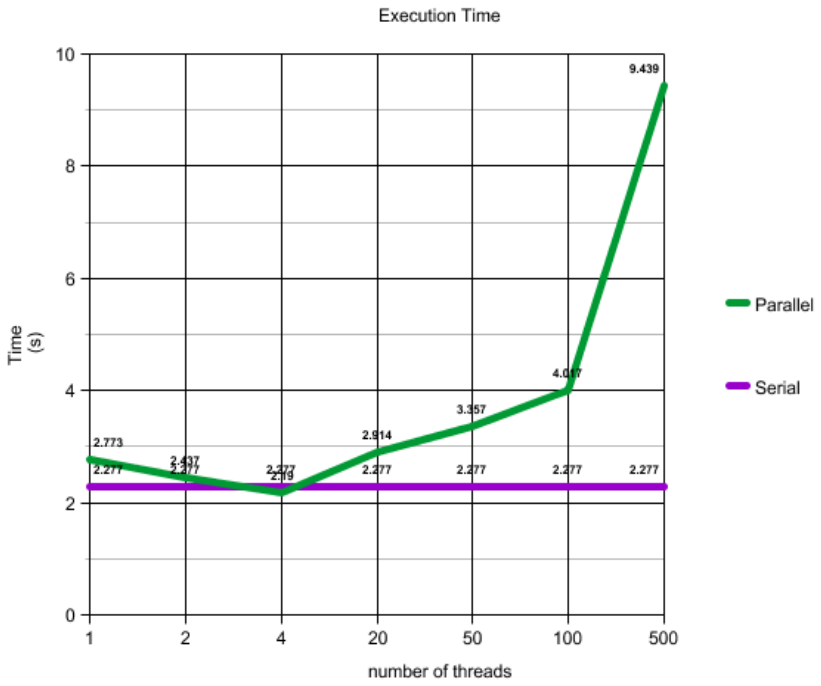
Ανάπτυξη Λογισμικού για Πληροφοριακά Συστήματα

Μετρήσεις

Εκτελώντας το part2 και το part3 παρουσιάστηκε μικρή βελτίωση στον χρόνο όπως φαίνεται από το παρακάτω διάγραμμα:



Σχετικά με την κατανάλωση μνήμης για τα 2 τελευταία part με valgrind δεσμεύτηκε η εξής μνήμη:

- part2: 1,620,510,297 bytes allocated
- part3: 1,480,973,679 bytes allocated

Παρατηρείται ότι στο part3 δεσμεύτηκε λιγότερη μνήμη

Μετρώντας τον χρόνο εκτέλεσης του κάθε thread-job για ένα join χρειάστηκαν σε second χρόνος:

- Histogram_thread: 0.001384
- NewRel_thread: 0.001679
- Bucket_thread: 0.010682

Είναι λογικό το Bucket_thread να χρειάζεται περισσότερο χρόνο αφού είναι αρκετά μεγαλύτερος ο αριθμός των jobs που απαιτείται να επεξεργαστεί και είναι πιο χρονοβόρα η δημιουργία των ευρετηρίων (bucket,chain).

Το πρόγραμμα εκτελείται κοντά σε 2 second.

Οι μετρήσεις έγιναν σε HP Pavilion 15 laptop AMD A10-9620P RADEON R5, 10 COMPUTE CORES 4C+6G

Part 1

Οργάνωση αρχείων:

- files-creation:
  file_creation.c δημιουργεί sample αρχείου με τυχαίους αριθμούς σε συγκεκριμένο range. Υπάρχει makefile και εντολή εκτέλεσης της μορφής
  ./create filename num_of_records min max
  1.txt 2.txt δύο sample αρχείων
- RHJ
  : main.c εκτελεί-τεστάρει την συνάρτηση RHJ
  RadixHashJoin.c υλοποιεί την RHJ
  Τα υπόλοιπα αρχεία είναι βοηθητικά.Υπάρχει makefile και η εκτέλεση της main είναι της μορφής
  ./run file1 file2
  π.χ. ./run ../files-creation/1.txt ../files-creation/2.txt > res.txt

Τρόποι υλοποίησης RHJ:

- Αρχικά δημιουργούνται οι δύο σχέσεις (relation) όπου διαβάζεται το κάθε αρχείο και αποθηκεύνται τα απαραίτητα στοιχεία payload και key.Τα key αρχίζουν από 1.
- Έπειτα επιστρέφεται στην μεταβλητή Result το αποτέλεσμα της συνάρτησης RadixHashJoin.
  - Τέλος αυτό εκτυπώνεται για επαλήθευση του αποτελέσματος

- Η συνάρτηση FirstHash (n τελευταία bits) δημιουργεί το ιστόγραμμα και επιστέφει το νέο relation οργανωμένο σε buckets

  - SecondHash k bit αριστερά μετά τα πρώτα n

- Συνολικά διατρέχονται όλοι οι κάδοι και για αυτόν με μικρότερο μέγεθος εγγραφών δημιουργείται η δομή HashBucket που περιέχει δύο πίνακες chain και bucket

- Συγκεκριμένα αυτοί αρχικοποιούνται με -1 και έτσι αν στον buket περιεχόμενα με αυτή την τιμή δηλώνουν άδεια ενώ στον chain όσες θέσεις έχουν -1 δείχνουυ ν το τέλος της αλυσίδας.Χρησιμοποιήθηκε το -1 και όχι το 0 για την δήλωση του τέλους

- Η ScanBuckets λαμβάνοντας υπ'όψιν από την μία σχέση όλο το bucket και από την άλλη την δομή HashBucket, διατρέχονται τα στοιχεία του bucket και για καθένα από αυτά χρησιμοποιώντας την τιμή επιστροφής της HashFunction διατρέχεται η αλυσίδα για σύγκριση των payload.Αν είναι ίδια αποθευκεύονται τα keys στην δομή Result

- Η δομή αυτή είναι ένα struct με το μέγεθος της λίστας και με δείκτη στην αρχή και το τέλος της λίστας κόμβων τύπου result_node

Part 2

Οργάνωση αρχείων:

-FullRelation.c οι δομές δεδομένων και οι συναρτήσεις που χρησιμοποιούνται για την αποθήκευση των σχέσεων στην μνήμη
-Sql_queries.c δομές δεδομένων και συναρτήσεις για την εκτέλεση των επερωτήσεων και εμφάνιση των αποτελεσμάτων
Τα υπόλοιπα αρχεία ειναι βοηθητικά

Στο part1 άλλαξε στη δομή tuple ο τύπος του payload σε uint64_t και προστέθηκαν συναρτήσεις στο RadixFunctions.c για την δημιουργία Result στις περιπτώσεις που δύο σχέσεις ενός join βρίσκονται στα ενδιάμεσα αποτελέσματα,μία κολώνα σχέσης συγκρίνεται με κάποιον αριθμό σύμφωνα με κάποιον τελεστή και στην ισότητα δύο κολωνών ίδιας σχέσης

Υπάρχει makefile και η εκτέλεση της main και είναι της μορφής:
./run relations_file queries
π.χ. ./run ../small/small.init ../small/small.work

Αλγόριθμος υλοποίησης-Παραδοχές:

- Στην αποθήκευση των σχέσεων στην μνήμη χρησιμοποιείται για κάθε σχέση ένας πίνακας τύπου relation μεγέθους όσες οι κολώνες της σχέσης.Για κάθε θέση του πίνακα ο δείκτης tuples δείχνει σε ένα σημείο ενός πίνακα κοινού για όλες τις κολώνες συμβολίζοντας την αρχή των δεδομένων για αυτή την κολώνα.Δεν χρησιμοποιήθηκε ένας απλός δείτης αλλά ένα realtion αφού η συνάτηση RHJ παίρνει ως όρισμα relation *

- Ως metadata χρησιμοποιείται η min και max τιμή για κάθε κολώνα,ο αριθμός των κολωνών και το μέγεθος των γραμμών της σχέσης

- Για κάθε query που έρχεται αποθηκεύονται οι πληροφορίες του FROM,WHERE,SELECT σε κατάλληλους πίνακες.Για το FROM δημιουργείται πίνακας μεγέθους όσες οι σχέσεις που συμμετέχουν με κάθε θέση του να είναι ένας δείκτης στον αρχικό πίνακα όλων των σχέσεων αντιπροσωπεύοντας την κάθε σχέση του query.Από αυτόν δημιουργείται ένας πίνακας ίσου μεγέθους με στοιχεία αντίγραφα από ολόκληρη την δομή για κάθε σχέση.Δεν τροποποιείται η αρχική δομή γιατί πρέπει να χρησιμοποιηθεί για τα επόμενα queries του ίδιου batch

- Για την εκτέλεση των κατηγορημάτων για κάθε ένα υπολογίζεται μία μετρική.Έτσι πρώτα υπολογίζονται αυτά που περιέχουν >,<,= με αριθμό η ισότητα κολωνών ίδια σχέσης και έπειτα τα join

- Για τα ενδιάμεσα αποτελέσματα υπάρχει ένας πίνακας πινάκων keys.Κάθε θέση του αντιπροσωπεύει μία σχέση της επερώτησης και αντισοιχεί σε έναν πίνακα με τα κλειδιά που έχουν φιλτραριστεί.Αρχικά,πριν την εκτέλεση των κατηγορημάτων κάθε θέση του keys είναι NULL.Σε όλες τις περιπτώσεις εκτός του join αλλάζουν τα κλειδιά της σχέσης. Αλλιώς σε join πρώτα αλλάζουν τα κλειδιά της σχέσης που ήδηυπάρχουν στον keys με αντίστοιχη αλλαγή όλων των υπολοίπων σχέσεων και έπειτα τοποθετούνται τα κλειδιά της άλλης σχέσης

- Όταν γίνει κάποιο join πρέπει να φιλτραριστούν εκτός από τα κλειδιά των σχέσεων που συμμετέχουν, και τα κλειδιά αυτών που έχουν συνδεθεί σε προηγούμενο κατηγόρημα με αυτά.Π.χ. 0.1=1.2 && 1.3=3.0, στο δεύτερο join δεν θα φιλτραριστούν μόνο τα κλειδιά των σχέσεων 1,3 αλλά και της 0.Στον κώδικα λαμβάνονται όλες οι περιπτώσεις άσχετα της σειράς των κατηγορημάτων.

- Στην εκτέλεση ενός κατηγορήματος αν οι σχέσεις που συμμετέχουν δεν έχουν κλειδιά στα ενδιάμεσα αποτελέσματα τότε λαμβάνονται τα relation από τον υποπίνακα της αρχικής δομής.Αλλιώς δημιουργείται για κάθε σχέση ένα relation όπου τα keys αντιστοιχούν στα index του πίνακα keys[i] αρχίζοντας από 1.Έτσι στο Result τα στοιχεία των πινάκων που θα προκύψουν μειωμένα κατά 1 δείχνουν ποιές θέσεις από τον πίνακα keys[i] πρέπει να κρατηθούν και ποιά να σβηστούν

- Μετά από κάθε κατηγόρημα υπολογίζεται το άθροισμα καθεμιάς κολώνας που απαιτείται και τοποθετείται σε μία λίστα.Όταν τοποθετηθεί το τελευταίο για το query το πεδίο flag γίνεται -1 για αλλαγή γραμμής.Έτσι στο τέλος κάθε batch εκτυπώνονται τα αποτελέσματα για τα queries

Σημαντικά:

Στο αρχείο small.init προσθέσαμε Done στην τελευταία γραμμή

Το εκτελέσιμο εμφανίζει τα αποτελέσματα όπως υπάρχουν στο αρχείο small.result

Υπάρχει έλεγχος συναρτήσεων με unit testing

Part 3

Οργάνωση αρχείων:

-RHJ/ThreadFunctions.c συναρτήσεις που χρησιμοποιούν τα threads
- RHJ/Threads.c οι συναρτήσεις των thread (..
- Queries/statistics.c συναρτήσεις σχετικές με τον υπολογισμό και την ενημέρωση των στατιστικών για κάθε κολώνα
-Queries/statistics.c/enumeration συνάρτηση που επιστρέφει τον καλύτερο συνδυασμό για την σειρά εκτέλεσης των predicate
-Queries/statistics.c/update_metadata_array συνάρτηση που αλλάζει τα στατιστικά των σχέσεων που συμμετέχουν στο predicate
Τα υπόλοιπα αρχεία ειναι βοηθητικά

Στο Queries/Queries.h προστέθηκε στην δομή statistics το πεδίο count που δείχνει τον αριθμό των distinct τιμών της κολώνας.

Από το part2 άλλαξε στο RHJ φάκελο η υλοποίηση του αλγορίθμου με χρήση threads για παραλληλοποίηση και το μέγεθος του buffer στην λίστα των αποτελεσμάτων από 1024 σε 128. Στον Queries φάκελο καλείται η συνάρτηση enumeration για το υπολογισμό της βέλτιστης σειράς των predicate με λογική παρόμοια της εκφώνησης

Υπάρχει makefile και η εκτέλεση της main και είναι της μορφής:
./run relations_file queries
π.χ. ./run ../small/small.init ../small/small.work

Αλγόριθμος υλοποίησης-Παραδοχές:

Περισσότερα σχόλια βρίσκονται μέσα στα αρχεία

- Παραλληλοποίηση:χρησιμοποιούνται semaphores και mutexes για την υλοποίηση. Υπάρχουν 3 threads για τα τρία στάδια και έτσι γίνονται 3 φορές pthread_create και pthread_exit. Ο λόγος είναι ότι κάθε thread για να ξεκινήσει χρειάζεται να χρησιμοποιήσει δεδομένα που απαιτούν τον τερματισμό προηγούμενων thread για την υλοποίησή τους. Η λειτουργία του JobSheduler εκτελείται από την συνάρτηση RadixHashJoin. Υπάρχει μία λίστα με Jobs που σε αυτήν γίνεται εισαγωγή και ένα μόνο thread το εξάγει κατεβάζοντας τον mutex της λίστας και εκτελεί την κατάλληλη δουλειά. Κατά την εισαγωγή γίνεται sem_post για την αύξηση του σημαφόρου. Έτσι τα threads εκτελούνται όσο υπάρχουν στοιχεία στη λίστα και ο τερματισμός τους γίνεται όταν είναι αληθής η συνθήκη shutdown==1 και η λίστα τψν Jobs είναι άδεια.

  -Histogram_thread: Γνωρίζοντας τα όρια από την σχέση που πρέπει να διαβάσει δημιουργεί το δικό του ιστόγραμμα και αποθηκεύει στο κεντρικό ιστόγραμμα τις τιμές του.
  -NewRel_thread: Σύμφωνα με τα όρια που γνωρίζει και το αθροιστικό ιστόγραμμα από την παλιά σχέση γράφει κάθε δεδομένο στο κατάλληλο bucket.
  -Bucket_thread: Λαμβάνοντας το index του bucket εισάγει στο αποτέλεσμα τα κατάλληλα κλειδιά.
- Καλύτερος συνδυασμός predicate:Παίρνοντας ως δεδομένο ότι σε κάθε query υπάρχει ακριβώς ένα φίλτρο στην τελευταία θέση των predicate δημιουργούμε έναν πίνακα μεγέθους όσα τα predicates. Κάθε γραμμή είναι μία πιθανή σειρά εκτέλεσης των predicates (πίνακα από int που δηλώνουν τα index των predicate . Για όλες τις γραμμές βάζουμε στην πρώτη στήλη το φίλτρο (το predicate που θα εκτελεστεί πρώτο). Έπειτα τοποθετείται στην δεύτερη στήλη το index όλων των υπολοίπων predicate. Τέλος για τις υπόλοιπες στήλες κρατάμε την καλύτερη επιλογή και αφού γεμίσει ο πίνακας διαλέγουμε τη γραμμή με τα λιγότερα ενδιάμεσα αποτελέσματα.

Υπάρχει βελτιωμένη έκδοση unit testing

Φοιτητές:
Καλογερόπουλος Ιωάννης 1115201500057
Κότσι Ηρακλή 1115201500073
Παπασωτηρίου Ηλίας 1115201500123