Κοντογιάννης Αλέξανδρος 03115048
Σταματελόπουλος Νικόλαος 03116138

Κατανεμημένα Συστήματα
Εξαμηνιαίο Πρότζεκτ (toyChord DHT)

Εισαγωγή:

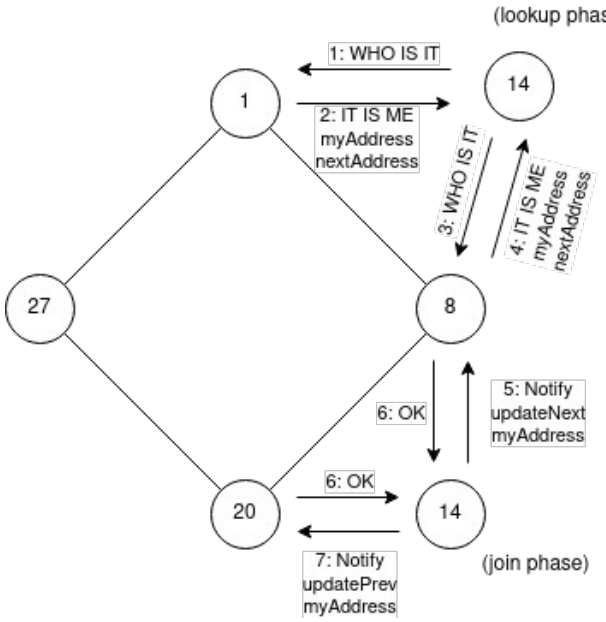
Στην άσκηση αυτή κληθήκαμε να υλοποιήσουμε μία απλή μορφή του Chord DHT όπου κάθε κόμβος που συμμετέχει στο δίκτυο γνωρίζει μόνο τον επόμενο κόμβο (και τον προηγούμενο για κάποιες λειτουργίες που αυτό είναι απαραίτητο) στο δίκτυο, ενώ όλα τα αιτήματα δρομολογούνται πάντα προς μία κατεύθυνση. Οι κόμβοι για τη μεταξύ του επικοινωνία χρησιμοποιούν TCP/IP sockets και επιτρέπουν σε ένα χρήστη από οποιοδήποτε κόμβο να εκτελέσει εισαγωγή ή αναζήτηση των key-value ζευγών που αποθηκεύονται στο δίκτυο (key: τίτλος ενός αρχείου, value: θεωρούμε την τοποθεσία του, αλλά αφού δεν έχουμε κάποιο υπαρκτό αρχείο και οι δύο τιμές είναι απλά strings). Άλλες λειτουργίες που υλοποιήθηκαν είναι προφανώς join/depart των κόμβων, με το πρώτο να γίνεται αυτόματα κατά την εκκίνησή του, μια overlay λειτουργία που παρουσιάζει στο χρήστη με τη σειρά όλους τους συνδεδεμένους κόμβους στο δίκτυο αλλά και μία πρόσθετη printAll εντολή που τυπώνει όλα τα ζεύγη που είναι αποθηκευμένα στον κόμβο που την εκτελεί (κυρίως για debugging λόγους αλλά προτιμήθηκε να μην αφαιρεθεί για λόγους ευκολίας και κατά την εξέταση της εργασίας).
Τέλος, υλοποιήθηκε replication των δεδομένων τόσο με linearzability (chain replication) όσο και eventual consistency όπως περιγράφεται στην εκφώνηση της εργασίας. Και στις δύο περιπτώσεις για να γνωρίζει κάθε κόμβος μιας “αλυσίδας” κόμβων με replicas (για το ίδιο αντικείμενο) από πιο id έως ποιο διαθέτει replicas κάθε κόμβος διατηρεί ένα tailHigh και tailLow ids που αντιστοιχούν στο keyRange για το οποίο είναι υπεύθυνος ο πιο απομακρυσμένος κόμβος (προς τα πίσω) για τον οποίο ο ίδιος διαθέτει replica (άρα το τελευταίο keyRange για το οποίο έχουμε στοιχεία).
Η επιλογή του consistency_type και replciation_factor γίνεται σε config αρχείο εντός του πρότζεκτ (απαιτεί ξανά compile πριν την εκτέλεση μετά την αλλαγή του αρχείου).
Ακολουθεί μια σύντομη περιγραφή του πρωτοκόλλου επικοινωνίας των κόμβων που σχεδιάσαμε και υλοποιήσαμε, καθώς και τα αποτελέσματα των μετρήσεων που πήραμε από τα απαιτούμενα πειράματα.

Πρωτόκολλο:

Join:
Κατά την είσοδο ένας κόμβος πρέπει να γνωρίζει τη διεύθυνση και την πόρτα στην οποία ακούει ένας κόμβος που ήδη υπάρχει στο δίκτυο (προτιμήθηκε να μην υλοποιηθεί κόμβος με ειδική λειτουργία ως leader, αλλά μπορεί ο οποιοσδήποτε να αποχωρεί και να εισέρχεται στο δίκτυο). Αυτά τα στοιχεία τα δίνει ο χρήστης ως είσοδο στο πρόγραμμα κατά την εκκίνηση του, μαζί με την port στην οποία πρόκειται να ακούει για συνδέσεις και ο ίδιος ο κόμβος που εκκινεί.

./start_node <my_port> <known_node_address> <known_node_port>

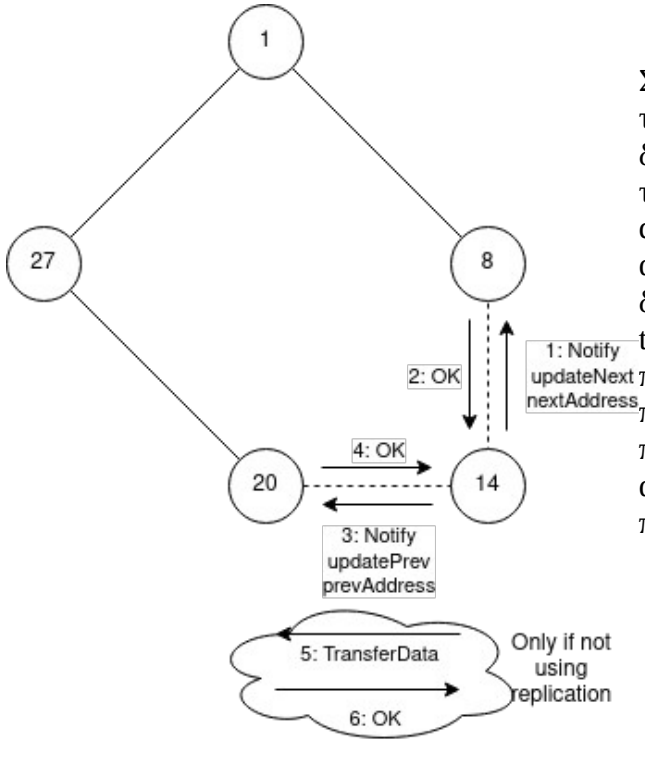
Για την εκκίνηση ενός δικτύου προφανώς δε χρειάζεται να παρέχουμε τίποτα παραπάνω από την πόρτα του κόμβου που πρόκειται να εκκινήσουμε, καθώς δε χρειάζεται ο πρώτος κόμβος του δικτύου να εκτελέσει κάποια λειτουργία join.
Για οποιοδήποτε άλλο κόμβο που εκκινεί, κατά την είσοδο του στο δίκτυο πρέπει να βρει τη θέση του βάση του id του που παράγεται μέσω εφαρμογής της συνάρτησης κατακερματισμού SHA1 στη διεύθυνσή του “<ip_address>:<port>”. Ξεκινώντας από τον ήδη γνωστό κόμβο στο δίκτυο ρωτά για την διεύθυνση αυτού και του επόμενου. Αν πρόκειται να εισέλθει στο διάστημα αυτό τότε ειδοποιεί και τους δύο να αλλάξουν τον επόμενο και προηγούμενο κόμβο τους, αλλιώς συνεχίζει με την ίδια ερώτηση στον επόμενο κόμβο.



Στο παράδειγμα αυτό βλέπουμε κόμβο με id 14 που κάνει join σε δίκτυο ξεκινώντας από τον κόμβο με id 1. Αφότου όμως κάνει join ο επόμενος του φροντίζει να του στείλει τα δεδομένα που του αναλογούν, οπότε εδώ ο 20 στέλνει στον 14 (και διαγράφει από τα data του) τα στοιχεία που είχε για το keyRange 9-14.

Στην περίπτωση χρήσης replication προφανώς πρέπει να ανανεωθούν τα tailHigh και tailLow για K+1 κόμβους ξεκινώντας από τον καινούργιο που μόλις έκανε join (για λόγους μεγάλης έκτασης δεν παρουσιάζεται εδώ ολόκληρη η λειτουργία αυτά αλλά περιγράφεται συνοπτικά).
Στέλνεται λοιπόν ένα μήνυμα από τον νέο κόμβο προς τον εαυτό του και προωθείται και στους K επόμενους. Κάθε ένας που το λαμβάνεις στέλνει ένα askKeyRange μήνυμα που προωθείται προς τα πίσω και ο (K-1)οστός προηγούμενος κόμβος απαντά στον αποστολέα με μήνυμα updateKeyRange που φέρει το το πεδίο για το οποίο αυτός είναι υπεύθυνος (έτσι μπορεί να γνωρίζει κάθε κόμβος για πιο πεδίο μπορεί να απαντήσει σε ερωτήματα στις δύο περιπτώσεις replication). Τέλος, κάθε κόμβος που λαμβάνει updateKeyRange μήνυμα και κάνει update τις αντίστοιχες τιμές του, ανανεώνει τα replicas του διαγράφοντας τα περιττά που φέρει ή ζητώντας όσα δεν έχει αλλά πλέον πρέπει να φέρει, από τον προηγούμενό του στη σειρά (το τελευταίο απαιτείται μόνο στην περίπτωση του νέου κόμβου).

Depart:
Ο κόμβος αφού στείλει notify στον προηγούμενο και επόμενο κόμβο, αποχωρεί από το δίκτυο και σταματά τη λειτουργία του. Στην περίπτωση που δεν χρησιμοποιείται replication, τότε ο κόμβος πριν αποχωρήσει μεταφέρει όλα του τα δεδομένα στον επόμενό του, ο οποίος πρόκειται να γίνει υπεύθυνος για αυτά. Για την εκτέλεση της λειτουργίας αρκεί η πληκτρολόγηση της εντολής *depart* στο CLI του κόμβου.

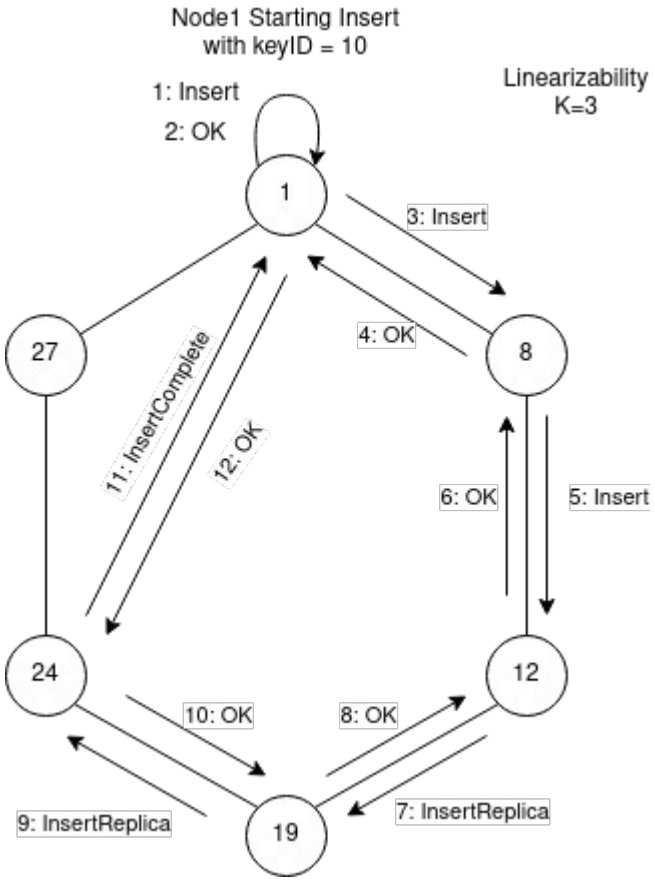


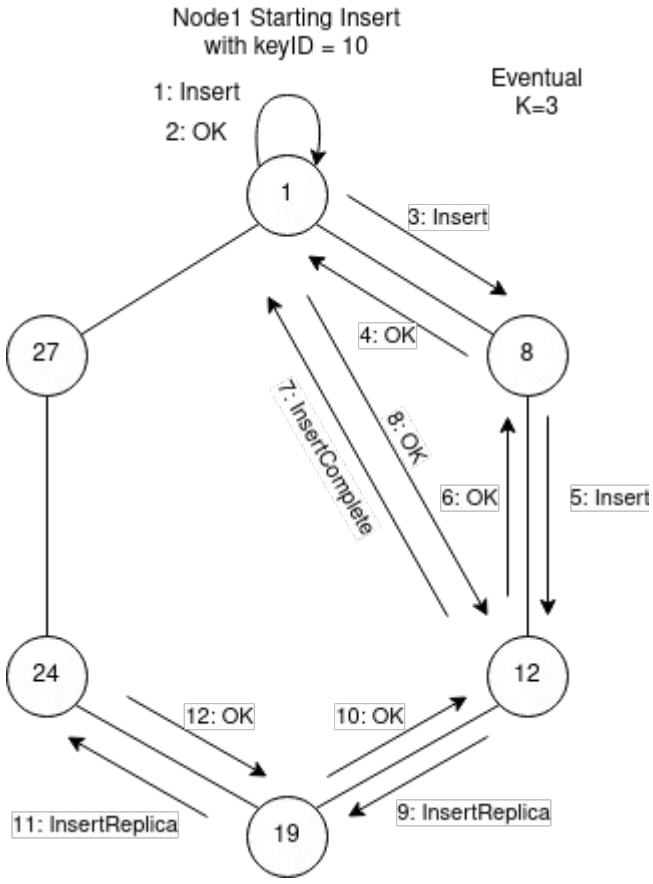
Στην περίπτωση που χρησιμοποιείται replication το dataTransfer δε χρειάζεται καθώς αυτά τα δεδομένα τα έχει ήδη ο επόμενος στα replicas του από τα οποία και τα ανακτά. Κατόπιν του depart οι επόμενοι K κόμβοι μετά του αποχωρούντος πρέπει να ακολουθήσουν την ίδια διαδικασία με πριν για την ανανέωση των tailHigh και tailLow και για τα replicas που πρέπει να βρουν, τα αναζητούν στον προηγούμενο από αυτούς κόμβο ο οποίος και πρέπει προφανώς να τα έχει (δεδομένου ότι δεν ασχολούμαστε με ταυτόχρονα join/depart δεν πρόκειται να δημιουργηθούν κενά στα replicas).

Insert:
Για την εισαγωγή ενός key-value ζεύγους στο toyChord DHT αρκεί η χρήση της αντίστοιχης εντολής από το CLI όπου το κλειδί και η τιμή είανι δύο συμβολοσειρές εντός εισαγωγικών/quotes (“ ”).

insety “<key>” “<value>”

Ο κόμβος από τον οποίο ξεκινάει το αίτημα, στέλνει μήνυμα Insert (ξεκινώντας από τον εαυτό του) το οποίο προωθείται ώσπου να φτάσει στον κόμβο που είναι υπεύθυνος (successor) για το id = SHA1(key). Τότε ανάλογα με τη στρατηγική replication ακολουθείται και διαφορετική πορεία στην εκτέλεση.
Στην περίπτωση που έχουμε linearizability ο υπεύθυνος για το κλειδί κόμβος εισάγει το ζεύγος στα δεδομένα/data του και στη συνέχεια στέλνει InsertReplica μήνυμα το οποίο προωθείται στους επόμενους K-1 κόμβους. Κάθε ένας από αυτούς που το λαμβάνουν τοποθετούν το ζεύγος στα replicas τους ενώ τελικά ο τελευταίος στη σειρά απαντά στο χρήστη (απαντά στον κόμβο που ξεκίνησε το insert αίτημα) σηματοδοτώντας το τέλος της εκτέλεσης του “transaction”.
Στην περίπτωση του eventual consistency, ο κόμβος που είναι υπεύθυνος για το κλειδί, απαντά στον χρήστη αμέσως μόλις έχει προσθέσει το ζεύγος στα δεδομένα του, και στη συνέχεια στέλνει το InsertReplica κατά αντίστοιχο τρόπο στους επόμενους K-1 κόμβους.





Query:
Για εκτέλεση μιας αναζήτησης είναι απαραίτητο να είναι γνωστό το key, και αρκεί η εκτέλεσης της αντίστοιχης εντολής στο CLI:

query “<key>”

Ο κόμβος που εκκινεί μια αναζήτηση στέλνει μήνυμα Query ξεκινώντας από τον εαυτό του και προωθείται ώσπου να φτάσει σε έναν κόμβο ο οποίος να είναι σε θέση να απαντήσει στο ερώτημα. Σε περίπτωση linearizability μπορεί να απαντήσει μόνο ο (K-1)οστός κόμβος μετά του υπεύθυνου για το πεδίο KeyRange στο οποίο υπάγεται το συγκεκριμένο key. Δηλαδή θα απαντήσει μόνο ο κόμβος για τον οποίο ισχύει ότι SHA1(“<key>”).isBetween(tailLow, tailHigh).
Σε περίπτωση eventual consistency αρκεί ένα κόμβος να το έχει είτε ως data είτε ως replica, δηλαδή SHA1(“<key>”).isBetween(tailLow, myID).

Ειδική μέριμνα έγινα για την υλοποίηση της λειτουργίας του ειδικού χαρακτήρα ‘*’ που μπορεί να παραχωρηθεί ως κλειδί. Στην περίπτωση αυτή, στέλνεται ένα μήνυμα QueryAll που προωθείται κάνοντας όλον τον κύκλο του δικτύου και κάθε κόμβος που το λαμβάνει πριν το προωθήσει, προσθέσει σε αυτό τα data του. Μόλις αυτό επιστρέφει στον κόμβο αποστολέα, φέρει όλα τα δεδομένα του δικτύου τα οποία και τυπώνονται στο CLI προς το χρήστη.

Overlay:
Αντίστοιχε με το *query * ,* στέλνεται ένα μήνυμα που προωθείται σε όλο το δίκτυο με κάθε κόμβο να προσθέτει σε αυτό την ip του. Τελικά μόλις φτάσει στο χρήστη, τυπώνονται οι διευθύνσεις και τα id όλων των κόμβων του δικτύου.

PrintAll:
Για λόγους debugging δημιουργήσαμε μια εντολή *printAll* η οποία απλά τυπώνει στο χρήστη όλα τα data και replicas που φέρει ο κόμβος. Προτιμήσαμε να μην το αφαιρέσουμε μετά την υλοποίηση του συστήματος για να διευκολυνθεί ίσως η εξέταση, και γενικά η παρακολούθηση της λειτουργίας του συστήματος.

Πειράματα

Για δίκτυο με 10 κόμβους καλούμαστε να εισάγουμε όλα τα key-value pairs που υπάρχουν στο αρχείο insert.txt που μας παρέχεται, και στη συνέχεια για κάθε key στο query.txt να εκτελέσουμε και αντίστοιχο query, με κάθε ένα από τα παραπάνω να αρχίζει από έναν τυχαίο κόμβο του δικτύου. Από την εκτέλεσή τους μας ενδιαφέρει να βρούμε το insert και query throughput αντίστοιχα για διαφορετικά configuration, και συγκεκριμένα για linearizability και eventual consistency και με K = 1, 3, 5. Η υλοποίηση των προγραμμάτων που εκτελούν της μετρήσεις βρίσκεται εντός του "supplementary" φακέλου του project ενώ υπάρχουν και αντίστοιχα bash scipts για την εύκολη εκτέλεση τους (σε περίπτωση που είναι επιθυμητό να εκτελεστούν εκτός της τοπολογίας στον okeanos που έχουμε φτιάξει, ίσως χρειαστεί μικρή αλλαγή των αντίστοιχων τιμών στο config αρχείο). Στον παρακάτω πίνακα παρουσιάζονται τα αποτελέσματα των μετρήσεων από όλες τις παραπάνω περιπτώσεις εκτέλεσης (τα νούμερα αυτά που παρουσιάζονται είναι ο μέσος όρος των κάποιου πλήθους επαναλήψεων που εκτελέσαμε, ενώ συμπεριλαμβάνεται μαζί με τα παραδοτέα και ένα .txt αρχείο με όλες τις μετρήσεις που πήραμε).

Replication Factor K	Insert Time (sec)		Query Time (sec)	
	Linearizability	Eventual	Linearizability	Eventual
1	2.19	2.11	1.92	1.85
3	3	3.15	1.86	1.68
5	3.72	2.97	1.87	1.6

Insert Throughput

Στην περίπτωση του linearizability όπως είναι φυσικό, με την αύξηση του replication factor (K) ο χρόνος για την ολοκλήρωση των insert αυξάνεται, άρα αντίστοιχα μειώνεται το throughput. Αυτό συμβαίνει γιατί απαιτείται η διάδοση περισσότερων αντιγράφων πριν ολοκληρωθεί κάθε inesrt, αυξάνοντας τον συνολικό απαιτούμενο χρόνο. Επιπλέον λόγω των περισσότερων replicas που πρέπει να μπουν, έχουμε μεγαλύτερο ανταγωνισμό για τα write locks οπότε και μεγαλώνει ο μέσος χρόνος αναμονής μιας εισαγωγής σε έναν κόμβο.
Στην περίπτωση του eventual consistency, ο φόρτος μπορεί να είναι ίδιος με την περίπτωση του linearizability, όμως παρατηρούμε ότι η αύξηση του χρόνου εκτέλεσης των insert ακολουθεί λίγο διαφορετική συμπεριφορά. Συγκεκριμένα βλέπουμε ότι δεν έχει το ίδιο diminishing return αν αυξήσουμε αρκετά το K (για K = 5 η επιβάρυνση δεν αυξήθηκε τελικά), αφού δε χρειάζεται να διαδοθούν όλα τα αντίγραφα για να ολοκληρωθεί ένα insert από τη σκοπιά του χρήστη, αλλά για εκείνον αρκεί να γραφτεί μόνο στον πρώτο κόμβο το insert. Προφανώς ο ανταγωνισμός για τα locks είναι ο ίδιος όμως η απάντηση σε εμάς έρχεται νωρίτερα οπότε βλέπουμε μεγαλύτερο throughput από ότι βλέπουμε για linearizability.
Γενικά όμως για μεγαλύτερο replication factor έχουμε και μείωση του Insert Throughput.

Query Throughput

Στην περίπτωση του linearizability, έχουμε μόνο έναν κόμβο στον δίκτυο που απαντά για ένα συγκριμένο κλειδί, ασχέτως του πλήθους των αντιγράφων, οπότε και ο χρόνος ολοκλήρωσης των queries παραμένει πρακτικά σταθερός άρα και το Query Throughput.
Αντίθετα στην περίπτωση του eventual consistency, όσο μεγαλώνει το replication factor, έχουμε περισσότερους κόμβους να φέρουν το key-value pair που μπορεί να αναζητούμε, και τελικά έχουμε περισσότερες πιθανότητες να λάβουμε και απάντηση από τον ίδιο τον κόμβο που εκτελείται το ερώτημα (η επιλογή του κόμβου προφανώς γίνεται τυχαία οπότε ανά εκτέλεση μπορεί να έχουμε διαφορετικά αποτελέσματα). Άρα για μεγαλύτερα K είναι λογικό να παρατηρούμε και μεγαλύτερο Query Throughput.

Requests

Για δίκτυο με 10 κόμβους και replication factor K = 3 καλούμαστε να τρέξουμε τις εντολές (insert/query) που περιλαμβάνονται στο αρχείο requests.txt και να συγκρίνουμε τα αποτελέσματα των απαντήσεων. Για διευκόλυνση μας θέσαμε στα μηνύματα και ένα timestamp της στιγμής αποστολής της απάντησης, άρα και ολοκλήρωσης του transaction.

ΠΡΟΣΟΧΗ

Το timetsamp αυτό τοποθετήθηκε μόνο για δική μας διευκόλυνση και είναι έγκυρο υπό την προϋπόθεση ότι όλοι οι κόμβοι τρέχουν στο ίδιο φυσικό μηχάνημα ή έχουν κεντρικό ρολόι από το οποίο λαμβάνουν τον χρόνο. Στη συγκεκριμένη περίπτωση είναι VMs στον okeanos και ελέγχθηκε με κατάλληλο script ότι δίνουν την ίδια ώρα. Κάτι τέτοιο σε ένα κατανεμημένο σύστημα δεν υφίσταται (θα ήταν λάθος).

Τα responses που τυπώνονται στην έξοδο είναι προφανώς out of order και δεν μπορούν να μας δείξουν με ποια σειρά εκτελούνται τα διαφορετικά transactions στο δίκτυο. Ποιοτικά βέβαια παρατηρούμε ότι το linearizability επιστρέφει λίγο πιο πρόσφατα αποτελέσματα (αρχεία linear_output.txt και eventual_output.txt για linearizability και eventual consistency αντίστοιχα).

Με χρήση του timestamp και εκτελώντας βάση αυτού ένα pseudo ordering των γεγονότων βλέπουμε ένα λογικό αποτέλεσμα όπου η σειρά αυτή είναι έγκυρη καθώς μετά από κάθε insert, οποιοδήποτε query ακολουθεί επιστρέφει πάντα και την πιο πρόσφατη τιμή. Αυτό ισχύει πάντα σε linearizability.
Τώρα για eventual consistency αναμένουμε ότι είναι πολύ πιθανό να παρατηρήσουμε stale reads όπου κάποιο query μπορεί να συμβεί για ένα <x> κλειδί σε έναν κόμβο που το έχει ως replica αφότου έχει προηγηθεί insert και τροποιήση του <x> στον κύριο κόμβο χωρίς όμως να έχει ακόμα φτάσει παντού το replica του. Αυτό όμως δεν καταφέρουμε να το παράγουμε τελικά στην πράξη γιατί για K=3 έχουμε ένα πολύ μικρό window of opportunity για stale read με πιθανότητα μόλις μία στις 5 να πέσει query αμέσως μετά insert, σε έναν από τους δύο επόμενους κόμβους πριν αποκτήσουν ακόμα την πιο πρόσφατη τιμή. Για να μην βρεθούμε λοιπόν με άδεια χέρια αυξήσαμε το K παραπάνω ώστε να έχουμε περισσότερες πιθανότητες να αναπαράγουμε το φαινόμενο αυτό. Όπως είναι λογικό, για linearizability συνεχίσαμε να έχουμε το ίδιο ακριβώς αποτέλεσμα όπου πάντα επιστρέφεται η πιο πρόσφατη τιμή. Για eventual consistency βρήκαμε περιπτώσεις όπου κατόπιν ολοκλήρωσης του insert, επιστράφηκαν παλαιότερες τιμές από query σε άλλον κόμβο του δικτύου όπως φαίνεται στο παρακάτω απόσπασμα της εξόδου που συλλέξαμε το οποίο σε αρκετά σημεία φαίνεται να απεικονίζει το φαινόμενο αυτό (για ευκολία κάποια τοποθετήθηκαν με **bold**).

(Οι έξοδοι όλων αυτών των εκτελέσεων είναι διαθέσιμες στον φάκελο experiments. Το αρχείο eventual_cheating.txt περιέχει την έξοδο για μεγαλύτερο K. Επίσης κρατήσαμε και τις pseudo_ordered εκδόσεις σε περίπτωση που απαιτηθεί επαλήθευσή τους, δεδομένου ότι παρουσιάζουμε μόλις ένα μικρό δείγμα ενός μόνο των αρχείων με τις εξόδους.)

Eventual sample:

<Like a Rolling Stone,1> Response from Node : 192.168.1.2:8800 at nsec: 283025164922528
<Hey Jude,502> inserted. Response from Node : 192.168.1.2:8700 at nsec: 283025165667805
<Hey Jude,502> Response from Node : 192.168.1.2:8700 at nsec: 283025166257544
<Hey Jude,8> Response from Node : 192.168.1.2:8800 at nsec: 283025170451663
<Hey Jude,502> Response from Node : 192.168.1.2:8700 at nsec: 283025188834578
<Respect,5> Response from Node : 192.168.1.2:8700 at nsec: 283025191840281
<Like a Rolling Stone,1> Response from Node : 192.168.1.2:8700 at nsec: 283025192886166
<Hey Jude,8> Response from Node : 192.168.1.2:8800 at nsec: 283025196365978
<Hey Jude,502> Response from Node : 192.168.1.2:8700 at nsec: 283025200865170
<Respect,5> Response from Node : 192.168.1.2:8800 at nsec: 283025212739825
<Like a Rolling Stone,504> inserted. Response from Node : 192.168.1.2:8700 at nsec: 283025216778344
<Hey Jude,513> inserted. Response from Node : 192.168.1.2:8700 at nsec: 283025229352609
<Respect,5> Response from Node : 192.168.1.2:8800 at nsec: 283025229543551
<Like a Rolling Stone,504> Response from Node : 192.168.1.2:8700 at nsec: 283025229992246
<Hey Jude,501> inserted. Response from Node : 192.168.1.2:8700 at nsec: 283025232685184
<Hey Jude,501> Response from Node : 192.168.1.2:8700 at nsec: 283025237190688
<Hey Jude,8> Response from Node : 192.168.1.2:8800 at nsec: 283025238530978
<Hey Jude,501> Response from Node : 192.168.1.2:8700 at nsec: 283025240788043
<Hey Jude,501> Response from Node : 192.168.1.2:8700 at nsec: 283025244753920
<Like a Rolling Stone,504> Response from Node : 192.168.1.2:8700 at nsec: 283025245183176
<Like a Rolling Stone,1> Response from Node : 192.168.1.2:8800 at nsec: 283025245609701
<Like a Rolling Stone,504> Response from Node : 192.168.1.2:8700 at nsec: 283025246084276