Τμήμα Πληροφορικής & Τηλεπικοινωνιών

K23α - Ανάπτυξη Λογισμικού Για Πληροφοριακά Συστήματα

Τρίτο Μέρος

Χειμερινό Εξάμηνο 2019 – 2020

Καθηγητής Ι. Ιωαννίδης

Περιγραφή λειτουργικότητας

Αποθήκευση Δεδομένων

- Διαβάζουμε το αρχείο με τις σχέσεις και το αποθηκεύουμε στη δομή all_data που είναι ένας πίνακας με δείκτες στη δομή relation_data. Κάθε relation_data έχει δείκτες στην αρχή κάθε column τη σχέσης. Κάθε φορά που διαβάζουμε νέα σχέση η δομή all_data επεκτείνεται κατά 1.
- Για κάθε σχέση αποθηκεύουμε τις πληροφορίες l, u ,f, d που ζητούνται. Η τιμη l αναφέρεται στην μικρότερη τιμή της εκάστοτε στήλης . Η τιμή u αναφέρεται στην μεγαλύτερη τιμή, η τιμή f αναφέρεται στο πλήθος των δεδομένων της και τέλος η τιμή d πλήθος των μοναδικών τιμών της. Όταν ο χρήστης πατήσει "Done" τελειώνει και η ανάγνωση τον σχέσεων.

- Το αρχείο με τα Queries αποθηκεύεται στη δομή Batches η οποία κρατάει όλα τα Batches και έχει δείκτες στη δομή Batch_lines. Κάθε batch_line είναι και ένα ξεχωριστό Query.

- Οι σχέσεις, τα κατηγορήματα και οι προβολές αποθηκεύονται σε ξεχωριστές δομές (predicates, check sums) όπου εκεί κρατάμε και τις πληροφορίες που χρειάζονται κάθε φορά, όπως το σε ποιά σχέση αναφερόμαστε κάθε φορά(rel_origin) και ποια είναι η αντίστοιχή της στο ερώτημα(rel_alias).

- Ύστερα δημιουργούμε τον Job Scheduler. Ο JS είναι μια λίστα με τη λογική μιας ουράς αναμονής. Κάθε κόμβος της λίστας περιέχει δείκτες σε query_jobs, join jobs ή sort_jobs. Κάθε φορά που διαβάζουμε ένα καινούργιο Query αυτό γίνεται push στον Job Scheduler. Μόλις ολοκληρωθεί ένα batch, threads που τα ορίζουμε με #define MAX THREADS αναλαμβανουμε την παράλληλη εκτέλεση των Queries. Οταν ολοκληρώνεται κάποιο Query βάζει το αποτέλεσμά του στην κατάλληλη θέση ενός πίνακα, δηλαδή αν πρώτο τελείωσε το Query 3 θα βάλει το αποτέλεσμά του στη θέση 3. Οταν η λίστα

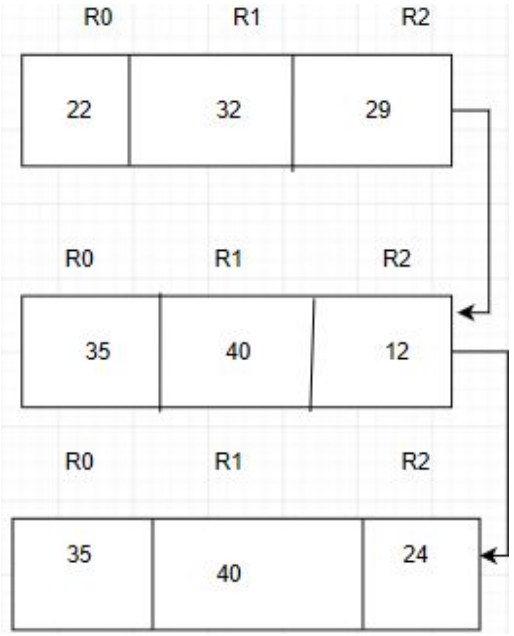
του Job Scheduler αδειάσει σημαίνει ότι το Batch ολοκληρώθηκε οπότε απλά. εκτυπώνουμε τα αποτελέσματα του πίνακα σειριακά.

Εκτέλεση Ερωτημάτων

Αρχικά εκτελούνται τα φίλτρα των ερωτημάτων για να μειωθεί το μέγεθος των πινάκων. Τα rowid των σχέσεων που φιλτράρονται αποθηκεύονται στη δομή Between, στον πίνακα farrays

Υλοποίηση με λίστα

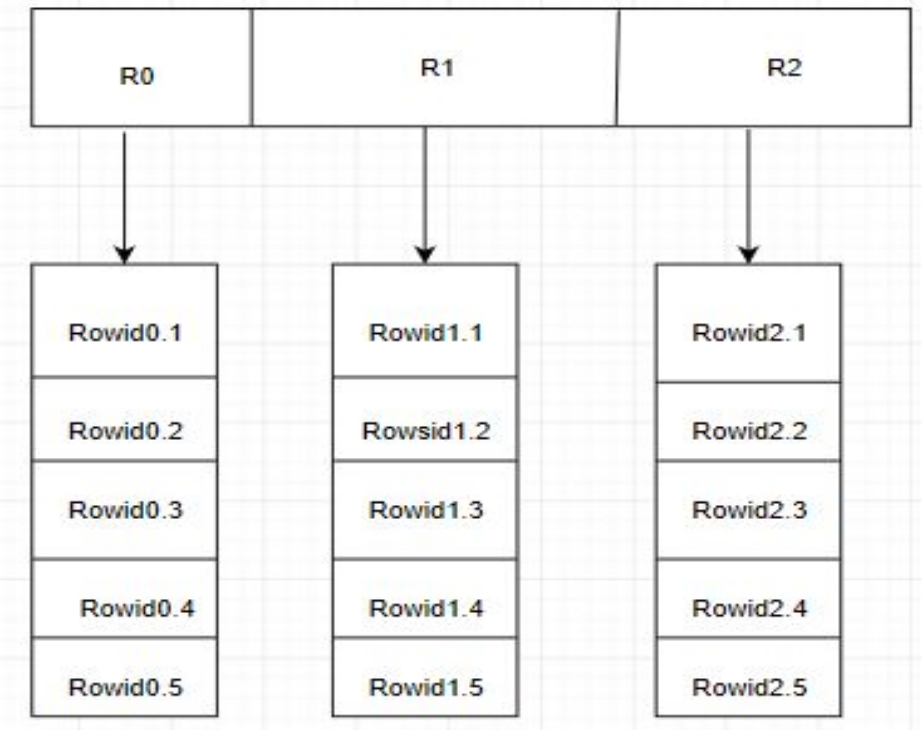
Αντίστοιχα στην ίδια δομή, κρατάμε ένα δείκτη στην δομή result_list που είναι μια λίστα που κρατάει τα ενδιάμεσα αποτελέσματα που προκύπτουν απο τις ζεύξης μεταξύ των σχέσεων. Κάθε κόμβος της λίστας είναι ένας πίνακας από ακέραιους που κρατάνε τα rowid των σχέσεων που γίνονται join . Ο πίνακας έχει τόσες θέσεις όσες ο αριθμός των relations.



Με αυτή τη δομή το πρώτο στοιχείο του πίνακα κάθε κόμβου της λίστας αναφέρεται στα rowid της σχέσης 1, το δεύτερο στοιχείο της σχέσης 2 κ.ο.κ.

Για το small αρχείο η συγκεκριμένη υλοποίηση έτρεχε σε ικανοποιητικό χρόνο, σε 54 δευτερόλεπτα. Όμως, για το medium αρχείο λόγω του μεγάλου μεγέθους των αρχείων έπρεπε να γίνονται εκατομμύρια mallocs() σε κάθε join οπότε έπρεπε να διαλέξουμε με διαφορετική προσέγγιση. Για αυτό καταλήξαμε σε μια άλλη δομή.

Υλοποίηση με δυναμικά δεσμευμένο δισδιάστατο πίνακα.



Με αυτή την υλοποίηση υπολογίζουμε μέσω της συνάρτησης calculate between size() από πριν τον χώρο που θα χρειαστεί να δεσμεύσουμε για τον κάθε πίνακα κατά το πρώτο join. Εάν έχουμε δεύτερο υπολογίζουμε εκ νέου την διάστασή του και σβήνουμε τον παλιό αφού αντιγράψουμε το αποτελέσματά του στον καινούργιο.

Παρακάτω παραθέτουμε ορισμένες παραδοχες και βελτιστοποιήσεις που κάναμε για την μείωση του χρόνου εκτέλεσης

- Πριν γίνει κάθε join καλούμε την συνάρτηση prepare_relation() η οποία αναλαμβάνει να ταξινομήσει την εκάστοτε σχέση. Ανάλογα με το αν αυτή η σχέση έχει φιλτραριστεί ή έχει γίνει join, επιλέγουμε θα πάρουμε τα δεδομένα της από τον αρχικό πίνακα ή απο την δομή των ενδιάμεσων αποτελεσμάτων.

- Κατα την εκτέλεση του πρώτου join σε κάθε query γεμίζουμε την λίστα με την συνάρτηση result_list_fill_empty(), ενώ αν έχει πραγματοποιηθεί ήδη κάποιο join καλούμε την συνάρτηση result_list_update() ώστε να ανανεώσουμε την λίστα με τα καινούργια αποτελέσματα.

- Αν έχουν συμμετάσχει δύο σχέσεις σε κάποια ζεύξη και στη συνέχεια ξαναζητηθούν τότε η εκάστοτε ζεύξη εκτελείται ως φίλτρο. Ακόμη αν μια

σχέση ζητηθεί δυό φορες τότε γίνεται τα join με εκείνη τη σχέση πρώτα για να εκμεταλευτούμε το γεγονός ότι η σχέση αυτή θα είναι ήδη ταξινομημένη.

- Κάθε φορά που γίνεται κάποιο join τα στατιστικά των σχέσεων που συμμετείχαν ανανεώνονται μέσω της Update_Statistics.

- Όταν έχει γίνει κάποιο join, έστω 0.1 = 1.2 και στη συνέχεια ζητηθεί καινούργιο, έστω 1.3 = 2.2 πρέπει να ενημερωθεί η δομή που κρατάει τα ενδιάμεσα αποτελέσματα. Για να γίνει αυτό πρέπει κάθε rowid της σχέσης 1 πρέπει να συγκριθεί γραμμή-γραμμή με τα αποτελέσματα του καινούργιου join, κάτι που έχει πολυπλοκότητα της τάξης του $O(n^2)$. Στο medium αρχείο μια τέτοια προσέγγιση οδηγεί σε τρισεκατομμύρια συγκρίσεις που πρέπει να γίνουν για να έχουμε κάποιο αποτέλεσμα, άρα και σε χρόνους μη αποδεκτούς. Παρατηρήσαμε όμως, ότι πολλά rowid επαναλαμβάνονται για αυτό και όταν κάνουμε μια σύγκριση για ένα rowid και μετά αυτό το rowid επαναλαμβάνεται αντιγράφουμε κατευθείαν στην ενδιάμεση δομή τα αποτελέσματα για το ίδιο rowid. Με αυτή τη προσέγγιση και λόγω των πολλών διπλότυπων οι συγκρίσεις που χρειάζεται να γίνουν μειώθηκαν δραστικά.

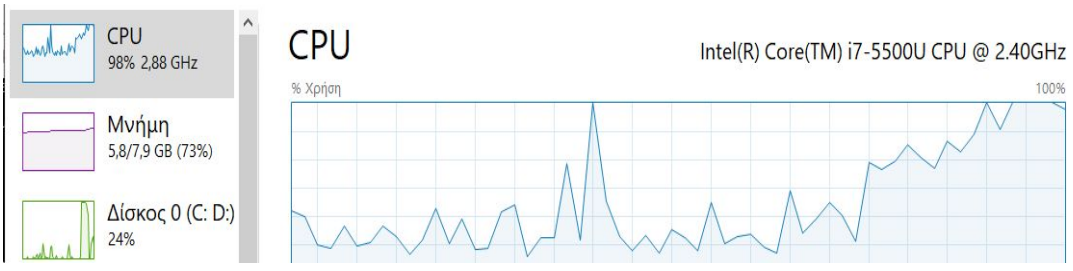
Έλεγχος αθροισμάτων

Στη συνάρτηση print_check_sums() δίνουμε ως όρισμα ποιά θέση του πίνακα τα rowid μας ενδιαφέρουν, αναλόγως τις προβολές του κάθε ερωτήματος, και απλά αθροίζουμε (απο το αρχικό datatable) τα payloads μόνο τον keys που περιέχονται στην ενδιάμεση δομή.

Χρόνοι Εκτέλεσης

Χωρίς παραλληλοποίηση τα benchmarks του προγράμματος μας κατά την εκτέλεση όλων των ερωτημάτων του “small.work” με την υλοπίηση της λίστας ήταν τα εξής :

<u>Για την χρήση της CPU και της RAM</u>



Ενώ ο χρόνος εκτέλεσης μετρήθηκε ως:

real 3m50.192s
sys 0m1.85

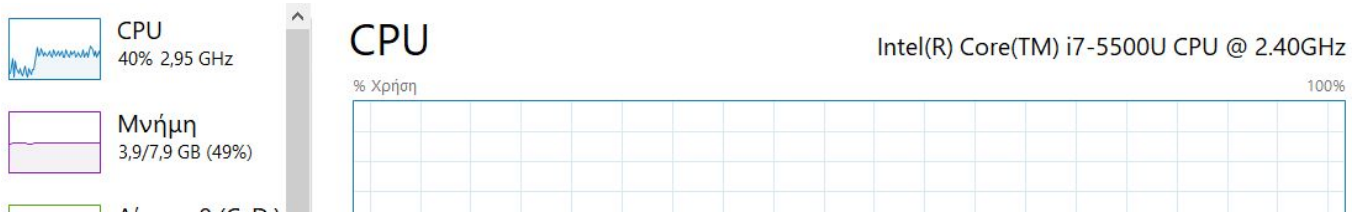
Με τη χρήση παραλληλοποίησης με 4 threads ο χρόνος έπεσε και συγκεκριμένα ήτανε:

real 2m13.192s
sys 0m1.85ι.

Υλοποίηση με δυναμικά δεσμευμένο δισδιάστατο πίνακα:

Χωρίς παραλληλοποίηση τα benchmarks του προγράμματος μας κατά την εκτέλεση όλων των ερωτημάτων του “small.work” ήταν τα εξής :

Για την χρήση της CPU και της RAM



Ενώ ο χρόνος εκτέλεσης μετρήθηκε ως:

real 1m7.192s
sys 0m1.85

Με τη χρήση παραλληλοποίησης σε επίπεδο Query με 3 threads ο χρόνος έπεσε και συγκεκριμένα ήτανε:

real 0m59.192s
sys 0m1.85

Με τη χρήση παραλληλοποίησης σε επίπεδο Query με 4 threads ο χρόνος αυξήθηκε και συγκεκριμένα ήτανε:

real 1m3.192s
user 1m2.382s
sys 0m1.85

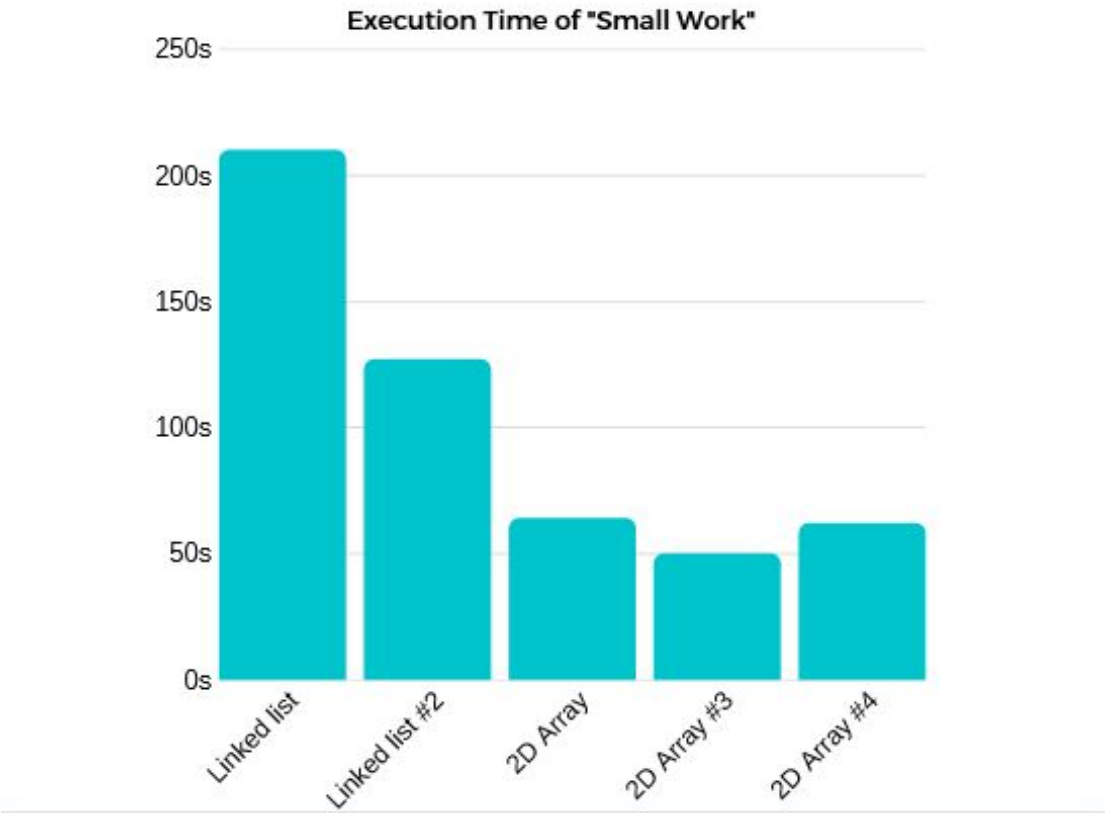
Για το medium.work οι μετρήσεις έγιναν στα μηχανήματα Linux της σχολής. Χωρίς παραλληλοποίηση μερικά από τα ερωτήματα αργούσαν πολύ να τελειώσουν ενώ

άλλα τρέχανε αισθητά πιο γρήγορα. Χρόνος εκτέλεσης για όλα τα bach δεν καταγράφονται καθώς είναι πάνω από 30 λεπτά.

Στη συνέχεια δοκιμάσαμε να εκτελέσουμε πάλι με 4 threads τα ερωτήματα όμως κάτι τέτοιο κατέστη ανέφικτο λόγω υπερφόρτωσης της κύριας μνήμης και των πάρα πολλών swap που γινόντουσαν. Δοκιμάσαμε στη συνέχεια με 2 threads και παρατηρήσαμε ότι εκτελώντας ένα batch ο χρόνος εκτέλεσης μειωνόταν.

Συμπεράσματα

Στην εργασία αυτή κληθήκαμε να υλοποιήσουμε τον αλγόριθμο ζεύξης πινάκων Sort Merge Join. Κατά τη διάρκεια υλοποίησης κληθήκαμε να πάρουμε πολλές αποφάσεις για να βελτιστοποιήσουμε όσο το δυνατόν την εκτέλεση των ερωτημάτων. Εν κατακλείδι παρακάτω παρουσιάζουμε τους χρόνους εκτέλεσης των ερωτημάτων του αρχείου small.work για μια συνολική επισκόπηση των αποτελεσμάτων.



Παρατηρούμε ότι στην περίπτωση της υλοποίησης με λίστα ένα Query έκανε πολύ ώρα να εκτελεστεί γιαυτο και κατά την παραλληλοποίηση σε επίπεδο Query η κλιμάκωση είναι μεγάλη και ο χρόνος μειώνεται δραστικά. Στην περίπτωση της

λίστας η εκτέλεση ενός Query δεν έπαιρνε πολύ χρόνο για αυτό και η κλιμάκωση που παρουσιάζεται είναι σαφώς μικρότερη.