Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Η/Υ

Κατανεμημένα Συστήματα - Ροή Υ

Εξαμηνιαία Εργασία - Αναφορά

Ονοματεπώνυμα:	**Αριθμοί Μητρώου:**
Οικονόμου Ελένη	el16301
Σταθά Ευσταθία	el16190
Τόλιας Δημήτριος	el15164

Πειράματα

1. Insert

1.1. Αποτελέσματα

Throughput (inserts/second)

	k = 1	k = 3	k = 5
eventual	34.0253	34.8043	34.3919
linearizability	37.3643	28.0456	22.3328

Time/Insert (seconds/insert)

	k = 1	k = 3	k = 5
eventual	0.0294	0.0287	0.0291
linearizability	0.0268	0.0357	0.0448

1.2 Σχολιασμός

Όσων αφορά το write throughput παρατηρούμε μια αρκετά αναμενόμενη συμπεριφορά για τους δύο τύπους consistency. Συγκεκριμένα, για την πιο χαλαρή μορφή συνέπειας (eventual) παρατηρούμε πως η αύξηση του πλήθους των replicas δεν αλλάζει σχεδόν καθόλου το ρυθμό εγγραφής των διαφόρων ζευγών κλειδιού-τιμής. Αυτό οφείλεται στο γεγονός πως, μόλις ένα αίτημα για insert φτάσει στον κόμβο που είναι υπεύθυνος για αυτή την τιμή, ο ίδιος απαντάει πίσω στον κόμβο που του έστειλε το αίτημα πως η εισαγωγή ολοκληρώθηκε, ανεξαρτήτως του αν υπάρχουν ή όχι και άλλοι Replica Managers που πρέπει να ενημερώσουν τα δεδομένα τους εισάγοντας τα αντίστοιχα αντίγραφα της τιμής που μόλις αποθηκεύτηκε στο σύστημα. Στην πραγματικότητα, αν το replication factor είναι 1, τότε εκεί έχει ολοκληρωθεί η δουλειά του υπεύθυνου κόμβου, σε αντίθετη περίπτωση ενημερώνει τότε τους επόμενους του για να εισάγουν το ζεύγος στα δεδομένα τους. Ωστόσο, στην πιο αυστηρή μορφή συνέπειας (linearizability) δεν παρατηρούμε την ίδια συμπεριφορά, αλλά αύξηση του replication factor προκαλεί μείωση του write throughput. Το γεγονός εξηγείται εύκολα, αν κανείς αναλογιστεί πως, πλέον, σε αντίθεση με προηγουμένως, η εισαγωγή ενός κλειδιού για έναν ή περισσότερους Replica Managers δεν καθυστερούν το response που θα σταλεί σε αυτόν που έκανε το αίτημα με τον ίδιο τρόπο. Αν δεν υπάρχουν replicas στο σύστημα, το ίδιο συμπεριφέρεται λίγο-πολύ όπως και με την πιο χαλαρή μορφή συνέπειας. Αν, όμως,

έχουμε κάποιους Replica Managers που πρέπει να ενημερωθούν, τότε ο κόμβος που είναι υπεύθυνος για το εκάστοτε κλειδί ενημερώνει τον αμέσως επόμενο του στην αλυσίδα των κόμβων και μόνο όταν η εισαγωγή ολοκληρωθεί και στον τελευταίο Replica Manager ο ίδιος απαντάει σε αυτόν που έστειλε το αίτημα για το insert πως η εισαγωγή ολοκληρώθηκε. Ασφαλώς, είναι προφανές πως όσο μακρύτερη γίνεται η αλυσίδα των κόμβων που κρατούν αντίγραφο για κάθε ζεύγος κλειδιού-τιμής, τόσο περισσότερο καθυστερεί και η απάντηση που σηματοδοτεί την ικανοποίηση του αιτήματος, έτσι μεγαλύτερο replication factor μειώνει το write throughput.

2. Query

2.1. Αποτελέσματα

Throughput (queries/second)

	k = 1	k = 3	k = 5
eventual	38.0702	50.4071	62.8726
linearizability	39.903	27.7763	23.0783

Time/Query (seconds/query)

	k = 1	k = 3	k = 5
eventual	0.0263	0.0198	0.0159
linearizability	0.0251	0.0360	0.0433

2.2 Σχολιασμός

Όσων αφορά το read throughput παρατηρούμε και πάλι μια συμπεριφορά που κρίνεται αρκετά αναμενόμενη και για τους δύο τύπους consistency. Συγκεκριμένα, για την πιο χαλαρή μορφή συνέπειας (eventual) παρατηρούμε πως η αύξηση του πλήθους των replicas αυξάνει τον ρυθμό ανάγνωσης των διαφόρων ζευγών κλειδιού-τιμής. Αυτό οφείλεται στο γεγονός πως, μόλις ένα αίτημα τύπου query για κάποιο κλειδί φτάσει σε κάποιον κόμβο, ο ίδιος ελέγχει αν έχει την τιμή για το κλειδί αυτό κι αν την έχει απαντάει πίσω τον κόμβο που έστειλε το αίτημα, χωρίς να κάνει κάποια περαιτέρω προώθηση του αιτήματος. Είναι προφανές, πως αν το σύστημα μας δεν διατηρεί αντίγραφα, ανεξαρτήτως του μεγέθους του δακτυλίου, πάντα μόλις ένας κόμβος θα έχει την απαραίτητη πληροφορία και θα μπορεί να ικανοποιήσει ένα αίτημα τύπου query. Όσο, όμως, προστίθενται Replica Managers τόσο πιθανότερο

είναι ένα αίτημα να πέσει σε κάποιον κόμβο που έχει την τιμή για το εκάστοτε κλειδί, με αποτέλεσμα ο χρόνος της αναζήτησης να μειώνεται όλο και περισσότερο. Ωστόσο, στην πιο αυστηρή μορφή συνέπειας (linearizability) δεν παρατηρούμε την ίδια συμπεριφορά, αλλά βλέπουμε μείωση του read throughput για αύξηση του replication factor. Το γεγονός εξηγείται εύκολα, αν κανείς αναλογιστεί πως, τώρα, σε αντίθεση με την προηγούμενη μορφή συνέπειας που εξετάστηκε, η αναζήτηση ενός κλειδιού για έναν ή περισσότερους Replica Managers μπορεί να ικανοποιείται μόνο από έναν κόμβο και, μάλιστα, τον τελευταίο στην αλυσίδα των Replica Managers. Αν δεν υπάρχουν replicas στο σύστημα, το ίδιο συμπεριφέρεται λίγο-πολύ όπως και με την πιο χαλαρή μορφή συνέπειας. Αν, όμως, έχουμε κάποιους Replica Managers που διατηρούν αντίγραφα για το κάθε ζεύγος κλειδιού-τιμής, τότε πρέπει να βεβαιωθούμε πως η απάντηση σε ένα query θα γίνεται αυστηρά από τον τελευταίο κόμβο στην αλυσίδα των υπεύθυνων. Ασφαλώς, λοιπόν δεν θα μπορούσαμε να οδηγηθούμε σε αύξηση του read throughput, όπως στην περίπτωση του eventual consistency. Φυσικά, θα μπορούσε ο χρόνος της αναζήτησης απλώς να παραμένει σχεδόν σταθερός, αφού όσο κι αν αυξάνεται το πλήθος των αντιγράφων στο σύστημα πάντα μόλις ένας κόμβος είναι υπεύθυνος για την ικανοποίηση του αιτήματος. Ωστόσο, η υλοποίηση μας χειρίζεται τα αιτήματα αυτά με έναν ιδιαίτερο τρόπο που οδηγεί στη μείωση του read throughput με αύξηση του πλήθους των Replica Managers. Συγκεκριμένα, στην υλοποίηση μας φροντίζουμε ώστε το αίτημα να φτάνει πρώτα στον υπεύθυνο κόμβο για κάθε κλειδί κι, έπειτα, αφού διατρέξει όλη την αλυσίδα ο τελευταίος να αναλάβει να στείλει το response στον κόμβο που έκανε το αίτημα, παραθέτοντας του την τιμή για το κλειδί αυτό. Δηλαδή, σε κάθε περίπτωση τα αιτήματα αναζήτησης δρομολογούνται αρχικά για τον υπεύθυνο του κλειδιού και, έπειτα, αυτός τα δρομολογεί εκ νέου προς τον τελευταίο Replica Manager της αλυσίδας. Αυτή η επιλογή έγινε για να εξασφαλίζουμε πως κάθε φορά θα απαντάει ο τελευταίος και θα μπορούσαμε, ενδεχομένως, να ικανοποιούμε αυτή την απαίτηση με διαφορετικό τρόπο.

3. Requests

3.1. Αποτελέσματα

3.2 Σχολιασμός

Όσων αφορά το freshness των αποτελεσμάτων κάνουμε, αρχικά, έναν σχολιασμό αναφορικά με το τι θα περιμέναμε να δούμε και, έπειτα, αναφέρουμε τι παρατηρήσαμε στα requests που έγιναν στο σύστημα. Στην αυστηρότερη μορφή συνέπειας (linearizability) αναμένουμε εκτελώντας αναζήτηση για ένα κλειδί, έπειτα από ένα insert κάποιου ζεύγος κλειδιού-τιμής, να λαμβάνουμε την τιμή που μόλις εισήχθει. Θα πρέπει το σύστημα να υλοποιείται με τρόπο τέτοιο, ώστε να προσφέρει ισχυρή εγγύηση σχετικά. Άλλωστε, αυτή η μορφή συνέπειας θυσιάζει σε απόδοση, όπως είδαμε με τα προηγούμενα παραδείγματα,

προκειμένου να μας προσφέρει πάντα σωστά δεδομένα. Πράγματι, δεν παρατηρήσαμε την εμφάνιση παλαιών δεδομένων για κάποιο κλειδί. Όσων αφορά την πιο χαλαρή μορφή συνέπειας (eventual), εκεί δεν υπάρχουν αυστηρές σχετικές εγγυήσεις. Μια αναζήτηση κάποιου κλειδιού μπορεί να μας δώσει την νεότερη ή κάποια παλαιότερη τιμή, ανάλογα πως έχει υλοποιηθεί το σύστημα και σε ποιον κόμβο πηγαίνει το αίτημα αναζήτησης. Η συμπεριφορά αυτή είναι τόσο αποδεκτή, όσο και αναμενόμενη, καθώς η συγκεκριμένη μορφή συνέπειας εγγυάται μεν πως κάποια στιγμή όλα τα αντίγραφα θα συγκλίνουν στην ίδια τιμή, όμως δεν εγγυάται πως κάθε στιγμή όλοι οι κόμβοι έχουν τα νεότερα δεδομένα που εισήχθησαν στο σύστημα. Στα requests που παρατηρήσαμε, συνήθως το σύστημα λειτουργούσε με αρκετά συνεπή τρόπο, και μόνο ελάχιστες φορές είδαμε παλαιότερα δεδομένα για κάποιο κλειδί.