο *query.txt* και να καταγράψουμε το read throughput του συστήματος.

Στο παρακάτω γράφημα βρίσκονται τα συγκεντρωτικά αποτελέσματα των 5 αυτών πειραμάτων:



όπου το read throughput προκύπτει αντίστοιχα με το write throughput.

Βλέπουμε πως το read throughput για linearizability παραμένει σχετικά σταθερό ανεξάρτητα με τον αριθμό των replicas, καθώς και και την ύπαρξή τους ή μη. Αυτό είναι λογικό, καθώς σε κάθε περίπτωση, όσα replicas και αν υπάρχουν, η read θα διαβάσει από τον τελευταίο στη σειρά κόμβο. Συνεπώς, η αναζήτηση από την query του τελευταίου αυτού κόμβου συμπεριφέρεται αντίστοιχα (χρονικά) με την αναζήτηση του υπεύθυνου κόμβου στην περίπτωση όπου δεν έχουμε replicas.

Από την άλλη, παρατηρούμε πως στην περίπτωση του eventual consistency το read throughput μειώνεται με την αύξηση των replicas, κάτι το οποίο και αναμέναμε, αφού η read διαβάζει από οποιονδήποτε κόμβο έχει αντίγραφο του κλειδιού.
Άρα, μεγαλύτερο k συνεπάγεται περισσότερους κόμβους με αντίγραφο, οπότε και λιγότερο χρόνο εύρεσης του κλειδιού.

Και για τα 2 αυτά πειράματα χρησιμοποιήσαμε το παρακάτω script για αποστολή αιτήματος σε τυχαίο κόμβο κάθε φορά (προφανώς με τις κατάλληλες μετατροπές για την περίπτωση της query) :

```python
from flask import Flask, render_template, request
import random
import requests
import time
#app = Flask(__name__)

if __name__ == '__main__':
    file1 = open('../input_files/insert.txt', 'r')
    Lines = file1.readlines()

    ips = ['http://192.168.0.1:5000', 'http://192.168.0.1:5001',
       'http://192.168.0.2:5000', 'http://192.168.0.2:5001',
       'http://192.168.0.3:5000', 'http://192.168.0.3:5001',
       'http://192.168.0.4:5000', 'http://192.168.0.4:5001',
       'http://192.168.0.5:5000', 'http://192.168.0.5:5001']

    start_time = time.time()
    for line in Lines:
        full = (line.strip().split(", "))
        key = full[0]
        value = full[1]
        ip = random.choice(ips)
        requests.post(str(ip+'/insert'), data = {'Key': key, 'Value': value})
    end_time = time.time()
    total_time = end_time - start_time
    print("TOTAL TIME: ", total_time)
```

Εικόνα 5: Αρχείο insert.py για μαζική αποστολή αιτημάτων

3ο Πείραμα

Στο πείραμα αυτό, καλούμαστε για το DHT με 10 κόμβους και k=3 να εκτελέσουμε τα requests του αρχείου *requests.txt* με σκοπό να σχολιάσουμε ποιο πρωτόκολλο replication, εκ των linearizability και eventual consistency, μας δίνει τις πιο fresh τιμές. Όπως και προηγουμένως επιλέξαμε να δημιουργήσουμε ένα απλό script, το οποίο εκτελεί μαζικά τα αιτήματα.

Για να έχει νόημα η σύγκριση των αποτελεσμάτων δημιουργήσαμε μία **τυχαία** μεν αλλά **κοινή** δε αλληλουχία από κόμβους, ώστε να αποτελούν τον εναρκτήριο κόμβο κάθε request. Επίσης για να εξάγουμε μία ground truth για να ελέγξουμε ποιο από τα δύο πρωτόκολλα έχει τις πιο “φρέσκες” τιμές, επαναλάβαμε το script μαζικής αποστολής των αιτημάτων του requests.txt αλλά αυτή τη φορά εισάγοντας μια χρονική καθυστέρηση μεταξύ της αποστολής του κάθε αιτήματος από το script, ώστε να δίνουμε στο δίκτυο χρόνο να διαδίδει με επιτυχία τις τιμές σε όλα τα replicas.

Παρατηρούμε πως στη συγκεκριμένη πειραματική διάταξη, τα αποτελέσματα της εκτέλεσης του script που αποστέλλει τα αιτήματα του requests.txt διαφέρουν μόνο σε μία απάντηση. Συγκεκριμένα, ελέγχοντας με την ground truth που παράξαμε με τον τρόπο που εξηγήσαμε παραπάνω παρατηρούμε πως τα πιο φρέσκα αποτελέσματα τα παίρνουμε με χρήση του πρωτοκόλλου συνέπειας “linearizability”, όπως είναι και αναμενόμενο από τη θεωρία. Ο πολύ μικρός αριθμός λανθασμένων - παλιών αναγνώσεων που επιστρέφεται στην περίπτωση του eventual consistency, οφείλεται ως ένα σημείο και στον τρόπο που επιλέξαμε να κάνουμε το πείραμα, δηλαδή στο γεγονός πως το script είναι γραμμικό και σε μεγάλο βαθμό η καθυστέρηση διάδοσης των replicas αντισταθμίζεται από την καθυστέρηση λήψης απάντησης από το script και υποβολής νέου αιτήματος.

Αν και δε ζητείται κάτι τέτοιο επιλέξαμε να επαναλάβουμε το παραπάνω πείραμα εισάγοντας μία τεχνητή καθυστέρηση με χρήση της **time.sleep()** μέσα στο σώμα της insert (αποκλειστικά όταν αυτή καλείται με σκοπό να γίνει insert κάποιο replica) με σκοπό να καταδείξουμε τον διαφορετικό τρόπο διαχείρισης των εγγραφών και των αναγνώσεων από τα δύο πρωτόκολλα. Καταφεύγουμε σε αυτή τη λύση, καθώς το script με το οποίο εκτελούμε τη μαζική αποστολή είναι γραμμικό με αποτέλεσμα να μην προλαβαίνει να προβάλει με μεγάλη ένταση τις αδυναμίες των πρωτοκόλλων. Η τεχνητή καθυστέρηση, λοιπόν, η οποία είναι ίδια και για τους δύο τύπους συνέπειας, προσομοιώνει ένα δίκτυο το οποίο δέχεται αιτήματα με συχνότητα τέτοια που υπερτερεί της συχνότητας και ταχύτητας διάδοσης των replicas σε αυτό. Για το συγκεκριμένο πείραμα παρέχουμε και τα στιγμιότυπα από την εκτέλεσή ενός απλού script που δημιουργήσαμε με σκοπό να συγκρίνει τις τιμές που επιστρέφονται από τις κλήσεις του script που στέλνει τα αιτήματα του requests.txt. Σαν αποτέλεσμα προβάλλονται οι γραμμές που ήταν διαφορετικές και το συνολικό πλήθος των διαφορετικών γραμμών.

```
user@snf-18470:~/Desktop/vm_scripts/input_scripts$ python compare.py result_event_3_late_005_1.txt result_linear_3_late_005_1.txt
('505\n', '506\n', '\n')
('510\n', '515\n', '\n')
('510\n', '515\n', '\n')
('513\n', '518\n', '\n')
('523\n', '524\n', '\n')
('530\n', '532\n', '\n')
('535\n', '539\n', '\n')
('544\n', '549\n', '\n')
('568\n', '570\n', '\n')
('573\n', '576\n', '\n')
('576\n', '579\n', '\n')
('587\n', '589\n', '\n')
('588\n', '591\n', '\n')
('591\n', '594\n', '\n')
('596\n', '597\n', '\n')
('597\n', '598\n', '\n')
16
user@snf-18470:~/Desktop/vm_scripts/input_scripts$
```

Εικόνα 6: Στιγμιότυπο από τη σύγκριση μεταξύ “eventual consistency” και “linearizability”

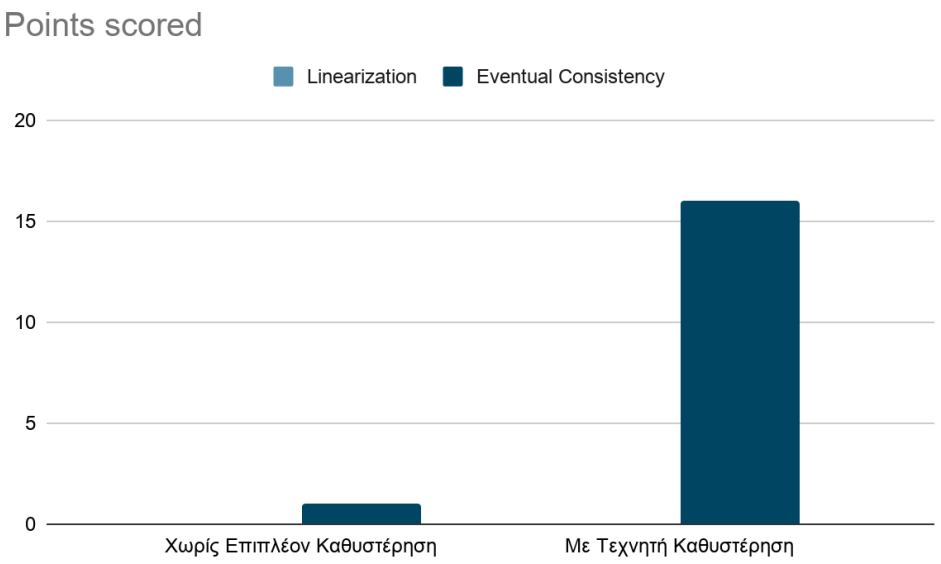
```
user@snf-18470:~/Desktop/vm_scripts/input_scripts$ python compare.py result_event_3_late_005_1.txt result_ground_3_late_005_1.txt
('505\n', '506\n', '\n')
('510\n', '515\n', '\n')
('510\n', '515\n', '\n')
('513\n', '518\n', '\n')
('523\n', '524\n', '\n')
('530\n', '532\n', '\n')
('535\n', '539\n', '\n')
('544\n', '549\n', '\n')
('568\n', '570\n', '\n')
('573\n', '576\n', '\n')
('576\n', '579\n', '\n')
('587\n', '589\n', '\n')
('588\n', '591\n', '\n')
('591\n', '594\n', '\n')
('596\n', '597\n', '\n')
('597\n', '598\n', '\n')
16
user@snf-18470:~/Desktop/vm_scripts/input_scripts$
```

Εικόνα 7: Στιγμιότυπο από τη σύγκριση μεταξύ “eventual consistency” και “ground truth”

```
user@snf-18470:~/Desktop/vm_scripts/input_scripts$ python compare.py result_linear_3_late_005_1.txt result_ground_3_late_005_1.txt
0
user@snf-18470:~/Desktop/vm_scripts/input_scripts$
```

Εικόνα 8: Στιγμιότυπο από τη σύγκριση μεταξύ “ground truth” και “linearizability”

Στο παρακάτω γράφημα φαίνονται οι διαφορές του κάθε είδους replication σε σχέση με την ground truth:



Στο συγκεκριμένο πείραμα φαίνεται ο διαφορετικός τρόπος λειτουργίας των δύο μορφών συνέπειας. Η “eventual consistency” αν και αυξάνει την ταχύτητα απόκρισης παράγει αρκετές (16 για την ακρίβεια) λανθασμένες - παλιές απαντήσεις. Αντιθέτως, η “linearizability” όπως ήταν και αναμενόμενο, καταφέρνει, επιβαρύνοντας βέβαια το δίκτυο με ένα μικρό χρονικό κόστος, να επιστρέφει πάντοτε την τελευταία - πιο νέα τιμή που αντιστοιχεί σε κάθε κλειδί.

Συνοψίζοντας, στη σειρά αυτή πειραμάτων τα αποτελέσματά μας συμβαδίζουν απόλυτα με τα αναμενόμενα από τη θεωρία αποτελέσματα. Το γεγονός αυτό πιστοποιεί σε μεγάλο βαθμό την ορθή υλοποίηση και λειτουργία του δικτύου ToyChord από την ομάδα μας.