Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Εξαμηνιαία εργασία στα Κατανεμημένα Συστήματα

Ιωάννης Ντάλλας, 03111418, ynts@outlook.com

***Σύνοψη* – Η παρούσα εργασία έχει ως στόχο την σχεδίαση και υλοποίηση ενός DHT – απλοποιημένης έκδοσης του Chord. Στη συνέχεια εισάγονται δεδομένα στο δίκτυο και εκτελούνται queries ώστε να βρεθεί η απόδοση του συστήματος για διάφορες τιμές των παραμέτρων του.**

A. Περιγραφή της υλοποίησης.

Το σύστημα έχει υλοποιηθεί με τη μορφή ενός προσομοιωτή σε Java. Κάθε κόμβος του δικτύου είναι ένα process του λειτουργικού συστήματος που μπορεί να επικοινωνεί μέσω sockets με του δυο γειτονικούς κόμβους επάνω στον λογικό δακτύλιο. Κάθε κόμβος τρέχει και από ένα ξεχωριστό thread όπου ικανοποιεί τα αιτήματα που δέχεται. Η επικοινωνία του χρήστη με το σύστημα γίνεται επίσης με ένα thread που ελέγχει τις εντολές και τις εισάγει στο δίκτυο.

α. Λειτουργίες του συστήματος.

1. Εισαγωγή καινούριων κόμβων με την εντολή ADD:ID, όπου ID είναι ένας μοναδικός ακέραιος αριθμός. Το μήνυμα προωθείται από τον αρχικό κόμβο προς τους επόμενους μέχρι να βρεθεί η σωστή θέση σύμφωνα με το hash του ID. Χρησιμοποιήθηκε SHA1 για την εξαγωγή του hash. Στη συνέχεια στέλνονται εντολές για τον διαμοιρασμό των αρχείων που ανήκουν πλέον στον καινούριο κόμβο.
2. Διαγραφή ενός κόμβου με την εντολή REMOVE:ID. Ο τρόπος διαγραφής είναι όμοιος με τον τρόπο εισαγωγής ενός κόμβου. Η διαγραφή γίνεται με gracefull τρόπο μοιράζοντας ξανά τα αρχεία του κόμβου σε αυτούς που μένουν στο δίκτυο και διορθώνοντας τα αντίγραφα.
3. Εισαγωγή ενός στοιχείου με την εντολή INSERT:KEY:VALUE. Το μήνυμα προωθείται και όταν βρεθεί ο κατάλληλος κόμβος εισάγεται σε αυτόν το αρχείο. Ο κόμβος επιλέγεται πάλι σύμφωνα με το hash του και το hash του κλειδιού του αρχείου. Στη συνέχεια κατασκευάζονται αντίγραφα του αρχείου και εισάγωνται στους επόμενους k – 1 κόμβους του λογικού δακτυλίου, όπου k είναι το replication factor.
4. Με τον ίδιο τρόπο λειτουργεί και η διαγραφή αρχείου DELETE:KEY:VALUE.
5. Η εύρεση μιας καταχώρησης γίνεται με την εντολή QUERY:KEY, η οποία ξεκινά από τυχαίο κόμβο του δικτύου και επιστρέφει το πρώτο αρχείο που θα βρεί, είτε κανονική καταχώρηση είτε αντίγραφο. Η εντολή με τη μορφή QUERY:* επιστρέφει όλες τις καταχωρήσεις του δικτύου.

6. Η εντολή PRINT επιστρέφει όλες τις καταχωρήσεις του δικτύου ξεκινώντας πάντα από τον πρώτο κόμβου που εισήχθει και η DUMP τυπώνει τους κόμβους μαζί με τις πληροφορίες σύνδεσης (hash, sockets).
7. Τέλος, η εντολή KILL τερματίζει όλες τις διεργασίες.

β. Τρόπος επικοινωνίας των κόμβων.

Όπως περιγράψαμε, σε κάθε κόμβο αντιστοιχεί και ένα thread το οποίο αναμένει αιτήματα προς εξυπηρέτηση. Ας περιγράψουμε ένα σενάριο μιας τέτοιας εξυπηρέτησης. Έστω ότι ο κόμβος 1 θέλει να στείλει μία εντολή εισαγωγής αρχείου στον κόμβο 2. Αρχικά ο κόμβος 1 θα καλέσει την συνάρτηση `sendRequest(socket_2, request)`. Ο δεύτερος κόμβος που περιμένει αιτήματα θα λάβει το αίτημα και θα στείλει ένα μήνυμα τύπου “DONE” στον 1 ώστε να τερματιστεί η λειτουργία της συνάρτησης. Στη συνέχεια θα κατακερματίσει το μήνυμα και θα καλέσει την συνάρτηση που αντιστοιχεί στην εντολή INSERT με ορίσματα τις παραμέτρους του μηνύματος. Η εντολή μπορεί να ολοκληρωθεί σε αυτόν τον κόμβο ή να προωθηθεί στον επόμενο. Όταν ολοκληρωθεί και εφόσον έχει ζητηθεί θα επιστραφεί μήνυμα “DONE_INSERT” στον κόμβο που ξεκίνησε τη διαδικασία, ο οποίος θα το στείλει στη συνέχεια στη main συνάρτηση του προγράμματος.

γ. Consistency

Όνταν ένας κόμβος λαμβάνει μήνυμα εισαγωγής ή διαγραφής μιας καταχώρησης θα πρέπει στη συνέχεια να εισάγει ή να διαγράψει και τα αντίστοιχα αντίγραφα. Όταν στέλνεται ένα τέτοιο μήνυμα το κύριο πρόγραμμα περιμένει μια απάντηση τύπου “DONE_INSERT” ή “DONE_DELETE”. Η επιλογή του consistency αφορά στο πότε θα σταλεί αυτό το μήνυμα.

Όταν έχουμε linear consistency στέλνεται αφού ενημερωθούν και όλα τα αντίγραφα, ενώ στην περίπτωση του eventual consistency στέλνεται αμέσως μετά την εκτέλεση της λειτουργίας για την βασική καταχώρηση και τα αντίγραφα ενημερώνονται lazily στη συνέχεια, με κίνδυνο να διαβαστούν μη έγκυρες τιμές. Στην εργασία το linear consistency έχει υλοποιηθεί με chain replication.

Β. Πειραματικές μετρήσεις.

Εκτελούμε το πρόγραμμα για linear και eventual consistency με τιμές του replication factor k=1, k=3 και k=5. Θα μετρήσουμε το χρόνο εκτέλεσης της εισαγωγής 500 στοιχείων στους κόμβους, της απάντησης σε 500 ερωτήματα και σε ακόμη 500 συνδιασμούς εισαγωγών και ερωτημάτων. Τα αποτελέσματα φαίνονται στον παρακάτω πίνακα.

Linear Consistency

	Insert	Query	Insert/Query
k=1	5004ms	2930ms	2765ms
k=3	5250ms	2595ms	2959ms
k=5	6661ms	2131ms	2930ms

Eventual Consistency

	Insert	Query	Insert/Query
k=1	4881ms	2938ms	2768ms
k=3	4562ms	2538ms	1306ms
k=5	4049ms	2037ms	1210ms

Αρχικά παρατηρούμε ότι και στις δύο περιπτώσεις consistency ο χρόνος εκτέλεσης των ερωτημάτων μειώνεται για μεγαλύτερες τιμές του k. Αυτό συμβαίνει διότι έχουμε περισσότερες καταχωρήσεις του ίδιου στοιχείου και συνεπώς χρειάζονται λιγότερες επαναπροωθήσεις μηνυμάτων μεταξύ των κόμβων. Για linear consistency ο χρόνος εισαγωγής στοιχείων αυξάνεται για μεγαλύτερο k, αφού το πρόγραμμα θα πρέπει να περιμένει και την δημιουργία των αντιγράφων για να συνεχίσει με τις επόμενες εντολές, πράγμα που δε συμβαίνει στην περίπτωση του eventual. Επιπλέον, βλέπουμε ότι στην περίπτωση των Insert/Query για eventual consistency η μείωση στο χρόνο είναι ακόμη μεγαλύτερη σε σχέση με το linear εφόσον οι εντολές εισαγωγής δεν χρειάζονται περισσότερο χρόνο όπως αναφέραμε.

Ωστόσο υπάρχει ενα σημαντικό μειονέκτημα για τα Insert/Query της δεύτερης περίπτωσης. Επειδή το κύριο πρόγραμμα δε θα περιμένει να ολοκληρωθεί η εισαγωγή των αντιγράφων υπάρχει πάντα ο κίνδυνος να διαβαστούν παρωχημένες τιμές. Αυτό συνέβει όντως στα πειράματα που εκτελέσθηκαν, όπως φαίνεται από το παρακάτω τμήμα των αρχείων καταγραφής που παρατίθενται μαζί με τον κώδικα και την αναφορά.

```
Adding data...Hey Jude, 579
Inserting replica of (Hey Jude, 579) in node 8
Forwarding to next socket: 40009 with counter = 3
Entry was found at node with ID 4, Hash: 1b6453892473a467d07372d45eb05abc2031647a

    [R] Key: Hey Jude, Hash: cf6f0c00413d9164b36cc83f8c54421227c2765e, Value: 576

Inserting replica of (Hey Jude, 579) in node 9
Forwarding to next socket: 40004 with counter = 2
Inserting replica of (Hey Jude, 579) in node 4
Forwarding to next socket: 40001 with counter = 1
Inserting replica of (Hey Jude, 579) in node 1
```

Όπως βλέπουμε εισάγεται η καταχώρηση *Hey Jude, 579* και στη συνέχεια γίνεται αναζήτηση με το ίδιο κλειδί και επειδή διαβάζεται ένα αντίγραφο που δεν έχει ενημερωθεί, επιστρέφεται μια stale καταχώρηση *Hey Jude, 576*. Στη συνέχεια βλέπουμε ότι ενημερώνεται και αυτή η καταχώρηση αντιγράφου αλλά είναι ήδη αργά και η άκυρη τιμή έχει αποσταλεί.

Συνεπώς η επιλογή του consistency θα πρέπει να γίνει με βάση την ανοχή που θέλουμε να έχουμε σε τέτοια σφάλματα και ελέγχοντας αν το κέρδος που παρατηρήσαμε στους πίνακες αξίζει την εισαγωγή αυτών των σφαλμάτων.