ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Κατανεμημένα Συστήματα

Χειμερινό Εξάμηνο 2020-21

Εξαμηνιαία Εργασία: Υλοποίηση ενός Chord Πρωτοκόλλου για P2P Δίκτυα βασισμένο σε Distributed Hash Tables

Ομάδα Πύραυλος

Ιωάννα Τάσου (03116055)

Δημήτρης Γαλάνης (03116088)

Δανάη-Νικολέτα Χαρίτου (03116045)

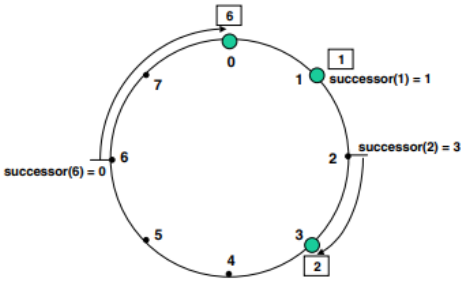
Περιγραφή

Σκοπός αυτής της εργασίας είναι η σχεδίαση του ToyChord, μιας απλοποιημένης έκδοσης του Chord πρωτοκόλλου για P2P δίκτυα [1]. Το Chord βασίζεται στα *Distributed Hash Tables (DHTs)* τα οποία αποτελούν μια κατανεμημένη μέθοδο παροχής υπηρεσιών αναζήτησης, παρόμοια με εκείνη που λαμβάνει χώρα σε ένα απλό Hash Table. Με αυτόν τον τρόπο, το πρωτόκολλο μπορεί να εξυπηρετεί αποδοτικά τον μοναδικό του *operation*: την αντιστοίχιση κλειδιού σε κόμβο του δικτύου, και ισοδύναμα την αντιστοίχιση δεδομένου σε κόμβο που το αποθηκεύει.

Στην δικιά μας (απλοποιημένη) υλοποίηση του Chord, υποστηρίζονται οι λειτουργίες *join, depart, insert, delete, query*. Ακόμη στο ToyChord υπάρχει ένας *bootstrap* κόμβος ο οποίος, πέραν της βασικής λειτουργίας της απεικόνισης κλειδιού σε κόμβο, παρέχει κεντρική υποστήριξη για τις λειτουργίες *join* και *depart* στους υπόλοιπους κόμβους του δικτύου, οι οποίες στο κανονικό πρωτόκολλο εξυπηρετούνταν κατανεμημένα από περιοδικούς ελέγχους σε γειτονικούς κόμβους. Τέλος, το ToyChord υποστηρίζει πολλαπλά αντίγραφα των δεδομένων (*replication),* τα οποία μένουν σταθερά στο πλήθος για κάθε graceful εισαγωγή/έξοδο κόμβου από το δίκτυο βάσει του κεντρικού συντονισμού από τον *bootstrap* κόμβο.

Αρχιτεκτονική Chord Δικτύου

Σε ένα P2P δίκτυο που ακολουθεί το Chord πρωτόκολλο, οι κόμβοι βρίσκονται διατεταγμένοι σε έναν νοητό δακτύλιο, γνωρίζοντας ο καθένας μόνο τις διευθύνσεις των 2 γειτονικών τους κόμβων. Στον κάθε χρήστη, κατά την είσοδό του στο δίκτυο, ανατίθεται ένας κόμβος (και το αντίστοιχο node_id) και όλα τα δεδομένα με κλειδιά ανάμεσα στον νεοεισαχθέν κόμβο και τον αμέσως τοπολογικά προηγούμενό του, θεωρώντας κάποια συγκεκριμένη φορά. Στο ToyChord, το mapping χρηστών σε κόμβους και δεδομένων σε κλειδιά γίνεται μέσω της SHA1.



Εικόνα 1: Τοπολογία ενός Chord P2P Δικτύου. (Πηγή: [1]).

Στην συνέχεια, διατυπώνουμε βασικές παρατηρήσεις σχετικά με την λειτουργία των μεθόδων του ToyChord πρωτοκόλλου.

- Οι μέθοδοι κόμβου που κάνουν *insert/depart* πρέπει να ενημερώνουν τους γείτονές τους και τον *bootstrap* κόμβο αλλά και για την διατήρηση της τοπολογίας των αποθηκευμένων δεδομένων στο δίκτυο.
- Οι μέθοδοι *insert/query/delete* στέλνουν μηνύματα στους γείτονες τους τα οποία προωθούνται κυκλικά στον δακτύλιο μέχρι την εύρεση του κατάλληλου κόμβου και στην συνέχεια προωθούνται μέχρι την εύρεση του κόμβου του αρχικού αποστολέα. Το δεύτερο σκέλος της προώθησης είναι αναγκαίο, αφού στο δίκτυο κάθε κόμβος δεν γνωρίζει περισσότερες ip από όσες χρειάζεται για την εύρυθμη λειτουργία του πρωτοκόλλου.

Πολιτικές Συνέπειας (Consistency Policy)

Όπως αναφέραμε, το ToyChord υποστηρίζει replication των δεδομένων. Τα δεδομένα του κάθε κόμβου του δικτύου αντιγράφονται και στους προηγούμενούς του δημιουργώντας έτσι για κάθε κλειδί μια συνεχόμενη λίστα κόμβων που το αποθηκεύουν (*replica list*).

Για τα replicas υποστηρίζονται 2 πολιτικές συνέπειας: *eventual consistency,* και *linearizability* η οποία υλοποιείται μέσω chain replication. Κατά την *linearizability* παρέχονται εγγυήσεις πως η τιμή που διαβάζεται είναι η πιο πρόσφατα εγγεγραμμένη, αφού το γράψιμο ενός κλειδιού ολοκληρώνεται μόνο αφού έχει ολοκληρωθεί η εγγραφή και για τον τελευταίο κόμβο του *replica list* για το κλειδί αυτό, ενώ η ανάγνωση γίνεται πάντα από τον τελευταίο κόμβο του *replica list*. Αντίθετα, στο *eventual consistency*, οι αναγνώσεις μπορούν να γίνουν από οποιαδήποτε κόμβο του *replica list,* και οι εγγραφές ολοκληρώνονται αφού έχουν ενημερώσει μόνο τον *replica manager* που αντιστοιχεί στο κλειδί αυτό και όχι όλους τους κόμβους της *replica list.*

Προφανώς, η ύπαρξη replicas περιπλέκει σημαντικά τις λειτουργίες *insert/delete/query/join/depart* του ToyChord πρωτοκόλλου, αφού σύμφωνα με αυτό και για *replication factor k*, πρέπει κάθε δεδομένο να υπάρχει ως αντίγραφο στους (k-1)-ακριβώς-προηγούμενους κόμβους -όταν αυτό είναι δυνατό από το πλήθος των κόμβων στο δίκτυο. Στην υλοποίησή μας φροντίζουμε έτσι ώστε να εξασφαλίζεται πάντα η προηγούμενη συνθήκη.

Δομή Υλοποίησης

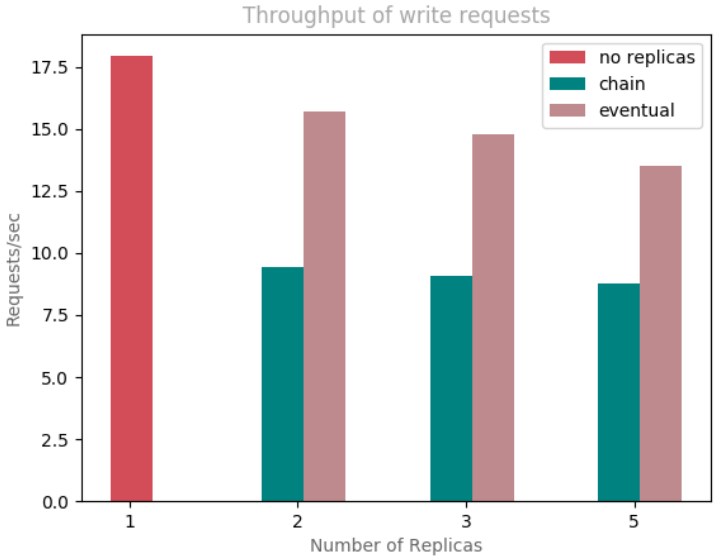
Στην δική μας υλοποίηση του ToyChord, κάθε κόμβος αποτελεί ταυτόχρονα έναν *server* και έναν *client* και επικοινωνεί με τους υπόλοιπους μέσω REST API requests. Για την κατασκευή του Web API χρησιμοποιήθηκε το web-framework Flask.
Επίσης κατασκευάσαμε, μέσω της βιβλιοθήκης Click, ένα εύχρηστο *Command Line Interface (CLI)*, που διευκολύνει την πρόσβαση στις λειτουργίες των κόμβων από τον χρήστη.

Μετρήσεις

Με στόχο την καλύτερη μελέτη της επίδρασης της χρήσης replicas στο ToyChord αλλά και την σύγκριση των τεχνικών αντιγραφής διεξάγουμε πειράματα στα οποία μετράμε τους χρόνους εισαγωγών και διαγραφών στο δίκτυο σε διάφορες περιπτώσεις αλλά ελέγχουμε και τις τιμές που επιστρέφονται. Η προσομοίωση γίνεται σε 10 κόμβους ενός private δικτύου υπολογιστικών πόρων (2 CPU cores, 2GB RAM) που μας έχουν διατεθεί (για τα πλαίσια του μαθήματος) απο τον okeanos.
Κάθε υπολογιστής εισέρχεται στο δίκτυο ως 2 διαφορετικοί κόμβοι, χρησιμοποιώντας προφανώς διαφορετικό port.
Πιο συγκεκριμένα προβαίνουμε στα εξής πειράματα:

❖ *Μετρήσεις throughput εγγραφής*

Σε ένα DHT με 10 κόμβους εισάγουμε όλα τα κλειδιά που βρίσκονται στο αρχείο *insert.txt*, που μας δόθηκε, χρησιμοποιώντας k=1 (χωρίς replication), k=3 και k=5 και με *linearizability* και *eventual consistency* και καταγράφουμε το write throughput του συστήματος (πόσο χρόνο πήρε η εισαγωγή των κλειδιών προς τον αριθμό τους). Τα inserts θα ξεκινούν κάθε φορά από τυχαίο κόμβο του συστήματος. Τα αποτελέσματα που προκύπτουν φαίνονται στο παρακάτω barplot:

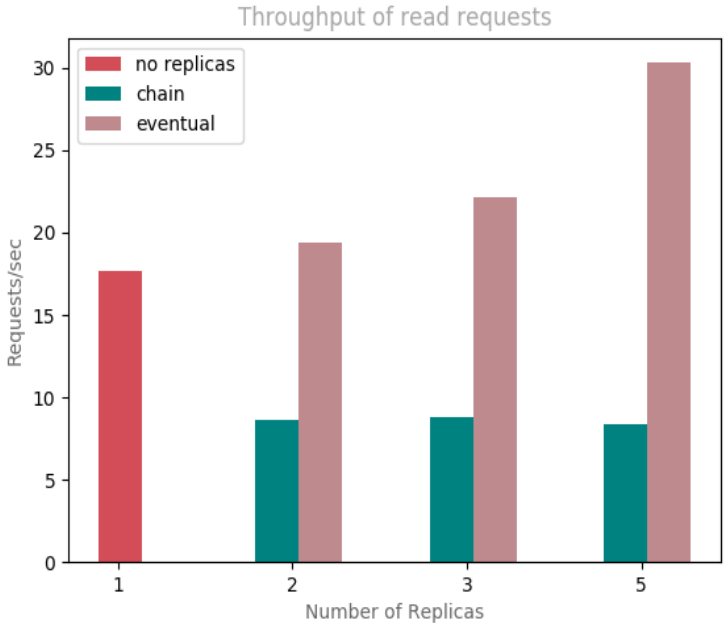


Εικόνα 2: Throughput request εγγραφής για διαφορετικά πλήθη replicas και πολιτικές consistency .

Παρατηρούμε, γενικά ότι οι χρόνοι που χρειάζονται για την ολοκλήρωση των writes στην περίπτωση που δεν χρησιμοποιούνται καθόλου replicas (k=1) είναι προφανώς μικρότεροι (μεγαλύτερο throughput) σε σύγκριση με την χρήση replicas, καθώς η τελευταία απαιτεί και ενημέρωση όλων των αντιγράφων (linearizability) ή έναρξη αυτής της ενημέρωσης (eventual). Συγκρίνοντας ακόμη την απόδοση του linearizability με την απόδοση του eventual consistency, είναι εμφανές ότι το eventual consistency παρουσιάζει αρκετά υψηλότερο write-throughput. Απαιτείται, δηλαδή, σημαντικά μικρότερος χρόνος για τα write σε σύγκριση με το chain replication, πράγμα το οποίο αναμέναμε καθώς στο chain replication περιμένουμε την ενημέρωση όλων των αντιγράφων πριν επιστρέψουμε το αποτέλεσμα του write request ενώ στο eventual consistency η ενημέρωση των αντιγράφων γίνεται ασύγχρονα αφού επιστραφεί το αποτέλεσμα του write request από τον replica manager του αντίστοιχου κλειδιού. Τέλος, όσο αυξάνεται ο αριθμός των replicas που χρησιμοποιούνται υπάρχει προφανώς μείωση του throughput καθώς απαιτείται ενημέρωση περισσότερων αντιγράφων.

❖ *Μετρήσεις throughput ανάγνωσης*

Ομοίως εκτελούμε τα δοσμένα request ανάγνωσης από το αρχείο *queries.txt*, που μας δόθηκε για όλες τις περιπτώσεις που περιγράφηκαν στο παραπάνω πείραμα και παίρνουμε τα παρακάτω αποτελέσματα:



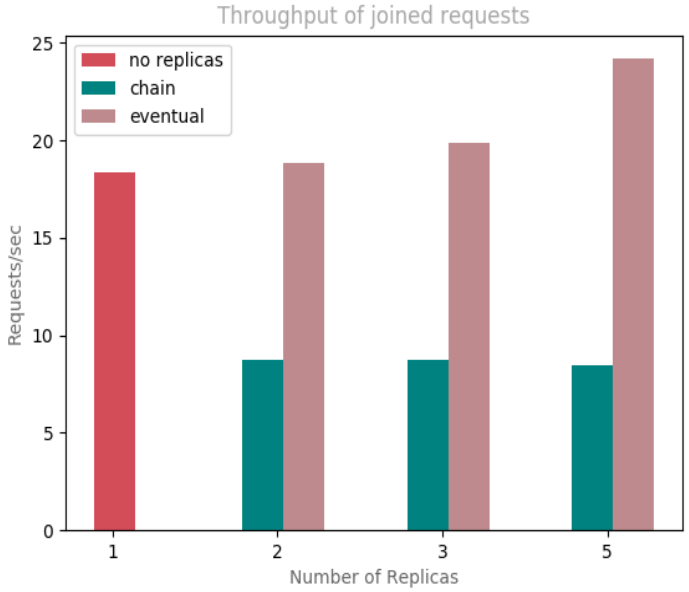
Εικόνα 3: Throughput request ανάγνωσης για διαφορετικά πλήθη replicas και πολιτικές consistency.

Στην περίπτωση που δεν χρησιμοποιούνται καθόλου replicas για να βρεθεί η τιμή του κλειδιού που αναζητούμε πρέπει αναγκαστικά να βρεθεί και ο κόμβος που αποθηκεύει αυτό το κλειδί καθώς μόνο αυτός περιέχει την τιμή του.
Αντίθετα στην περίπτωση του *eventual consistency* η πληροφορία που ψάχνουμε εντοπίζεται είτε στα replica είτε στον replica manager και έτσι είναι πιθανότερο να εντοπίσουμε γρηγορότερα το value του επιθυμητού κλειδιού. Το *eventual consistency* παρουσιάζει, έτσι, υψηλότερο read-throughput (μικρότερους χρόνους) σε σύγκριση με την μη χρήση replicas.
Στην περίπτωση όμως του *linearizability* παρατηρούμε σημαντικά μικρότερα read-throughputs (υψηλότερους χρόνους) στα queries, καθώς αναζητείται πάντα το τελευταίο replica για να επιστρέφεται η πιο fresh τιμή. Έτσι το αίτημα χρειάζεται να διανύσει μεγαλύτερη απόσταση στο δίκτυο για την εύρεση της επιστρεφόμενης τιμής.

Όσον αφορά το πλήθος των αντιγράφων, στην περίπτωση του *eventual consistency* παρατηρούμε ότι η αύξηση των αντιγράφων οδηγεί και σε αύξηση της απόδοσης καθώς η πληροφορία που αναζητείται υπάρχει σε περισσότερους κόμβους στο δίκτυο και συνεπώς βρίσκουμε την ζητούμενη τιμή γρηγορότερα. Στην περίπτωση του *linearizability* παρατηρούμε ελάχιστη αύξηση του απαιτούμενου χρόνου με την αύξηση του k, το οποίο είναι λογικό καθώς αυξάνεται η απόσταση που πρέπει να διανύσει το αίτημα ανάλογα με τον εκάστοτε αριθμό των replicas.

❖ *Μετρήσεις throughput μεικτών requests*

Τέλος εκτελέσαμε τα μεικτά requests που δίνονται στο αρχείο *requests.txt*, που μας δόθηκε για τις ίδιες τιμές με τα παραπάνω πειράματα και παίρνουμε τα εξής αποτελέσματα:



Εικόνα 4: Throughput μεικτών request ανάγνωσης και εγγραφής για διαφορετικά πλήθη replicas και πολιτικές consistency.

Επίσης ελέγξαμε τα αρχεία των αποτελεσμάτων του *eventual consistency* για stale τιμές και παρατηρήσαμε την ύπαρξη stale τιμών στα αποτελέσματα του *eventual consistency* όπως φαίνεται και στις παρακάτω φωτογραφίες (Εικ. 5-6):

```
{'status': 'Success', 'msg': 'Successful insertion of data (Like a Rolling Stone, 583)'}
{'status': 'Success', 'msg': 'Successful query result for key Hey Jude is value: 581'}
{'status': 'Success', 'msg': 'Successful query result for key Like a Rolling Stone is value: 582'}
{'status': 'Success', 'msg': 'Successful query result for key Hey Jude is value: 581'}
{'status': 'Success', 'msg': 'Successful query result for key Hey Jude is value: 581'}
{'status': 'Success', 'msg': 'Successful insertion of data (Respect, 584)'}
{'status': 'Success', 'msg': 'Successful query result for key Hey Jude is value: 581'}
{'status': 'Success', 'msg': 'Successful query result for key Like a Rolling Stone is value: 583'}

{'status': 'Success', 'msg': 'Successful insertion of data (Like a Rolling Stone, 583)'}
{'status': 'Success', 'msg': 'Successful query result for key Hey Jude is value: 581'}
{'status': 'Success', 'msg': 'Successful query result for key Like a Rolling Stone is value: 583'}
{'status': 'Success', 'msg': 'Successful query result for key Hey Jude is value: 581'}
{'status': 'Success', 'msg': 'Successful query result for key Hey Jude is value: 581'}
{'status': 'Success', 'msg': 'Successful insertion of data (Respect, 584)'}
{'status': 'Success', 'msg': 'Successful query result for key Hey Jude is value: 581'}
{'status': 'Success', 'msg': 'Successful query result for key Like a Rolling Stone is value: 583'}
```

Εικόνα 5: Αποτελέσματα μεικτών request σε Chord P2P δίκτυο με χρήση replicas για διαφορετικές πολιτικές consistency: (πάνω) *Eventual Consistency*, (κάτω) *linearizability* μέσω *chain replication*. Τα αποτελέσματα βρίσκονται στην γραμμή 411 των αποτελεσμάτων.

```
{'status': 'Success', 'msg': 'Successful query result for key Hey Jude is value: 591'}
{'status': 'Success', 'msg': 'Successful query result for key Hey Jude is value: 591'}
{'status': 'Success', 'msg': 'Successful insertion of data (Hey Jude, 594)'}
{'status': 'Success', 'msg': "Successful query result for key What's Going On is value: 592"}
{'status': 'Success', 'msg': 'Successful query result for key Hey Jude is value: 591'}
{'status': 'Success', 'msg': 'Successful query result for key Like a Rolling Stone is value: 593'}
{'status': 'Success', 'msg': 'Successful query result for key Hey Jude is value: 594'}

{'status': 'Success', 'msg': 'Successful query result for key Hey Jude is value: 591'}
{'status': 'Success', 'msg': 'Successful query result for key Hey Jude is value: 591'}
{'status': 'Success', 'msg': 'Successful insertion of data (Hey Jude, 594)'}
{'status': 'Success', 'msg': "Successful query result for key What's Going On is value: 592"}
{'status': 'Success', 'msg': 'Successful query result for key Hey Jude is value: 594'}
{'status': 'Success', 'msg': 'Successful query result for key Like a Rolling Stone is value: 593'}
{'status': 'Success', 'msg': 'Successful query result for key Hey Jude is value: 594'}
```

Εικόνα 6: Αποτελέσματα μεικτών request σε Chord P2P δίκτυο με χρήση replicas για διαφορετικές πολιτικές consistency: (πάνω) *Eventual Consistency*, (κάτω) *linearizability* μέσω *chain replication*. Τα αποτελέσματα βρίσκονται στην γραμμή 464 των αποτελεσμάτων.

Παραθέτουμε ενδεικτικά τις εικόνες των αντίστοιχων requests αλλά αυτή την φορά με την χρήση του chain replication (*linearizability*) και βλέπουμε ότι αυτή την φορά δεν επιστρέφονται stale τιμές αλλά οι πιο πρόσφατες που έχουν εισαχθεί στο δίκτυο, αφού τα insert requests επιστρέφουν αποτέλεσμα αφού ανανεωθούν οι τιμές όλων των αντιγράφων και τα queries λαμβάνουν τιμή μόνο από το τελευταίο αντίγραφο. Εξασφαλίζεται έτσι το consistency των τιμών που επιστρέφονται από το δίκτυο σε όλα τα requests αλλά αυτό κοστίζει σε χρόνο.

Αναφορές

[1] Stoica, Ion, et al. "Chord: A scalable peer-to-peer lookup service for internet applications." ACM SIGCOMM Computer Communication Review 31.4 (2001): 149-160.