University of Crete

Computer Science Department

Diploma Thesis

Improving the efficiency of RocksDB state recovery from a local HDFS replica

Επιτάχυνση της ανάκτησης κατάστασης της βάσης κλειδιού-τιμής RocksDB από τοπικό αντίγραφο κατανεμημένου συστήματος αρχείων HDFS

Λευτέρης Συνατσάκης AM 4102

Advisor: Kostas Magoutis

Examination Committee: Kostas Magoutis, Dimitris Plexousakis

Heraklion, July 2022

Abstract

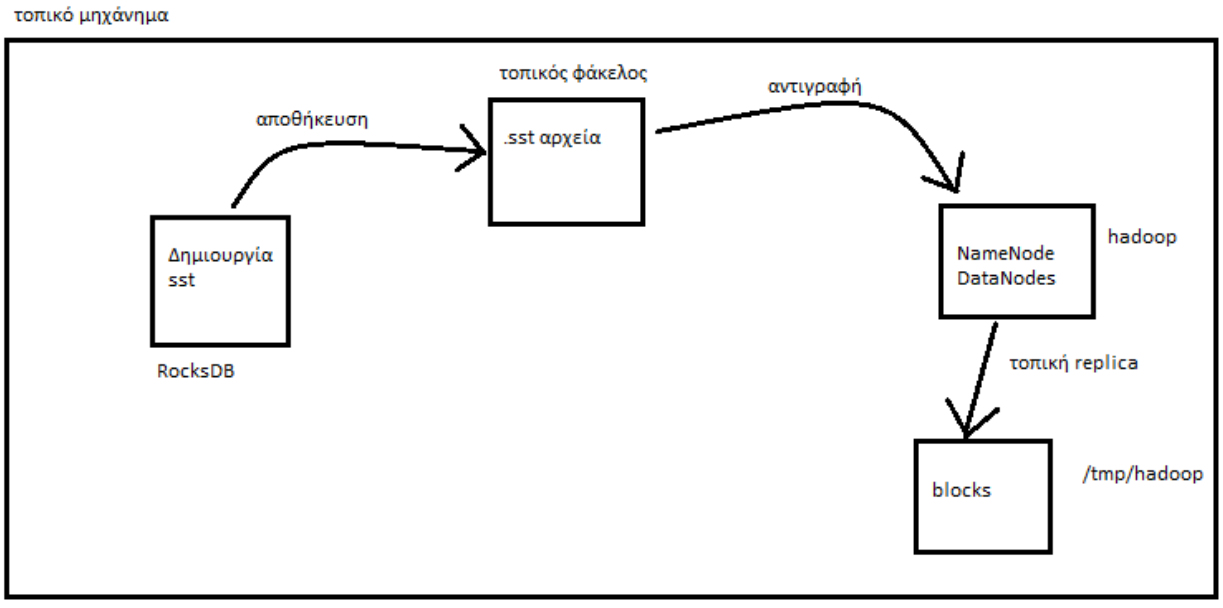
Η παρούσα διπλωματική εργασία μελετά μια νέα μέθοδο για την επιτάχυνση της ανάκτησης των αρχείων που παράγει ένα σύστημα κλειδιού-τιμής, η RocksDB, από αντίγραφά τους τα οποία υπάρχουν σε σύστημα αρχείων HDFS και το οποίο τα αποθηκεύει τοπικά. Η τεχνική την οποία υλοποιήθηκε και μελετήθηκε πειραματικά και η οποία αντικαθιστά την αντιγραφή δεδομένων με διαχείριση δεικτών στα δεδομένα, μας βοηθάει στο να μειώσουμε τον χρόνο ανάκτησης και την μείωση χρήσης πόρων του μηχανήματος, ειδικά για μεγάλα αρχεία τα οποία θα χρειάζονταν μεγάλο χρόνο για την αντιγραφή από αρχείο σε αρχείο. Η τεχνική την οποία υλοποίησα ταυτοποιεί τα τοπικά Linux αρχεία το οποίο αντιστοιχούν σε κάποιο αρχείο HDFS στο τοπικό σύστημα αρχείων και δημιουργεί νέο αρχείο RocksDB μέσω εντολών hard link στα πρώτα. Η μείωση του χρόνου ανάκτησης του αρχείου της RocksDB είναι αξιοσημείωτη, αφού το speedup μας μπορεί να φτάσει από τρεις φορές εώς και εκατοντάδες φορές επί της συμβατικής λύσης (αντιγραφή αρχείων).

Table of contents

Εισαγωγή

Αρχικά επέλεξα αυτό το θέμα γιατί μου φαινόταν αρκετά ενδιαφέρον και χρήσιμο το γεγονός ότι θα είχα μια αρκετά καλή διατριβή με τα κατανεμημένα συστήματα, κάτι το οποίο το θεωρώ πολύ σημαντικό καθώς είναι πολύ βασικό στον τομέα της επιστήμης υπολογιστών και πόσο μάλλον στο software κομμάτι που θέλω να ασχοληθώ μελλοντικά.

Ένα πρόβλημα που υπάρχει διαχρονικά είναι το να έχουμε κάποιου είδους checkpoint στα δεδομένα μας. Μπορεί αυτό να έχει λυθεί με μηχανισμούς όπως το Hadoop, αλλά τίθεται ένα δεύτερο πρόβλημα το οποίο είναι η ταχύτητα ανάκτησης, καθώς αν υπάρξει κάποια δυσλειτουργία με τα αρχεία του μηχανήματός μας (χαθούν ή καταστραφούν) τότε είμαστε αναγκασμένοι να περιμένουμε όσο χρειαστεί, αν υπάρχουν σε τοπικό αντίγραφο HDFS, μέχρι το Hadoop να κάνει copy χάνοντας πολύ πολύτιμο χρόνο. Πόσο μάλλον αν τα αρχεία είναι αρκετά GB ή ακόμα χειρότερα η ταχύτητα του ίδιου του δικτύου μας είναι αργή. Αρχικά είχαμε τις αμφιβολίες μας αν το αρχείο δεν ήταν ακριβώς το ίδιο ή ακόμα χειρότερα δεν το αναγνώριζε η RocksDB, κάτι το θα σήμαινε ότι δεν θα μπορούσαμε να προχωρήσουμε την έρευνά μας. Εμείς προσπαθήσαμε και τα καταφέραμε, άν το αρχείο ανήκει σε τοπικό αντίγραφο HDFS να υπάρχει μια γρήγορη ανάκτηση της κατάστασης της μηχανής και των αρχείων χωρίς να μας νοιάζει καθόλου η ταχύτητα του δικτύου μας. Μια γενική ιδέα φαίνεται και στην Εικόνα 1.



Εικόνα 1: Αλληλεπίδραση RocksDB με δημιουργία .sst αποθήκευση στο hadoop και δημιουργια blocks

Στην περίπτωση που θέλουμε να κάνουμε ανάκτηση στα αρχεία μας και από τα blocks του hadoop να τα ενώσουμε σαν files τότε τα βελάκια της Εικόνας 1 θα αλλάξουν φορά.
Για να μπορέσει κάποιος να δει τα αποτελέσματα αρχικά πρέπει να έχει εγκαταστήσει την python και την java απαραίτητα, επιπρόσθετα πρέπει να έχει την RocksDB καθώς και το hadoop στο

μηχάνημά του. Έπειτα να μπορέσει να τρέξει την RocksDB που θα δημιουργήσει/διαβάσει τα αρχεία μας και στην συνέχεια να τα αποθηκεύσει τοπικά στο HDFS,σε περίπτωση που αυτά χαθούν τότε έχουμε υλοποιήσει ένα πρόγραμμα που κάνει αυτόματα την ανάκτηση και σύνδεση των αρχείων με βάση των blocks που δημιουργεί το hadoop.Βέβαια είχαμε παρατηρήσει ότι το πρόγραμμα μας δεν μπορεί να διαχειριστεί ακριβώς με τον ίδιο τρόπο αρχεία τα οποία έχουν γίνει ανάκτηση ήδη μια φορά, κάτι το οποίο στην αρχή μας φάνηκε λίγο περίπλοκο αλλά το λύσαμε σχετικά εύκολα. Παρόλα αυτά, το πρόγραμμά μας υλοποιεί ακριβώς αυτό που θέλουμε χωρίς απώλεια αρχείων και προφανώς με ένα ακριβές αντίγραφο σε καθένα από αυτά. Τα πειράματα που υλοποιήσαμε ήταν αρχικά μικρά αρχεία ώστε να δούμε αν το αρχικό αρχείο είναι ίδιο με αυτό που δημιουργούμε, όταν τα πειράματά μας ήταν όλα επιτυχή τότε το δοκιμάσαμε σε μεγαλύτερα αρχεία για να δούμε την διαφορά στην ταχύτητα, δηλαδή αν καταφέραμε τον αρχικό στόχο μας.

Πρώτα, είχαμε παρατηρήσει ότι το HDFS σπάει το αρχείο σε blocks, οπότε κάναμε την υπόθεση πως η ένωση των μικρών αυτών blocks θα μας οδηγήσει στην κατάσταση που βρισκόμασταν.Το θέμα ήταν ότι η ένωση έπρεπε να γινόταν byte ανά byte. Δυστυχώς στο linux δεν μας δίνεται κάποιος μηχανισμός ώστε να ενώσουμε πολλά αρχεία σε ένα temp αρχείο που θα αποτελείται από αυτά τα μικρότερα, οπότε η αρχική λύση να κρατάμε κείμενο όπου απλά θα το διαβάζουμε κάθε φορά για να ξέρουμε πιο αρχείο αντιστοιχεί που ήταν, εκτός από περίπλοκο στο πώς θα ξαναδιαβάσουμε το αρχείο, θα έκανε την επεξεργασία του ίσως και αδύνατη κάτι το οποίο δεν θα ήταν αποδεκτό.

Οπότε καταλήξαμε στο γεγονός να έχουμε ένα αρχείο στο οποίο θα ενώσουμε με την σειρά τα blocks που δημιουργούνται από το HDFS, byte προς byte ώστε το τελικό αποτέλεσμα να είναι το αρχικό μας αρχείο. Για να κάνουμε επίσης σωστά τον έλεγχο κρατήσαμε το αρχικό μας αρχείο σε ένα άλλο μηχάνημα και όπως αποδείχθηκε δεν υπήρχε διαφορά.

Στην λύση και το τελικό μας σκοπό έφτασα με την άριστη συννενόηση που είχα με τον υπεύθυνο καθηγητή μου κ.Μαγκουτη καθώς και την άψογη επίγνωση του πάνω στο θέμα. Έπειτα για εντολές και λειτουργίες που έπρεπε να είχα μια εξοικείωση με βοήθησε η αναζήτηση είτε από το διαδίκτυο είτε από papers ατόμων που έχουν ασχοληθεί στο παρελθόν

Σε αυτή η εργασία κάνουμε τις εξής συνεισφορές

- Υλοποιούμε ένα σύστημα ανάκτησης ενός τοπικού αρχείου Linux από ένα αρχείο HDFS του οποίου αντίγραφο βρίσκεται στο ίδιο μηχάνημα

- Αξιολογούμε το υλοποιηθέν σύστημα δείχνοντας πειραματικά τα πλεονεκτήματα τα οποία προσφέρει

Το σύστημα το οποίο υλοποιήθηκε σε αυτή την εργασία μπορεί να χρησιμοποιηθεί για την επιτάχυνση της διαδικασίας ανάκαμψης ενός ευρύτερου συστήματος το οποίο χρησιμοποιεί την RocksDB για αποθήκευση κατάστασης τοπικά και το HDFS για την περιοδική παραγωγή

στιγμιοτύπων αυτής της κατάστασης σε απομακρυσμένο κόμβο. Παρότι τα αντίγραφα του αρχείου HDFS βρίσκονται αρχικά σε απομακρυσμένους κόμβους, σε περίπτωση κατάρρευσης του κόμβου που φιλοξενεί την RocksDB, η ανάκαμψη της RocksDB μπορεί να συμβεί σε κόμβο που φιλοξενεί τοπικά αντίγραφο του HDFS, δημιουργώντας της ευκαιρία ταχύτερης ανάκαμψης των αρχείων της RocksDB σε αυτό τον κόμβο μέσω των μηχανισμών που περιγράφονται σε αυτή την εργασία.

Υπόβαθρο

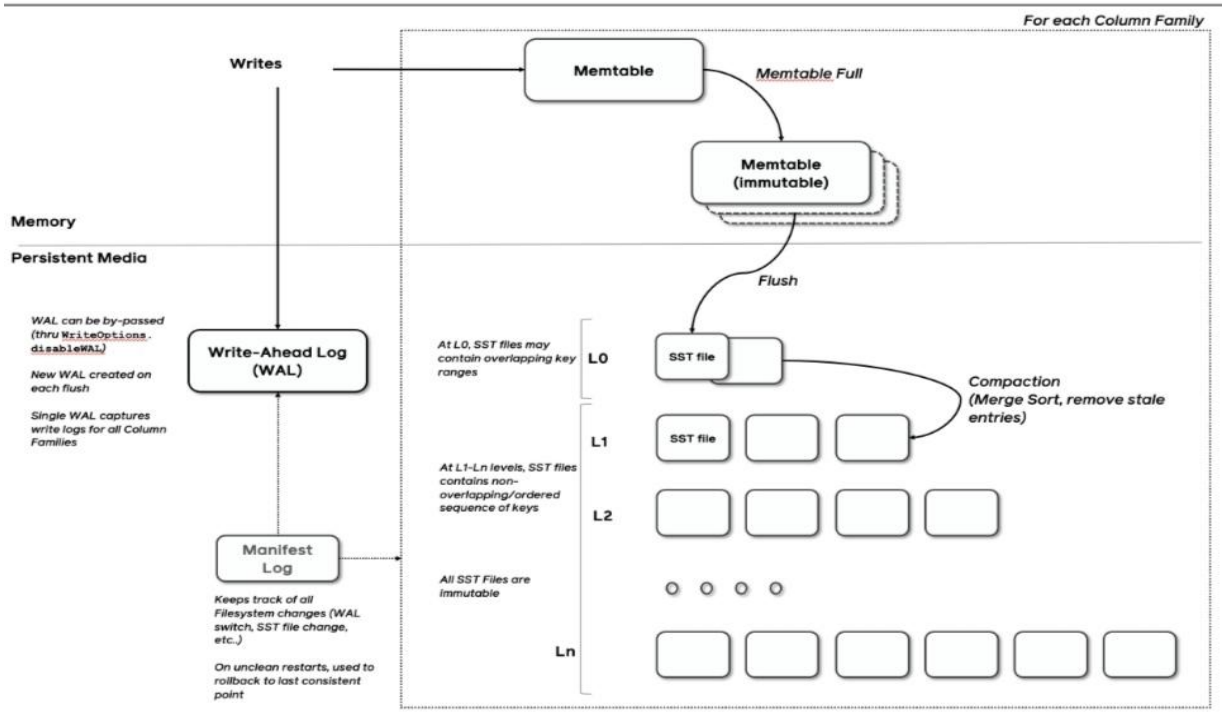
Αρχικά για να μπορέσει κάποιος να κατανοήσει την πτυχιακή πρέπει να κατανοήσει βασικές έννοιες και εργαλεία όπως το ycsb, την rocksdb, sst files, το hdfs, τα blocks καθώς και πως λειτουργούν.

Η βάση κλειδιού-τιμής RocksDB

Το θεωρούμε σαν ένα σύστημα το οποίο στηρίζεται στην τεχνική key-value για γρήγορη αποθήκευση.Το συγκεκριμένα σύστημα είναι open-source και έχει δημιουργηθεί από τους μηχανικούς της γνωστής εταιρίας Facebook.Λόγο της αρχιτεκτονικής που ακολουθεί είναι πολύ πιθανό να χρησιμοποιηθεί σαν θεμέλιο σε βάση δεδομένων client-server, ακόμα μπορεί να χρησιμοποιηθεί και σε μεγάλο database τόσο σαν μηχανή αποθήκευσης όσο και σαν μηχανή για ανάλυση δεδομένων.

Έπειτα ο τρόπος λειτουργίας της είναι να αποθηκεύει τα key-values σε memory buffers(memtables) από byte και στην συνέχεια να τα ταξινομεί. Εφόσον η RocksDB ειναι υλοποιημένη σε C++ το memtable είναι στην ουσία ένα vector όπου κάθε νέο entry κάνει insert στο τέλος.Πληροφοριακά αν και δεν φτάσαμε τόσο βαθιά διαθέτει έναν μηχανισμό που λέγεται WAL (Write Ahead Log) ο οποίος ξανα εκτελεί όλα τα write operation που γράφτηκαν στο memtable (put,delete,merge) σε κάθε restart.

Όταν αυτό το buffer (βλέπουμε το παράδειγμα από κάτω) γεμίσει τότε αυτό γίνεται flush στον δίσκο και αποθηκεύεται σε ένα αρχείο .sst(sorted static table) που όπως είναι λογικό το συγκεκριμένο αρχείο περιέχει όλες τις πληροφορίες του πίνακα μέχρι το συγκεκριμένο σημείο και διαγράφει οποίο από τα keys έχουν ή ξαναγραφτεί ή δεν είναι μοναδικά.Όλα τα sst files είναι αποθηκευμένα σε ένα LSM tree για να μπορεί η RicksDB να γράψει γρήγορα,ειδικά σε περίπτωση που τα δεδομένα μας βρίσκονται στην ram και όχι στον δίσκο, καθώς θα καταλήγαμε σε bottleneck. Ένα παράδειγμα για την λειτουργία της RocksDB φαίνεται στην Εικόνα 2.



Εικόνα 2: Λειτουργία RocksDB καθώς γίνονται flush τα data[1]

Το benchmark YCSB

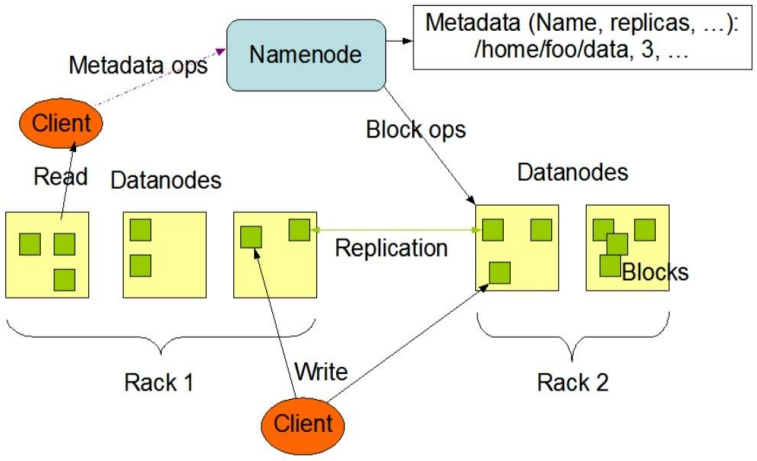
Το συγκεκριμένο σύστημα το χρησιμοποιούμε μόνο για την δημιουργία των sst αρχείων που είχαμε ασχοληθεί κατά κύριο λόγο.Στην ουσία είναι ένα benchmark που έχει δημιουργηθεί από τους μηχανικούς της Yahoo ώστε να μπορούν να ελέγξουν τις δυνατότητες ενός προγράμματος. Ακόμα, το συγκεκριμένο benchmark μας δίνει την δυνατότητα μέσω του αρχείου workloada και της μεταβλητής recordcount να επιλέξουμε πόσες περίπου γραμμές επιθυμούμε να έχουμε,οπότε σαν συνέπεια να ελέγχουμε και το πόσο φόρτο να έχει το memtable για την δημιουργία των sst.

Το πλαίσιο κατανεμημένης αποθήκευσης/επεξεργασίας HADOOP

Το Hadoop είναι ένα framework το οποίο χρησιμοποιείται για παράλληλη αποθήκευση και επεξεργασία πολύ μεγάλων datasets,έπειτα είναι open source και έχει σχεδιαστεί να λειτουργεί από μεμονωμένους server μέχρι και σε χιλιάδες μηχανές ταυτόχρονα.Δεν επηρεάζεται από το hardware καθώς έχω σχεδιαστεί ώστε να μπορεί να ανιχνεύει αστοχίες στο επίπεδο εφαρμογής. Χρησιμοποιεί την τεχνική του clustering οπότε με το να λειτουργούν πολλά μηχανήματα παράλληλα μπορούμε να αναλύσουμε δεδομένα γρηγορότερα.

[1] https://github.com/facebook/rocksdb/wiki/RocksDB-Overview

Συγκεκριμένα,το εργαλείο του Hadoop που ασχολήθηκα είναι το HDFS(Hadoop distributed file system) είναι ένα κατανεμημένο σύστημα αρχείων του οποίου η δουλειά είναι η αποθήκευση δεδομένων.Είναι πολύ ανθεκτικό σε σφάλματα σε σχέση με τα άλλα κατανεμημένα συστήματα και έχει σχεδιαστεί για να λειτουργεί χωρίς να επηρεάζεται από το hardware.Το HDFS ακολουθεί την τεχνική master-slave όπου έχει θέσει σαν master τον NameNode και σαν slaves τους DataNodes.Ο NameNode έχει πλήρη χαρτογράφηση των αρχείων και των blocks, δηλαδή σε ποιο ακριβώς DataNode βρίσκονται και σε ποιο αρχείο ανήκουν.Επίσης ο NameNode είναι υπεύθυνος για το πώς θα αποθηκεύσει και θα χωρίσει τα δεδομένα καθώς και σε ποιο DataNode θα καταλήξει το κάθε block που δημιουργείται.Ακόμα ο NameNode εκτελεί λειτουργίες όπως άνοιγμα,κλείσιμο και μετονομασία αρχείων και φακέλων, ενώ οι DataNodes είναι υπεύθυνοι για την εξυπηρέτηση των αιτημάτων ανάγνωσης και εγγραφής, εκτελούν επίσης αιτήματα δημιουργίας,διαγραφής και αντιγραφής blocks αφού τους δώσει εντολή ο NameNode.Τα DataNodes περιέχουν blocks από πληροφορίες,το κάθε block κατουσια είναι ένα τοπικό αρχείο που δημιουργεί το HDFS στο τοπικό μας μηχάνημα το μέγεθος του δεν υπερβαίνει τα 128MB οπότε όπως είναι εμφανές λειτουργεί σαν αποθήκευση. Ένα παράδειγμα για την λειτουργία του HDFS φαίνεται στην Εικόνα 3.



Εικόνα 3:Λειτουργία των NN και DN μέσα στο HDFS[2]

Επίσης τα DataNodes είναι σε συνεχή επικοινωνία με τον NameNode για το άμα χρειαστεί να γίνει οποιαδήποτε ενέργεια.Τέλος αν ο NameNode καταλάβει ότι κάποιος DataNode δεν λειτουργεί,μέσω της λήψη ενός σήματος που το ονομάζει Heartbeat,αναπροσαρμόζει τις εργασίες του συγκεκριμένου DataNode σε κάποιο άλλο DataNode που έχει τα ίδια blocks,εφόσον για το κάθε block που αποθηκεύεται χρησιμοποιεί την τεχνική του replication.Η γενική ιδέα είναι ότι κάθε αρχείο όσο μεγάλο και αν είναι αποθηκεύεται σαν μια ακολουθία από blocks.Όπως είναι λογικό όλα τα blocks(εκτός του τελευταίου έχουν 128MB) και αντιγράφεται

[2] https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

όσες φορές καθορίζει η εφαρμογή μας και έχει σαν αποτέλεσμα να μειώσει δραματικά την απώλεια δεδομένων.

Σχετική δουλειά

Μελετήσαμε δύο επιστημονικές εργασίες οι οποίες σχετίζονται θεματικά με την παρούσα διπλωματική εργασία ώστε να βρούμε τις ομοιότητες και διαφορές με την δική μας δουλειά στον χώρο της ανάκαμψης κατάστασης σε κατανεμημένα συστήματα.

Fault-tolerant Stream Processing using a Distributed, Replicated File System[3]

Η γενική ιδέα του άρθρου είναι ότι έχει υλοποιηθεί ένας τρόπος για ανοχή σφαλμάτων για κατανεμημένες μηχανές stream processing ώστε να επιταχύνει την ανάκαμψη. Ελέγχει την κατάσταση των κόμβων περιοδικά και αναδιαμορφώνει τους κόμβους που έχουν καταρρεύσει στο πιο πρόσφατο checkpoint που έχει αποθηκεύσει για αυτούς.Μπορεί να λειτουργήσει με όσο το δυνατό λιγότερους πόρους, ανεξάρτητο από το hardware χωρίς να υπάρχει επιπλέον επιβάρυνση.Βασίζεται στην τεχνική του rollback-recovery όπου κάθε κόμβος περιοδικά γράφει την κατάσταση του.Εφόσον λειτουργεί ασύγχρονα αποθηκεύει την κατάσταση σε έναν buffer χωρίς να υπάρχει κάποια ταξινόμηση αλλά κρατώντας την θέση του κάθε Page για να διαμορφώσει ξανά την προηγούμενη κατάσταση.
Οι ομοιότητες με την παρούσα εργασία ήταν ότι δεν βασίζονται στις δυνατότητες του hardware για να είναι λειτουργικά ή για να είναι αποδοτικά, επιπλέον και τα δύο καταφέρνουν ένα καλό speedup στο recovery και τέλος σπάνε τις καταστάσεις σε μικρότερα κομμάτια (Page,blocks) και τις ενώνουν για το τελικό αποτέλεσμα.

Οι διαφορές με την παρούσα εργασία ήταν ότι είναι ασύγχρονο με πολλούς κόμβους ταυτόχρονα ενώ εμείς ελέγχουμε το local speedup στο τοπικό μας μηχάνημα, ακόμα στο recover κομμάτι αποθηκεύει τις θέσεις των Pages και τις τοποθετεί ασύγχρονα ενώ εμείς ολοκληρώνουμε το κάθε recover του κάθε file, ξεχωριστά και συνεχίζουμε στο επόμενο και τέλος το save γίνεται περιοδικά ενώ στην δικιά μας την περίπτωση πρέπει να αποφασίσουμε ποια κατάσταση του μηχανήματός μας επιθυμούμε να αποθηκεύσουμε.

The Primary-Backup Approach[4]

Σε αυτή την έρευνα βλέπουμε μια προσέγγιση στο primary-backup replica, ώστε να μπορούμε να ανακάμψουμε την προηγούμενη κατάσταση σε περίπτωση που το πρωτεύον κλειδί χαθεί.Το καθιστά χρήσιμο σε συστήματα ανοχής σφαλμάτων καθώς αποθηκεύει καταστάσεις του

[3] YongChul Kwon, Magdalena Balazinska, and Albert Greenberg. 2008. Fault-tolerant stream processing using a distributed, replicated file system. Proc. VLDB Endow. 1, 1 (August 2008), 574–585. https://doi.org/10.14778/1453856.1453920

[4] Navin Budhiraja, Keith Marzullo, Fred B. Schneider, and Sam Toueg. 1993. The primary-backup approach. Distributed systems (2nd Ed.). ACM Press/Addison-Wesley Publishing Co., USA, 199–216.

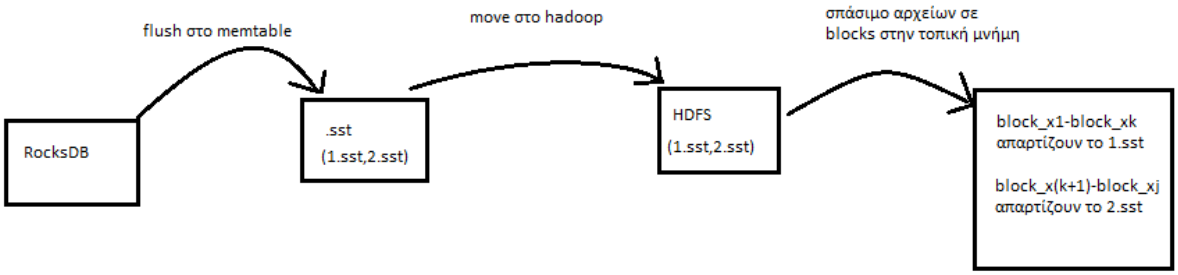
μηχανήματος. Έπειτα μας αναλύει βασικές έννοιες και πως αυτές χρειάζονται για κάθε primary-backup protocol,καθώς και πως αυτές λειτουργούν.Χρησιμοποιεί την τεχνική των bounds για να επιλέξει πόσες replicas να παράγει αναλόγως με το τι λάθη συνέβησαν προσελκύοντας το κάθε ένα με ξεχωριστό τρόπο και πόσο χρόνο θα πάρει για τα requests πριν αποφασίσει ότι απέτυχε.Επίσης ακολουθεί πρωτόκολλα διαφορετικά για κάθε περίσταση, καθώς το κάθε ένα πρωτόκολλο είναι καλό σε διαφορετικές εργασίες.

Οι ομοιότητες με την παρούσα εργασία ήταν ότι αρχικά η δουλειά αυτή ασχολείται με την διαχείριση ενός πρωτεύοντος για (ενός ή περισσοτέρων) εφεδρικών αντιγράφων, όπως κάνει και η παρούσα εργασία μεταξύ των SST files (πρωτεύοντα αντίγραφα) και των HDFS files (εφεδρικά), έπειτα υπάρχει ασφάλεια με το να αποθηκεύω την κατάσταση που ήταν το μηχάνημα μου, ελεγχος με μηνύματα τύπου “are you alive” όπως ο NN=>DN κάτι επίσης πολύ σημαντικό είναι ότι σε περίπτωση που χάσω τα δεδομένα δηλαδή έχουμε είτε με crash ή κάποιον άλλο τρόπο υπάρχει μια ανάκτηση στην κατάσταση που βρίσκονταν και τέλος και τα δύο υποστηρίζουν παράλληλη λειτουργία.

Οι διαφορές με την παρούσα εργασία ήταν ότι αυτή βασίζεται σε bounds, συγκεκριμένα στο bound of replicas για το βαθμό του replication, στο bounds of blocking time και του failover time για να δούμε αν η επικοινωνία απέτυχε, αναλόγως με το τι λάθος έγινε αντιμετωπίζει το κάθε πρόβλημα διαφορετικά.Το hadoop σπάει το κάθε αρχείο σε blocks,ενώ με την άλλη τεχνική απλά κάνει εφεδρικά. Δεν περιμένει όπως το hadoop απάντηση απλά στέλνει απάντηση στον client, εμάς η απάντηση από το hadoop είναι σημαντική, έπειτα τα requests μπαίνουν σε μια ουρά που λειτουργεί σαν FIFO σε αντίθεση με εμάς που έχουμε μόνο το request για το μεμονωμένο μηχάνημα και έχει πολλά πρωτόκολλα ενώ εμάς παραμένει το ίδιο.

Σχεδίαση

Το αρχικό μας σχέδιο ήταν εφόσον γίνουν flush τα αρχεία μας στον δίσκο του μηχανήματος μας να τα τοποθετήσουμε στο HDFS και από εκεί να κάνουμε μια αντιστοίχιση με βάση το όνομα του αρχείου και με τις πληροφορίες των blocks.



Εικόνα 4: Αλληλεπίδραση RocksDB με δημιουργία .sst, αποθήκευση στο hadoop, δημιουργία blocks με αντιστοίχιση

Αρχικά για να ξεκινήσουμε πρέπει πρώτα να δημιουργήσουμε τα sst files από την RocksDB ή το YCSB, τα οποία είναι τα δεδομένα που θα τοποθετήσουμε στο Hadoop και στην συνέχεια θα τα διαγράψουμε. Έπειτα παρατηρήσαμε ότι τα blocks που απαρτίζουν το κάθε αρχείο υπάρχουν και σε τοπικό φάκελο στο μηχάνημά μας, οπότε με ένα request πήραμε τις πληροφορίες για την χαρτογράφηση και την ακριβή θέση των blocks καθώς και το όνομα του linux αρχείου πριν την διαγραφή. Αφού τα διαγράψουμε με τον μηχανισμό του hardlink ενώσαμε το πρώτο block ώστε να μπορεί να κάνει αναφορά με το όνομα του linux αρχείου μας και στην συνέχεια κάναμε ένωση του πρώτου block με τα υπόλοιπα blocks που μας έδωσε η χαρτογράφηση. Ο αριθμός των blocks που απαρτίζουν το κάθε αρχείο αυξάνεται αναλογικά με το μέγεθος του αρχείου.

Το σύστημα το οποίο σχεδιάστηκε σε αυτή την διπλωματική εργασία αφορά την ανάκτηση της κατάστασης της RocksDB, και άρα δρα συμπληρωματικά σε ένα υπάρχον σύστημα δημιουργίας στιγμιοτύπων (snapshots) της κατάστασης αυτής στο HDFS, όπως αυτό του συστήματος Flink[5]. Σχεδιαστικά, το σύστημα το οποίο υλοποιήθηκε ενεργοποιείται όταν διαγνωστεί σφάλμα / απώλεια κατάστασης στη RocksDB, το οποίο συμβαίνει συνήθως μετά από κατάρρευση της RocksDB ή/και καταστροφή του δίσκου που αποθηκεύει τα αρχεία της. Το σύστημά μας έχει τους εξής στόχους

1. Την ταυτοποίηση του τελευταίου αποθηκευμένου στιγμιοτύπου κατάστασης στο HDFS
2. Τον εντοπισμό των τοπικών αρχείων Linux από τα οποία αποτελείται αυτή η κατάσταση
3. Την ανάκτηση (επαναδημιουργία) των αρχείων SST της RocksDB
4. Την επανεκκίνηση της RocksDB

Η καινοτομία της παρούσας διπλωματικής αφορά τον νέο τρόπο εκτέλεσης του σημείου (3).

[5] https://nightlies.apache.org/flink/flink-docs-master/docs/dev/datastream/fault-tolerance/checkpointing/

Υλοποίηση

Την υλοποίηση την έχουμε χωρίσει σε 3 βήματα για να δούμε τα αποτελέσματα σωστά σημαντικό είναι να ακολουθήσουμε τα βήματα με την σειρά που έχουν υλοποιηθεί και αναγράφονται παρακάτω.

Βήμα 1: Δημιουργία αρχείων SST

Αρχικά για να επιλέξουμε το μέγεθος των sst αρχείων μας πρέπει να έχουμε το YCSB τοπικά στον υπολογιστή μας και στην συνέχεια να δούμε στον φάκελο workloads το αρχείο workloada όπου θα βρούμε την μεταβλητή recordcount. Με αυτόν τον τρόπο μπορούμε να ελέγξουμε τον φόρτο εργασίας που θα υπάρχει. Έπειτα τρέχουμε την εντολή

./bin/ycsb load rocksdb -s -P workloads/workloada -p rocksdb.dir=/tmp/ycsb_data έτσι έχουμε

σαν αποτέλεσμα να φορτώνουμε τα δεδομένα μας θέτοντας για φάκελο της rocksdb το /tmp/ycsb_data και στην συνέχεια τρέχουμε την εντολή

./bin/ycsb run rocksdb -s -P workloads/workloada -p rocksdb.dir=/tmp/ycsb_data

όπου τρέχουμε και το benchmark μας για την δημιουργία των sst αρχείων. Έχουμε επίσης υλοποιήσει ένα πολύ βασικό script το οποίο τρέχει με τις δύο αυτές εντολές και άλλο ένα που όταν το τρέξουμε μας δείχνει καθόλη την διάρκεια που τρέχει το benchmark το τι συμβαίνει στον φάκελο που έχουμε επιλέξει για την δημιουργία των αρχείων μας. Επίσης παρατηρήσαμε ότι πολλά sst αρχεία έχουν μέγεθος 128MB κάτι που θα μας βοηθήσει στο speedup της ανάκτησης

Βήμα 2: Αντιγραφή αρχείων στο HDFS

Με το που ολοκληρωθεί η δημιουργία των αρχείων μας τρέχουμε ένα script το οποίο δημιουργεί έναν φάκελο και τοποθετεί όλα τα αρχεία στο HDFS.Για να υλοποιήσουμε αυτά τα βήματα στηριχθηκαμε στο έτοιμο cli που υπάρχει στο hadoop για την αποθήκευση και την επεξεργασία των αρχείων με την εντολή hadoop -fs put filename.Το HDFS σπάει το κάθε αρχείο σε blocks των 128 MB. Στη συνέχεια έπρεπε να βρούμε το που ακριβώς αποθηκεύονται τα blocks τα οποία το hadoop τα αποθηκεύει τοπικά σε έναν φάκελο που συνήθως είναι το tmp και μετά κατατάσσεται αναλόγως σε ποιο DataNode ανήκει το κάθε block.

Βήμα 3: Ανάκτηση αρχείων από το HDFS

Αφού ολοκληρωθεί και η αποθήκευση των αρχείων στο HDFS καθώς και η αποθήκευση των blocks στην τοπική μνήμη του μηχανήματος μας, πρέπει να δούμε πρακτικά την ανάκτηση των αρχείων μας.Αρχικά πρέπει να κάνουμε ένα τεχνητό τρόπο για την απώλεια των αρχείων και ο

πιο απλός είναι με μια απλή διαγραφή από το μηχάνημά μας.Έπειτα στέλνουμε ένα request μέσω των εντολών του cli για να έχουμε την πλήρη χαρτογράφηση για το που βρίσκεται κάθε block,καθώς και σε πιο file αντιστοιχεί. Στην συνέχεια με το hardlink ενώνουμε το κανονικό όνομα του αρχείου με το πρώτο block. Το hardlink στην ουσία είναι μια έτοιμη εντολή του linux ώστε ένα όνομα αρχείου να δείχνει αυτούσια στο πρώτο καθορισμένο block και αν γίνει οποιαδήποτε αλλαγή να ενημερωθεί κανονικά. Έπειτα χρησιμοποιούμε την εντολή cat και αντιγράφουμε κάθε στο πρώτο block φορά το επόμενο block που έχει σειρά, οπότε,όπως είναι λογικό καταλήγουμε με το πρώτο block του αρχείου να απαρτίζεται από το πλήρες μέγεθος του αρχικού μας αρχείου. Παρόλα αυτά δεν μας επηρεάζει, καθώς το HDFS κοιτάει μόνο τα πρώτα 128MB του κάθε block και σε περίπτωση που ξαναχαθουν τα αρχεία από το μηχάνημά μας ελέγχουμε αν το πρώτο block είναι πάνω από 128MB. Αν δεν ισχύει αυτό κάνουμε την ίδια διαδικασία, αν όμως ισχύει τότε κάνουμε το hardlink και συνεχίζουμε την ανάκτηση μας στο επόμενο αρχείο οπότε με αυτόν τον τρόπο δεν θα έχουμε επανάληψη πληροφορίας.

Βήμα 4: Έλεγχος

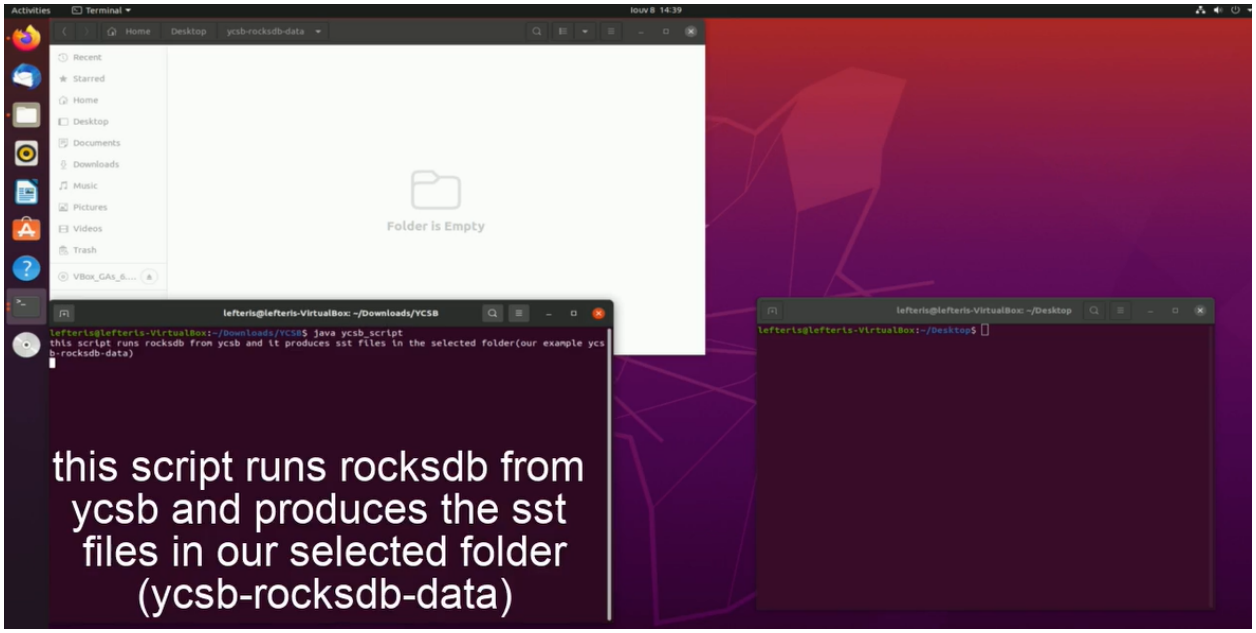
Αφού ολοκληρωθεί η ανάκτηση μπορούμε να χρησιμοποιούμε το sst_dump της RocksDB για να δούμε ότι τα αρχεία μπορούν να αναγνωριστούν κανονικά σαν sst και να διαβαστούν σαν να μην χάθηκαν καθόλου

Τυπική εκτέλεση (demo) και πειραματική αξιολόγηση

Αφού κατάλαβα πλήρως τις εντολές και πως ακριβώς λειτουργούν τα συστήματα με τα οποία έπρεπε να ασχοληθώ, να υλοποιώ τα scripts μου για δώσω με αυτοματοποίηση.

Δημιουργία αρχείων SST μέσω YCSB

Στο συγκεκριμένο στάδιο τρέχουμε το script το οποίο είναι υπεύθυνο για την δημιουργία των sst αρχείων στον επιλεγμένο φάκελο στο παράδειγμά μας ειναι Desktop/ycsb-rocksdb-data.
(Εικόνα 5)



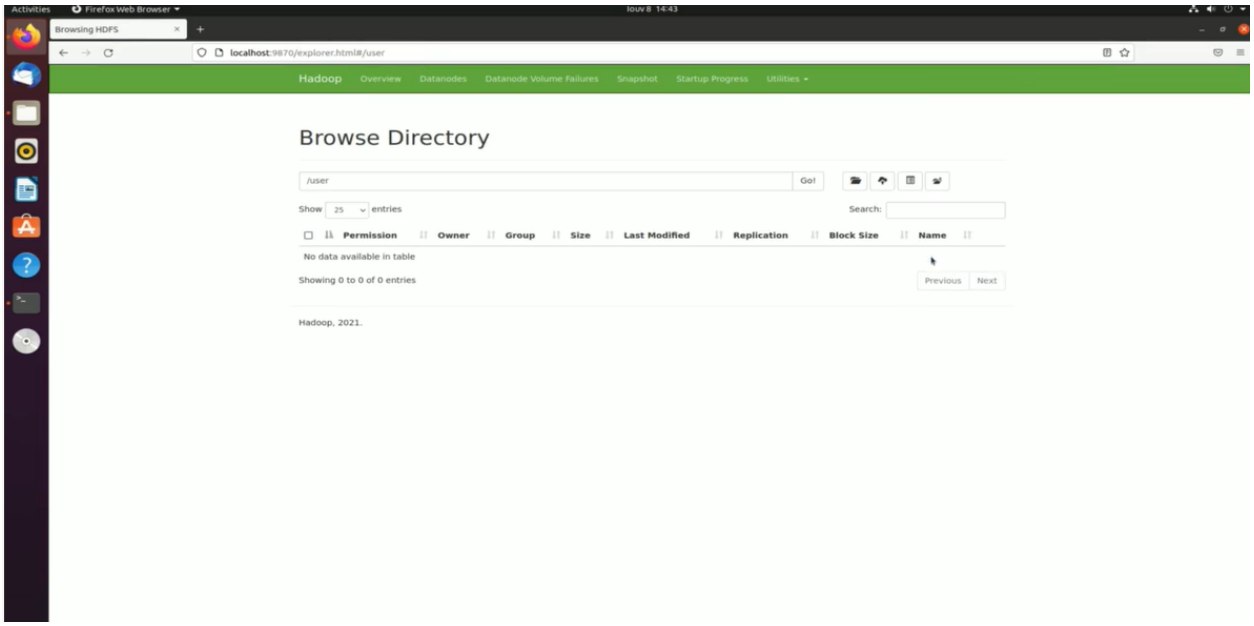
Εικόνα 5

Και εδώ βλέπουμε το τελικό αποτέλεσμα του συγκεκριμένου script.(Εικόνα 6)

Εικόνα 6

Επιβεβαίωση ότι το HDFS λειτουργεί

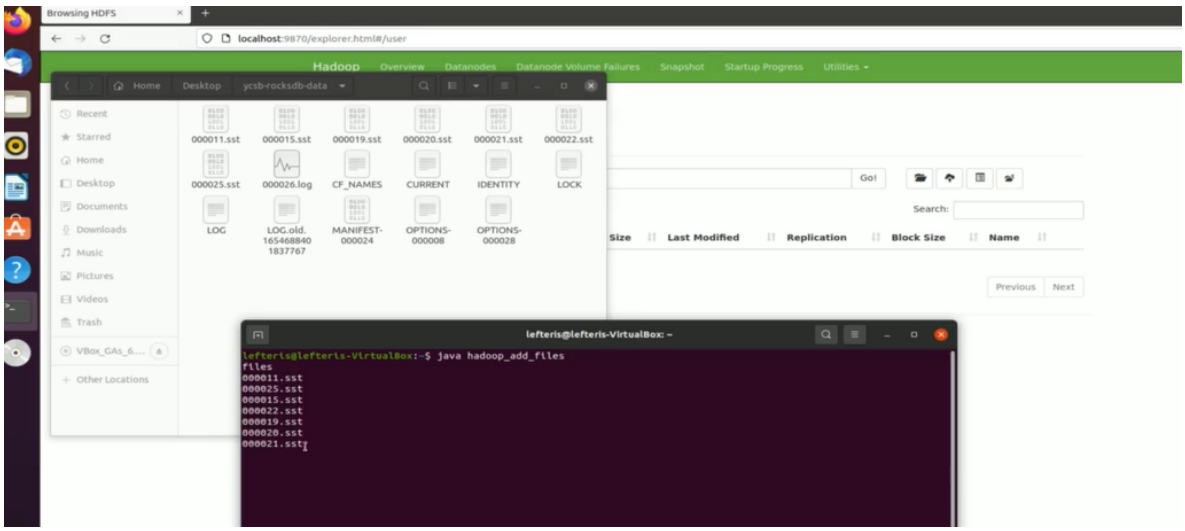
Τώρα όπως βλέπουμε στην παρακάτω εικόνα πρέπει να έχουμε ήδη το hadoop να τρέχει στο background και πατώντας στον browser μας localhost::9870 τότε πρέπει να μας βγάλει στο συγκεκριμένο interface.(Εικόνα 7)



Εικόνα 7

Επιλογή SST αρχείων

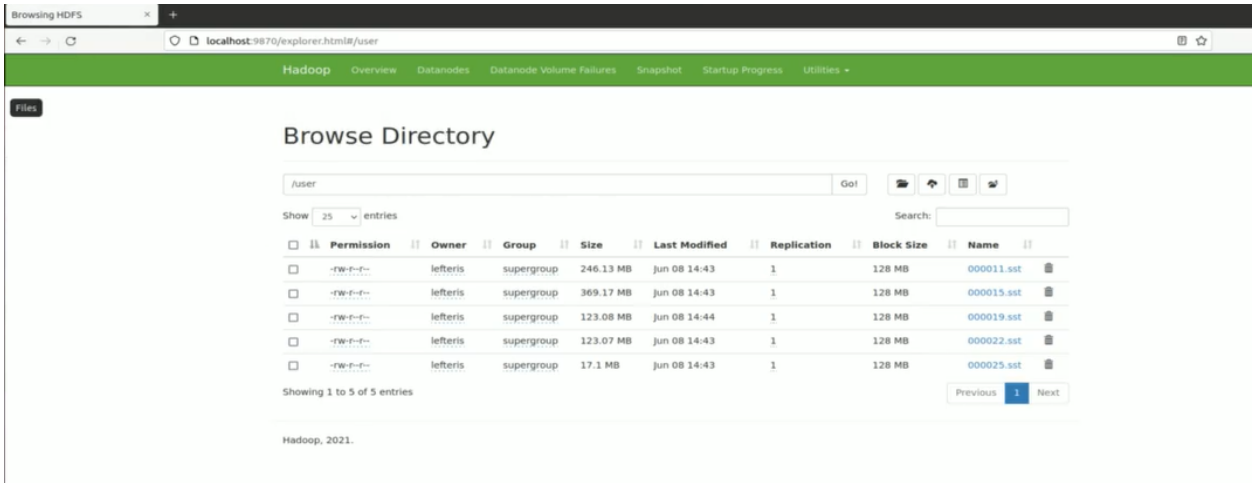
Στην συνέχεια,τρέχουμε το παρακάτω script το οποίο επιλέγει από τον φάκελό μας μόνο τα sst αρχεία (Εικόνα 8).



(Εικόνα 8)

Αντιγραφή στο HDFS

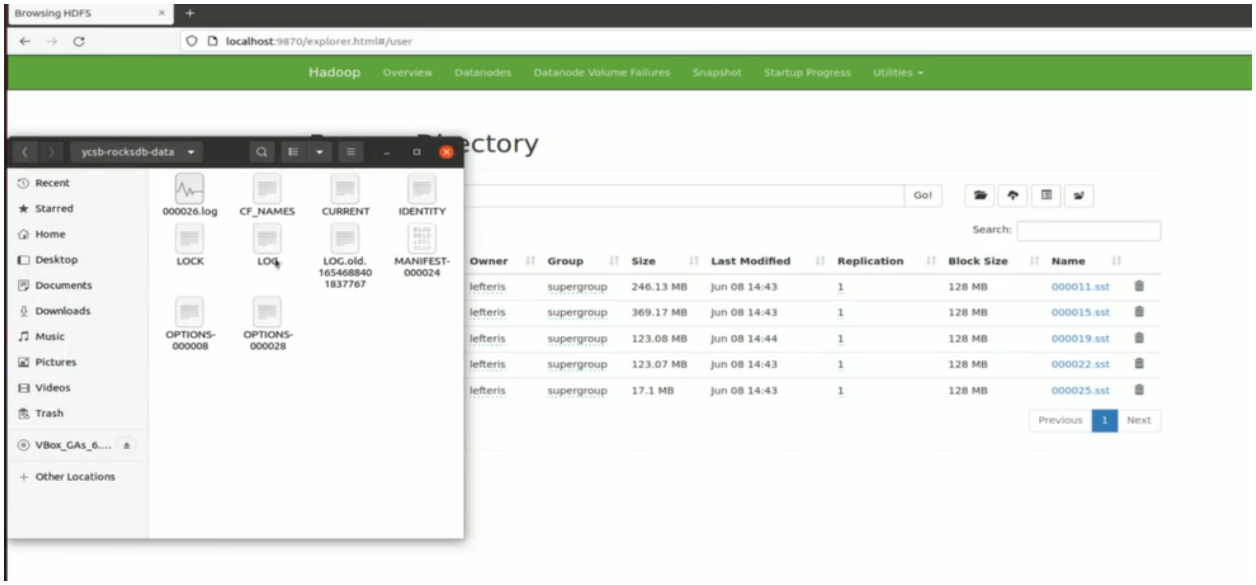
Εδώ βλέπουμε το να ανεβαίνουν τα αρχεία μας καθώς λειτουργεί το αποπάνω script στο hadoop.(Εικόνα 9)



(Εικόνα 9)

Διαγραφή αρχείων SST

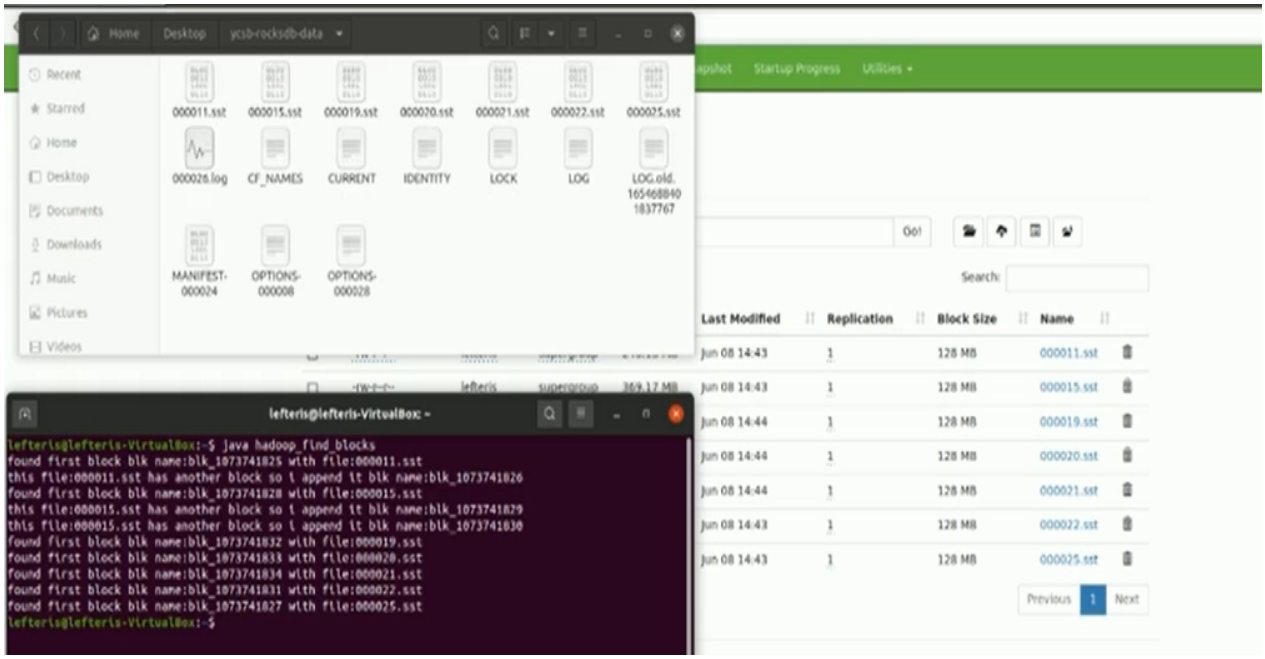
Αφού το script μας όπως βλέπουμε καταφέρει να ανεβάσει όλα τα sst αρχεία στο hadoop τότε κάνει ένα πιθανό “data loss” με το να διαγράψει όλα τα sst αρχεία από τον δίσκο του μηχανήματός μας.(Εικόνα 10)



(Εικόνα 10)

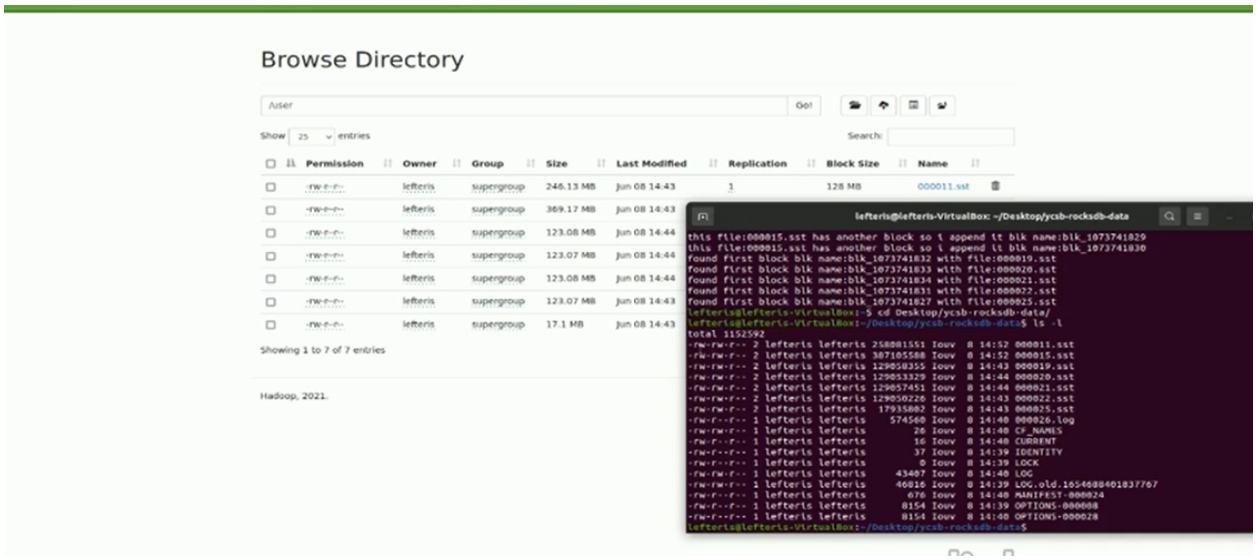
Αντιστοίχιση αρχείων HDFS σε blocks και ανάκαμψη αρχείων SST

Τέλος σε αυτό το script στέλνουμε ένα request στο hadoop να μας στείλει απλά τις αντιστοιχίες των blocks, τις οποίες τις εκτύπωσα ώστε να είναι κατανοητή και η υλοποίησή μας στο συγκεκριμένο κομμάτι (Εικόνα 11).



(Εικόνα 11)

Όπως βλέπουμε (Εικόνα 11) με το που αποθηκεύσει τις αντιστοιχίες αρχίζει και τις εκτελεί. Συγκεκριμένα αν βρει το πρώτο block το κάνει απλά hardlink με το όνομα του αρχείου στο HDFS οπότε το block με όνομα blk_1073741825 που είναι τα πρώτα 128MB του αρχείου 000011.sst θα γίνει απλά hardlink,έπειτα το συγκεκριμένο αρχείο αποτελείται από άλλο 1 block το οποίο απλά θα κάνει cat στο πρώτο block που απαρτίζεται το αρχείο μας.



(Εικόνα 12)

Στην Εικόνα 12 είχαμε πολλά αρχεία τα οποία τα τοποθετήσαμε όλα μαζί στο hadoop. Ας δούμε πρώτα όμως τους χρόνους σε ατομικά .sst αρχεία τα οποία τα τοποθετήσαμε στο HDFS. Αρχικά

δοκιμάσαμε να τα ανακτήσουμε με την εντολή copyToLocal και στην συνέχεια με το script που φτιάξαμε.

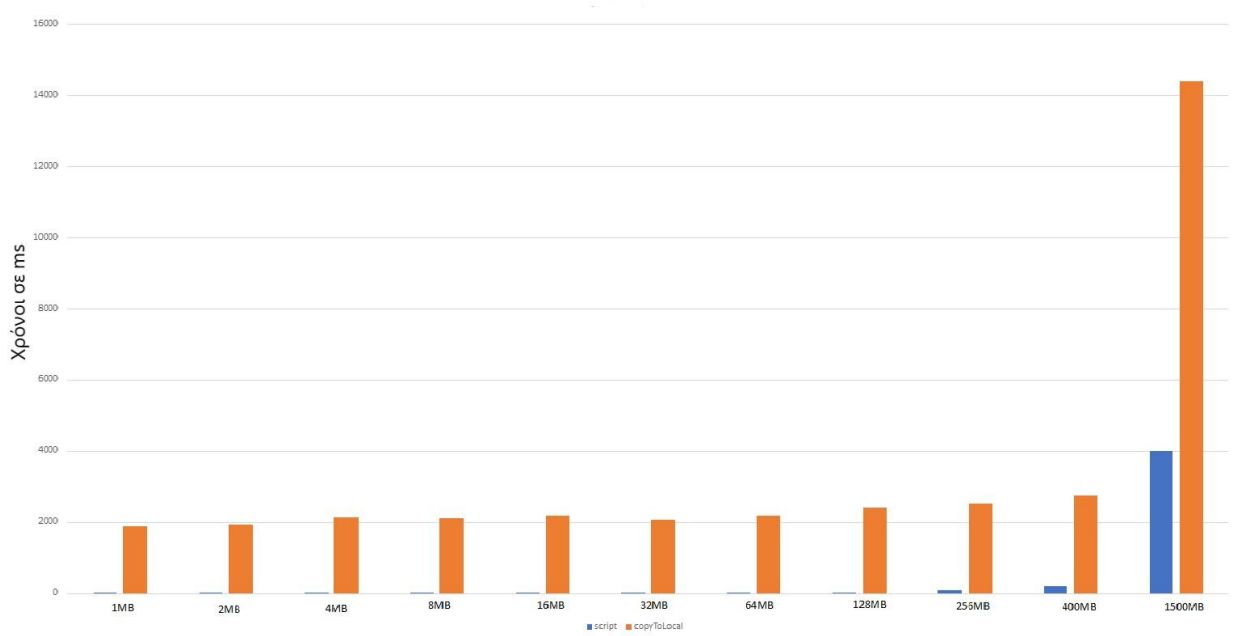
Πειραματικά αποτελέσματα

Αναφορικά στους χρόνους μας έχουμε υπολογίσει μόνο από το σημείο που αρχίσαμε να εκτελούμε το copyToLocal και στην δική μας περίπτωση από όταν άρχισε το hardlink με το cat. Όλοι οι χρόνοι είναι μετρημένοι σε ms.(Πίνακας 1)

Μέγεθος αρχείων (MB)	Χρόνοι με script (ms)	Χρόνοι με copyToLocal (ms)	Speedup
1	5	1876	375,2
2	5	1932	386,4
4	3	2138	712,6
8	6	2102	350,3
16	5	2168	431,6
32	5	2071	414,2
64	5	2179	435,8
128	5	2410	482
256	76	2423	31,8
400	190	2638	13,8
1500	3989	14384	3,6
5400	112114	134730	1,2

Πίνακας 1:Σύνοψη μεγεθών, χρόνων και speedup

Παρατηρούμε ότι το speedup είναι πολύ μεγάλο στα μικρά .sst αλλά μειώνεται όσο το αρχείο μας μεγαλώνει. Όλοι οι χρόνοι είναι από την μνήμη καθώς τα προσπελάσαμε ενώ τα γράψαμε στο HDFS, όμως όσο το μέγεθος των αρχείων μας μεγαλώνει τότε είναι πολύ πιθανό η RAM να διαγράφει τα αρχεία μας επειδή ακολουθεί την πολιτική LRU, οπότε μετά από ένα σημείο δεν μπορούμε να μιλήσουμε με σιγουριά. Το δικό μας script από την άλλη όπως ήταν αναμενόμενο χρειάζεται 5ms για να κάνει restore αρχείο το οποίο δεν υπερβαίνει τα 128MB, αφού αποτελείται από ένα block και το μόνο που χρειαζόμαστε είναι το hardlink. Στη συνέχεια (άνω των 128MB) ο χρόνος αυξάνεται γιατί με την εντολή cat αντιγράφει τα δεδομένα, ωστόσο και πάλι η διαφορά στο speedup είναι αισθητή.Το πρόβλημα όμως είναι ότι όσο μεγαλώνει το αρχείο το speedup μειώνεται με αποτέλεσμα στα 5000MB να φτάνει κοντά στο 1 και να έχουμε speedup 20%. Ακολουθεί διάγραμμα.



Διάγραμμα 1: Χρόνοι ανάκτησης ενός αρχείου αυξανόμενου μεγέθους

Εδώ βλέπουμε(Διάγραμμα 1) σε μορφή bar chart τους χρόνους που χρειάστηκε το script μας (μπλε) και το copyToLocal (πορτοκαλί) για την ανάκτηση των δεδομένων μας με ms.

Σαν επόμενο στάδιο ασχοληθήκαμε με την ανάκτηση πολλών αρχείων ταυτόχρονα. Κάθε φορά μεγαλώναμε το recordcount για να μας παράγει περισσότερα και μεγαλύτερα αρχεία.

Για recordcount ίσο με 500.000 δημιουργήθηκαν:

4 SST αρχεία μεγέθους 128MB [6]
1 SST αρχείο μεγέθους 70MB
Σύνολο: 582MB

Με το copytoLocal χρειαστήκαμε 11772ms
Με το script χρειαστήκαμε 15ms
SpeedUp: **784,8**

Για recordcount ίσο με 1.000.000 δημιουργήθηκαν:

4 SST αρχεία μεγέθους 128MB
1 SST αρχείο μεγέθους 256MB (το αντίστοιχο HDFS αρχείο θα έχει δύο γεμάτα blocks)

[6] Παρατηρούμε ότι συμπίπτει το αρχικό μέγεθος SST που γράφει το RocksDB (δηλ. το μέγεθος memtable) με το block size του HDFS, και τα δύο είναι 128MB

1 SST αρχείο μεγέθους 400MB
1 SST αρχείο μεγέθους 17MB
Σύνολο:1.185MB

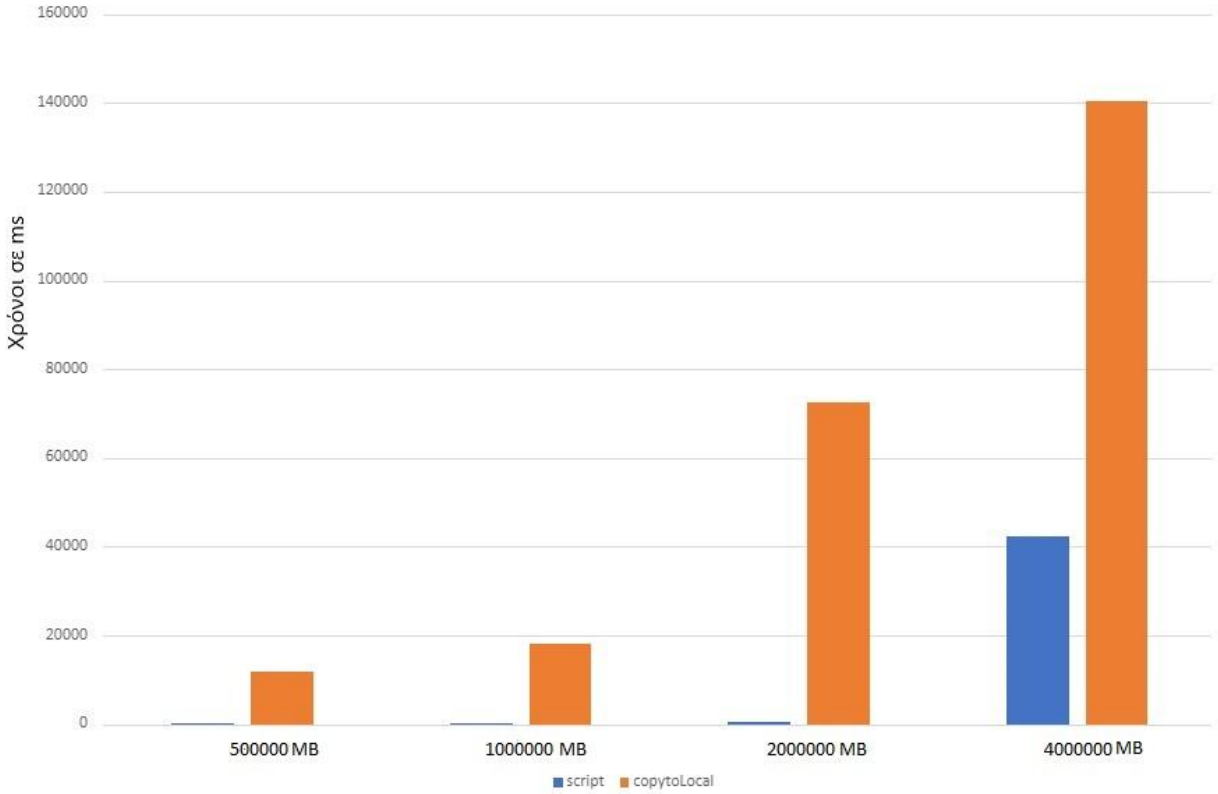
Με το copytoLocal χρειαστήκαμε 17991ms
Με το script χρειαστήκαμε 329ms
SpeedUp: **54,6**

Για recordcount ίσο με 2000000 δημιουργήθηκαν:
9 SST αρχεία μεγέθους 67MB
5 SST αρχεία μεγέθους 128MB
2 SST αρχεία μεγέθους 256MB
2 SST αρχεία μεγέθους 400MB
1 SST αρχείο μεγέθους 40MB
Σύνολο: 2.595MB

Με το copytoLocal χρειαστήκαμε 72408ms
Με το script χρειαστήκαμε 583ms
SpeedUp: **124,2**

Για recordcount ίσο με 4.000.000 δημιουργήθηκαν:
19 SST αρχεία μεγέθους 67MB
3 SST αρχεία μεγέθους 128MB
2 SST αρχεία μεγέθους 256MB
4 SST αρχεία μεγέθους 400MB
1 SST αρχείο μεγέθους 70MB
1 SST αρχείο μεγέθους 6MB
1 SST αρχείο μεγέθους 1400MB
Σύνολο: 5.245MB

Με το copytoLocal χρειαστήκαμε 140295ms
Με το script χρειαστήκαμε 42241ms
SpeedUp: **3,32**
(εδώ το speedup συμπίπτει με το speedup που έχουμε όταν το αρχείο μας είναι 1500MB (προτελευταία γραμμή του Πίνακα 1) και ο λόγος είναι ότι τα υπόλοιπα αρχεία θα μας πάρουν κάτι ms καθώς θα χρειαστούμε μόνο ένα hardlink για να τα επαναφέρουμε ενώ το αρχείο που είναι 1400MB θα χρειαστεί ένα hardlink και στην συνέχεια να κάνουμε cat τα υπόλοιπα blocks στο αρχικό. Συγκεκριμένα αποτελείται από 11 blocks, από τα οποία το 1 θα γίνει hardlink και τα 10 cat, που είναι και ο λόγος που μας αυξάνονται οι χρόνοι)



Διάγραμμα 2:Χρόνοι ανάκτησης με πολλά αρχεία

Αρχικά παρατηρούμε ότι το speedup είναι πολύ μεγάλο στην περίπτωση που έχουμε αρχεία τα οποία αποτελούνται από 1 block, δηλαδή το συνολικό τους μέγεθος είναι κάτω από 128MB καθώς το μόνο που αρκεί για το recovery είναι ένα hardlink. Χωρίς να μας ενδιαφέρει το πόσα πολλά είναι. Επίσης εάν έχουμε μικρά αρχεία και ένα μεγάλο αρχείο τότε το συνολικό speedup εξαρτάται από το πόσο μεγάλο είναι αυτό το αρχείο. Τέλος αν έχουμε αρχεία τα οποία είναι αρκετά GB τότε το speedup θα πλησιάζει το 1, και ο λόγος είναι ότι θα κάνουμε πάρα πολλά cat, οπότε θα έχουμε αυτούσια αντιγραφή.

Συμπεράσματα

Η έμπνευση για αυτή τη διπλωματική εργασία ήρθε από την παρατήρηση ότι η ανάκτηση αρχείων SST της RocksDB από τοπική replica χρησιμοποιούσε αντιγραφή (copy) δεδομένων, κάτι το οποίο μπορούσε να γίνει –όπως και επετεύχθη– αρκετά γρηγορότερο για τον λόγο ότι είχαμε αυτούσιο το αρχείο στο μηχάνημα μας (χωρισμένο ωστόσο σε blocks, διαφορετικά αρχεία Linux, των 128MB). Χρησιμοποιήσαμε την παραδοσιακή εντολή του Unix hardlink για αρχεία μικρότερα των 128MB και την συνδιάσαμε με την εντολή cat όταν το αρχείο ήταν μεγαλύτερο από αυτό το μέγεθος. Κάτι το οποίο σε κάποιες περιπτώσεις μας έδωσε θεαματικά αποτελέσματα speedup ειδικά σε περιπτώσεις που είναι ελάχιστα ή πολλά μικρά αρχεία τα οποία πρέπει να ανακτήσουμε, από την άλλη έχουμε όμως αξιοσημείωτα αποτελέσματα για μεγάλα αρχεία. Είναι σημαντικό να σημειωθεί ότι σε καμία περίπτωση οι χρόνοι μας δεν είναι χειρότεροι από τους χρόνους της εντολής copyToLocal κάτι το οποίο δείχνει ότι η προσέγγισή μας δεν προκαλεί καμία επιβάρυνση συγκριτικά με την παραδοσιακή λύση της αντιγραφής αρχείων. Μια μελλοντική δουλειά που θα έκανε το πρόγραμμά μας ακόμα πιο γρήγορο θα ήταν μια επέκταση της εντολής hardlink για την περίπτωση που θέλουμε αντιστοίχιση ενός προς πολλά αρχείων, το οποίο θα βελτίωνε την περίπτωση που θέλουμε να ανακτήσουμε ένα αρχείο SST από πολλά blocks. Έτσι η σύνδεση του αρχείου θα ήταν άμεση (χωρίς καθόλου copies) και το speedup θα αυξανόταν ακόμα περισσότερο.