Εθνικο Μετσοβειο Πολυτεχνειο

Κατανεμημενα Συστηματα

Αναφορα Εξαμηνιαιας Εργασιας

ToyChord

Ιωάννης Στεργιου (03115039)
Γιώργος Χιονας (03115132)

26 Μαρτίου 2021

Περιεχόμενα

1 Εισαγωγή

Ο σκοπός της συγκεκριμένης εργασίας είναι η εισαγωγή στα κατανεμημένα συστήματα υλοποιώντας μια απλοποιημένη εκδοχή του πρωτοκόλλου Chord, το ToyChord. Η κύρια απλούστευση έγκειται στο ότι η δική μας εκδοχή του πρωτοκόλλου δεν περιέχει finger tables για την δρομολόγηση, αλλά ο κάθε κόμβος θα περιέχει δείκτες για τον προηγούμενο και επόμενο κόμβο του. Η απλούστευση αυτή επηρεάζει την αποδοτικότητα του πρωτοκόλλου καθώς η εύρεση του κατάλληλου κομβου, η οποία είναι μια απο τις πιο σημαντικές διαδικασίες, ανέρχεται σε $O(n)$, οπου n ο αριθμός των κόμβων. Στην περίπτωση όπου χρησιμοποιούνται finger tables η παραπάνω διαδικασία γίνεται σε $O(logn)$.

Το πρωτόκολλο ToyChord, κατα τα άλλα, υλοποιεί τις υπόλοιπες λειτουργίες του Chord. Επιπλέον για το συγκεκριμένο κατανεμημένο συστήματα ζητήθηκε η υλοποιήση 2 ειδών συνέπειας με linearizability(chain replication) και με eventual consistency. Η επίδρασή των δύο αυτών ειδών συνέπειας θα αναλυθεί παρακάτω.

2 Πρωτόκολλο ToyChord

2.1 Περιγραφή του πρωτοκόλλου

Οι λειτουργίες του πρωτοκόλλου ειναι η εισαγωγή κλειδιού, αναζήτηση κλειδιού, διαγραφή κλειδιού, εκτύπωση των συνδεδεμένων κόμβων και διαγραφή συγεκριμένου κόμβου απο το δίκτυο. Επισημαίνεται ότι ένας κόμβος μπορεί να αποχωρήσει μόνο γραςεφυλλψ, καθώς το ToyChord δεν διαχειρίζεται απρόσμενες αποχωρήσεις κόμβων (node failures).

Το πρωτόκολλο αυτό επιτρέπει τη δημιουργία ενός κατανεμημένου συστήματος, ορίζοντας μία τοπολογία κόμβων σε ένα δακτύλιο, στην οποία κάθε κόμβος μπορεί να επικοινωνεί με τον προηγούμενο και τον επόμενο κόμβο. Επιπλέον το πρωτόκολλο ορίζει την θέση που αντιστοιχεί σε κάθε κόμβο, χρησιμοποιώντας Consistent Hashing. Αυτό το είδος Hashing εξασφαλίζει πολλά καλά ποιοτικά χαρακτηριστικά για τη ορθή λειτουργία του συστήματος. Τέλος, το πρωτόκολλο ορίζει τους απαραίτητους μηχανισμούς, ώστε το δίκτυο να λειτουργεί ορθά και αποδοτικά, ακόμη κι αν αποχωρήσουν ή εισέλθουν κόμβοι στο δίκτυο.

Μία ιδιότητα που είναι επιθυμητό να εξασφαλίζει μία κατανεμημένη τοπολογία είναι η διασφάλιση της συνέπειας των δεδομένων. Στην παρούσα εργασία εξετάστηκαν δύο είδη συνέπειας, συνέπεια μέσω linearizability και συνέπεια

μέσω eventual consistency.

3 Υποδομή για την εργασία

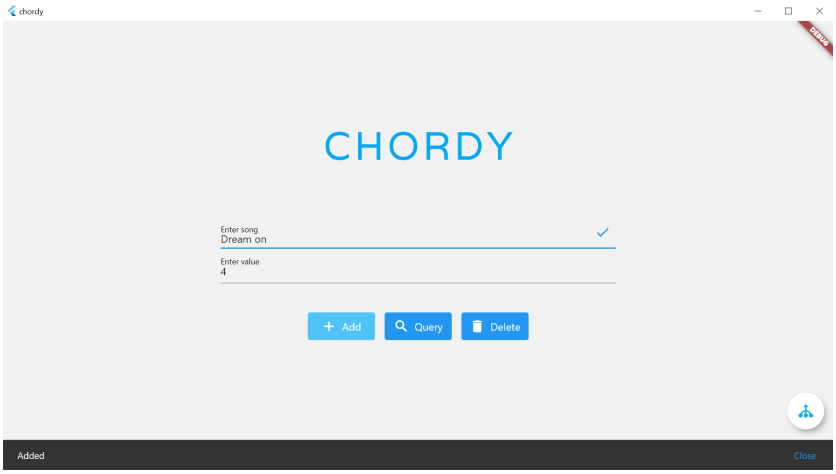
Για την διεξαγωγή των πειραμάτων και προκείμενου να μελετήσουμε τα χαρακτηριστικά του πρωτοκόλλου σε πραγματικές συνθήκες, μας παραχωρήθηκαν 5 εικονικές μηχανές στον Okeanos. Αξιοποιώντας τους πόρους δημιουργήσαμε 5 σερερς με λειτουργικό σύστημα Ubuntu Server LTS και σε καθέναν παραχωρήσαμε 2 πυρήνες, 2GB μνήμης και GB δίσκου. Στη συνέχεια, δημιουργήσαμε ένα τοπικό δίκτυο μεταξύ των 5 εικονικών μηχανών, ώστε να μπορεί να υπάρχει επικοινωνία μεταξύ τους.

3.1 Τεχνολογίες που χρησιμοποιήθηκαν

Σε ότι αφορά την ανάπτυξη του σερερ, αποφασίσαμε να χρησιμοποιήσουμε ως γλώσσα προγραμματισμού την Python, για το λόγο ότι προσφέρεται για γρηγόρη ανάπτυξη εφαρμογών αλλά και γιατί προσφέρει ποικιλία ώριμων πακέτων για ανάπτυξη server εφαρμογών. Για την επικοινωνία μεταξύ των διαφορετικών κόμβων χρησιμοποήθηκε το framework grpc. Το gRPC είναι ένα συγχρόνο rpc framework, το οποίο επιτρέπει με αποδοτικό τρόπο επικοινωνία μεταξύ υπολογιστών, ενώ για την κωδικοποίηση των μήνυματων που στέλνονται μεταξυ των κόμβων χρησιμοποιήθηκαν τα protocol buffers, τα οποία προσφέρουν απλή σύνταξη για να ορίσει κανείς τα μηνύματα και τις υπηρεσίες ενός gRPC server. Ένα βασικό πλεονέκτημα του gRPC και κύριος λόγος που το επιλέξαμε είναι ότι μεταδίδει δεδομένα σε δυαδική μορφή και κατά κύριο λόγο επιτυγχάνει καλύτερες επιδόσεις από αυτές που προσφέρει ένα απλό rest api.

Σχετικά με τον client, για την ανάπτυξη της cli εφαρμογής, η οποία υλοποιήθηκε και αυτή σε γλώσσα Python χρησιμοποιήθηκε το framework cement, το οποίο πρόκειται για ένα πλούσιο σε λειτουργικότητες προιόν για την καλύτερη και γρηγορότερη ανάπυξη εφαρμογών cli. Συγκεκριμένα, προσφέρει εύκολη σύνταξη αλλά και τη δυνατότητα ενσωμάτωσης άλλων γνωστών πακέτων για ανάπτυξη cli εφαρμογών σε Python, όπως είναι τα πακέτα argparse, tabulate, colorlog, κλπ.

Για τη δημιουργία ενός πιο ολοκληρωμένου client, αναπτύχθηκε επίσης μία εφαρμογή desktop για windows, καθώς και μία εφαρμογή για κινητά (android & ios), χρησιμοποιώντας το flutter framework σε γλώσσα dart. Παρακάτω, παράτιθονται εικόνες της εφαρμογής αυτής σε λειτουργία.



Σχήμα 1: Στιγμιότυπο κατά την εισαγωγή κλειδιού μέσω της client εφαρμογής σε flutter

4 Πειράματα

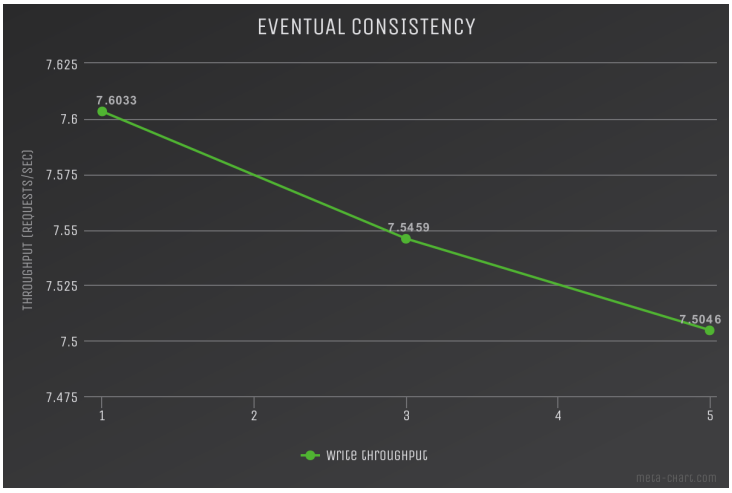
Χρησιμοποιώντας τα μηχανήματα που μας παραχωρήθηκαν και τρέχοντας την δικιά μας υλοποίηση του Chord, συλλέξαμε μετρικές όπως ο χρόνος εκτέλεσης, το read throughput, και το write throughput για διαφορετικές τιμές του replication factor και για 2 διαφορετικά είδη consistency (Linearizability με chain replication και Eventual consistency).

4.1 Αποτελέσματα

4.1.1 Eventual Consistency

Για την περίπτωση που επιλέγουμε ως είδος συνέπειας το Eventual consistency, εκτελέσαμε πειράματα για 3 αρχικές τιμές. Η μεταβλητή που αλλάζει σε κάθε διαφορετικό πείραμα ειναι το replication factor, δηλαδή ο αριθμός των αντιγράφων που πρέπει να υπάρχει για κάθε ζεύγος (key,value). Το Eventual consistency ειναι λιγότερο αυσηρή μορφή συνέπειας σε σχέση με τη Linearizability και αναμένεται να φανεί και στις χρόνους των προσομοιώσεων. Παρακάτω, παρουσιάζουμε σε μορφή πίνακα τα αποτελέσματα που λάβαμε.

Οπως αναφέρεται και στην εκφώνηση οι αλλαγές διαδίδονται lazily, δηλαδή αφου βρεθεί ο primary κόμβος επιστρέφει την απάντηση του write και στη συνεχεια διαδίδεται η πληροφορία και στους replica managers. Αυτό σημάινει



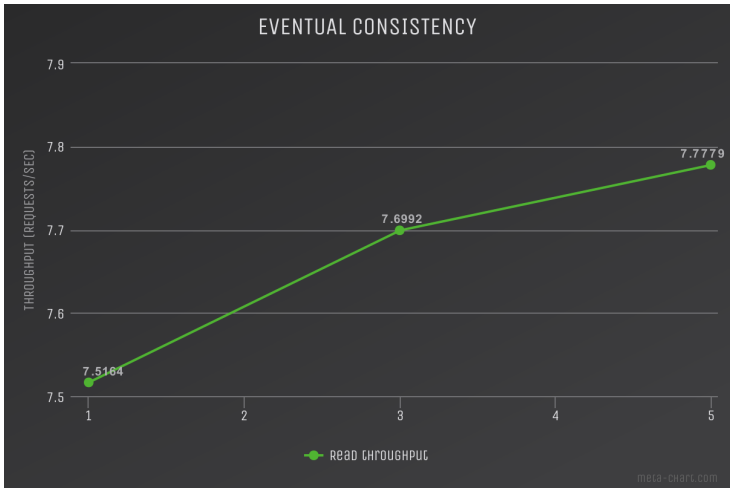
Σχήμα 2: Write throughput with eventual consistency

k	Χρόνος(s)	Write throughput
1	65.7604	7.6033
3	66.2609	7.5459
5	66.6252	7.5046

Πίνακας 1: Write throughput κατά την εισαγωγή 500 κλειδιών.

ότι η επόμενη εισαγωγή δεν χρειάζεται να περιμένει έως ότου διαδοθεί η πληροφορία σε ολους τους replica managers. Με βάση αυτήν την πρόταση ειναι αναμενόμενο να μην διαφερεί αρκετα ο χρόνος που απαιτείται για την εισαγωγή όλων των στοιχείων ανεξαρτήτος του replication factor. Σημειώνεται ότι για να επιτευχθεί αυτό χρησιμοποιήθηκε πολυνηματικός προγραμματισμός. Αξίζει να τονίσουμε επίσης, ότι στα αποτελέσματα των χρόνων που δίνονται επιδρά και η τυχαιότητα. Η τυχαιότητα σχετίζεται με τον τυχαίο κόμβο απο τον οποίο γίνεται το αντίστοιχο αίτημα και με τον αποτέλεσμα της cryptographic hash function, δηλαδή της sha1.Ωστόσο γνωρίζουμε ότι και οι δύο κατανομές προσεγγίζουν την ομοιόμορφη. Με βάση αυτήν την παρατήρηση θα ήταν πιο χρήσιμο να τρέχουμε την κάθε προσομοίωση αρκετές φορές ώστε να εξετάσουμε την σύγκληση.

Οσον αφορά το χρόνο διαβάσματος των 500 requests που δίνονται, στην περίπτωση του Eventual constistency αναμένουμε ο χρόνος να μειώνεται όσο αυξάνεται το replication factor. Αύτο περιμένουμε να συμβεί διότι το διάβα-



Σχήμα 3: Read throughput with eventual consistency

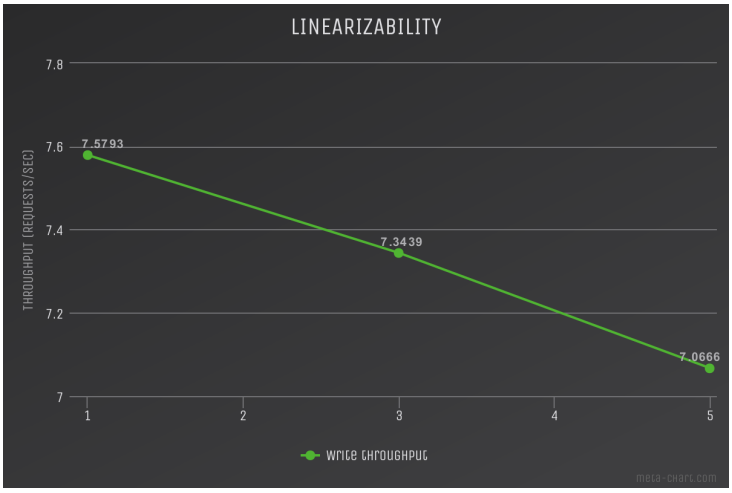
k	Χρόνος	Read throughput
1	66.5208	7.5164
3	64.9419	7.6992
5	64.2842	7.7779

Πίνακας 2: Read throughput κατά την αναζήτηση 500 κλειδιών.

σμα με το συγκεκριμένο είδος συνέπειας, επιστρέφει την τιμή απο οποιοδήποτε replica manager συναντήσει πρώτο. Οσο αυξάνεται το replica factor αυξάνεται και η πιθανότητα να βρεθεί κάποιος κόμβος στο δακτύλιο που περιέχει το κλείδι συντομότερα. Από την άλλη πλευρά αξίζει να σημειωθεί ότι όσο αυξάνεται το replica factor αυξάνεται και το expected μέγεθος της κάθε βάσης. Παρόλα αυτά ο τελευταίος παραγοντας δεν αντισταθμίζει τον πρώτο.

4.1.2 Linearizability

Για την περίπτωση που επιλέξαμε ως είδος συνέπειας Linearizability και συγκεκριμένα chain replication, εκτελέσαμε πειράματα για 3 διαφορετικές τιμές του replication factor κατά την εισαγωγή κλειδιών και κατά την εύρεση κλειδιών στο σύστημα του Chord. Πιο συγκεκριμένα, ελέγξαμε το χρόνο εκτέλεσης για $k = 1$, $k = 3$, $k = 5$ για καθένα από τα δύο σενάρια. Παρακάτω, παρουσιάζουμε σε μορφή πίνακα τα αποτελέσματα που λάβαμε.

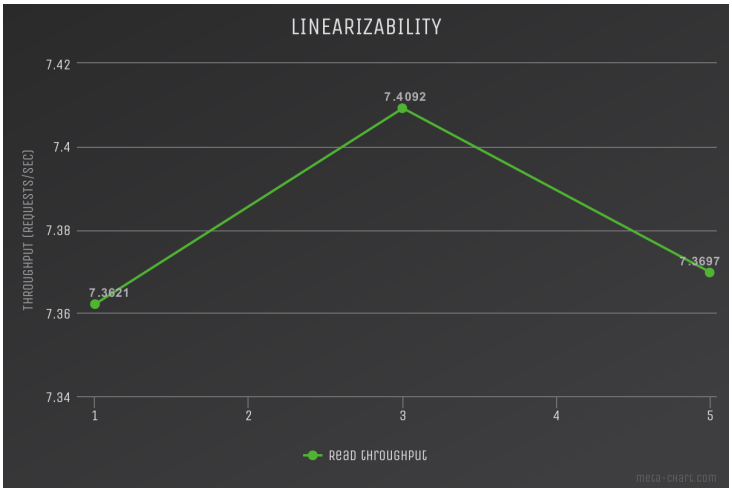


Σχήμα 4: Write throughput with chain replication

k	Χρόνος(s)	Write throughput
1	65.9689	7.5793
3	68.0835	7.3439
5	70.7551	7.0666

Πίνακας 3: Write throughput κατά την εισαγωγή 500 κλειδιών.

Παρατηρούμε ότι καθώς αυξάνεται το replication factor, τόσο αυξάνεται και το write throughput, που έχουμε. Αυτό είναι αναμενόμενο, αφού κατά την εισαγωγή ενός κλειδιού όταν έχουμε ως είδος συνέπειας linearizabilty με chain replication, τότε πρέπει να λάβουμε την απάντηση από τον τελευταίο, πιο fresh replica manager. Επομένως, ο χρόνος εισαγωγής πρέπει να είναι ανάλογος του αριθμού των replicas, δηλαδή του k.



Σχήμα 5: Read throughput with eventual consistency

k	Χρόνος(s)	Read throughput
1	67.9149	7.3621
3	67.4833	7.4092
5	67.8452	7.3697

Πίνακας 4: Read throughput κατά την αναζήτηση 500 κλειδιών.

Σε αυτό το πείραμα, οι τιμές που παρατηρούμε βλέπουμε ότι επηρεάζονται σε μικρό βαθμό από την τιμή του k. Αυτό εξηγείται, διότι σε αυτή την περίπτωση, αναζητάται πάντοτε ένας συγκριμένος κόμβος, αυτός που ειναι τελευταίος στην αλυσίδα. Λόγω αυτού, ο χρόνος δεν εξαρτάται απο το replication factor. Αξίζει να σημειωθεί ότι στην υλοποίηση μας, φροντίσαμε ώστε για να βρεθεί ο κόμβος που ειναι τελευταίος στην αλυσίδα των replicas, να μην απαιτείται πρώτα να φτάσουμε στον primary replica manager, κάνοντας έτσι έναν παραπάνω κύκλο στο δίκτυο. Αλλά εξασφαλίζουμε ότι θα ακολουθηθεί το συντομότερο μονοπάτι από τον κόμβο, που λαμβάνει το αίτημα μέχρι τον τελευταίο replica manager, που είναι υπεύθυνος για την συγκεκριμένη τιμή.

4.2 Queries και Inserts

Στην τελυταία προσομοίωση που μας ζητείται, έχουμε ένα αρχειο το οποίο περιέχει εισαγωγές κλειδιών και αναζητήσεις τραγουδιών. Καταγράψαμε τα αποτελέσματα και για τα δύο είδη συνέπειας που έχουμε υλοποιήσει. Η διάταξη

του δικτύου ειναι ίδια όπως και στα προηγούμενα πειράματα, δηλαδή έχουμε 10 κόμβους και $replicationfactor = 3$. Από τη συγκεκριμένη προσομοίωση γίνονται πιο ξεκάθαρες διαφορές των δύο ειδών συνέπειας. Συγκεκριμένα, το πρώτο query για ένα κλειδί που μόλις προστέθηκε δεν δίνει την πιο fresh τιμή στην περίπτωση του eventual consistency. Αυτό δεν συμβαίνει ποτέ όταν χρησιμοποιείται linearizability. Τα αποτελέσματα των δύο αυτών προσομοιώσεων δίνονται σε δύο αρχεια results_eventual.txt και results_linearizability.txt. Οι χρόνοι που χρειάστηκαν για να ολοκληρωθούν τα δύο αρχεία ειναι 64.6s και 67.8s αντίστοιχα. Όπως αναμέναμε ο συνολικος χρόνος εκτέλεσης αυξάνεται στην περίπτωση του linearizability καθώς πρόκειται για πιο αυστηρό είδος συνέπειας, το οποίο έχει αιτιολογηθεί και παραπάνω.

5 Κατακλείδα

Στην παρούσα εργασία μελετήσαμε σε βάθος τις αρχές και τις προκλήσεις υλοποίησης μίας κατανεμημένης εφαρμογής και εμβαθύναμε περαιτέρω στο πρωτόκολλο Chord. Κάναμε πλήρη υλοποίηση του πρωτοκόλλου Toychord σε γλώσσα Python και διεξάγαμε σείρα πειραμάτων για την μέτρηση των επιδόσεων του πρωτοκόλλου και για το αντίκτυπο που προκαλούν τα διαφορετικά είδη συνέπειας. Παρατηρήσαμε διαφορές στα αποτελέσματα μας, μεταξύ των δύο ειδών συνέπειας που μελετήθηκαν, οι οποίες ωστόσο θα ήταν πιο ευδιάκριτες, σε ένα μεγαλύτερης κλίμακας δίκτυο (περισσότεροι κόμβοι).