ΑΝΑΠΤΥΞΗ ΛΟΓΙΣΜΙΚΟΥ ΓΙΑ ΠΛΗΡΟΦΟΡΙΑΚΑ ΣΥΣΤΗΜΑΤΑ

Τελική αναφορά

Γιαμπουόνκα-Κανελλάκος Κάρολος 1115201600030
Σόφι Αδαμαντία 1115201600158

- **Συνοπτική αναφορά στην συνολική λειτουργία του κώδικα:**

Στα πλαίσια της εργασίας, αναπτύχθηκε κώδικας πάνω στην εκτέλεση ερωτημάτων (queries) βασισμένη στον τελεστή σύζευξης ισότητας, με υλοποίηση αλγορίθμου Sort Merge Join, και έγινε ανάλυση αυτών που χρησιμοποιήθηκε στον προγραμματιστικό διαγωνισμό SIGMOD του 2018.

Στο πρώτο σκέλος, υλοποιήθηκε ο αλγόριθμος Sort Merge Join, για την σύζευξη ανάμεσα σε δύο σχέσεις με στοιχεία τύπου uint64_t.
Για την ταξινόμηση υλοποιήθηκε ο αλγόριθμος ταξινόμησης Radixsort in-place και in-memory, πάνω στην L1 cache, και ο επαναληπτικός αλγόριθμος Quicksort. Για την σύζευξη, τα αποτελέσματα αποθηκεύονται, με ταυτόχρονη σύγκριση των δύο σχέσεων, σε λίστα με buffer μεγέθους 1MB ανά κάδο.

Στο δεύτερο σκέλος, ακολούθησε η εκτέλεση των ερωτημάτων (queries) με προβολή αθροίσματα ελέγχου βάσει της σύζευξης (join) παραπάνω, self join και εφαρμογή φίλτρων στα κατηγορήματα που αποτελούν ένα ερώτημα.

Στο τρίτο και τελευταίο σκέλος της εργασίας, έγινε βελτιστοποίηση του κώδικα παραπάνω, στηριγμένη σε παραλληλία, υλοποίηση Job Scheduler και εργασιών για εκτέλεση ταξινόμησης και σύζευξης στα κατηγορήματα ενός ερωτήματος, QueryJob, sortJob, joinJob, αλλά και σε βελτιστοποιητή ερωτημάτων (query optimizer), υλοποίηση αλγορίθμου βάσει ζητούμενων στατιστικών και άλλων στοιχείων.

Έγιναν έλεγχοι ορθότητας κώδικα, σε κάθε σκέλος της εργασίας, με τη χρήση του framework Catch2.

- **Δομές και Οντότητες**

Αρχικά, υλοποιήθηκε δομή στοίβα, υλοποιημένη στα πλαίσια της εργασίας, για την επαναληπτική Quicksort, και η ζητούμενη λίστα με buffer των 1MB ανά κάδο, όπου έγινε αποθήκευση των αποτελεσμάτων της Sort Merge Join.

Βασικές οντότητες αποτέλεσαν ο πίνακας, class Matrix, για την αποθήκευση των δεδομένων εισόδου, οι σχέσεις που αντλούνται από αντίστοιχο πίνακα, class Relation, και σε πρώτο επίπεδο η οντότητα που συμμετέχει στην ταξινόμηση και σύζευξη, class Tuple, χρησιμοποιώντας την βασική δομή Vector υλοποιημένη και αυτή στα πλαίσια της εργασίας, δομικό στοιχείο των παραπάνω.

Στην συνέχεια, κύρια οντότητα αποτέλεσε το ερώτημα, class Query που αποτελείται από κατηγορήματα, class Predicate.

Υλοποιήθηκε, βασική δομή απλής συνδεδεμένης λίστας που κρατά τα αποτελέσματα από την εκτέλεση ενός ερωτήματος.

Τέλος, στο τρίτο σκέλος, για την παραλληλία που βελτιστοποιεί την εκτέλεση ερωτημάτων, υλοποιήθηκε διαχειριστής, class Job Scheduler με βασική οντότητα την εργασία, abstract class Job. Ρόλος του ο συγχρονισμένος, μέσω υλοποιημένης class Barrier και χρήση των mutexes, condition variables και semaphores, προγραμματισμός νημάτων, threads της βιβλιοθήκης pthread. Ο αριθμός αυτός είναι ορισμένος από τον χρήστη και γίνεται συλλογή αυτών σε ένα thread pool, για εκτέλεση εργασιών σχετικά με την εκτέλεση ερωτημάτων, ταξινόμηση και σύζευξη σχέσεων.

Επίσης, βάσει δοθέντων στατιστικών, υλοποιήθηκε ο αλγόριθμος δυναμικού προγραμματισμού για το Join Enumeration, προκειμένου να βρεθεί η βέλτιστη σειρά εκτέλεσης των κατηγορημάτων σε ένα ερώτημα συντελώντας στην βελτιστοποίηση αυτού.

- Παραδοχές

Για τον query optimizer χρησιμοποιήθηκαν οι τύποι στατιστικών που δίνονται στην εκφώνηση με μια μικρή προσθήκη. Όταν υπάρχει κάποιο φίλτρο ισότητας πάνω σε πίνακα που είναι το αποτέλεσμα μιας ζεύξης, τότε το φίλτρο αυτό εφαρμόζεται ακριβώς μετά από την εκάστοτε ζεύξη.

Η παραλλαγή αυτή μειώνει το χρόνο αλλά και την μνήμη που απαιτούνται για τον υπολογισμό και την αποθήκευση των ενδιάμεσων αποτελεσμάτων.

Το πρόγραμμα στην τελική του μορφή τρέχει 1 query job τη φορά, εφαρμόζοντας παραλληλία εσωτερικά του query, για το κομμάτι της ταξινόμησης και της ζεύξης.

Το κομμάτι της ταξινόμησης και το κομμάτι της ζεύξης δεν είναι απαραίτητο να τρέχουν με τον ίδιο αριθμό από νήματα και όπως θα φανεί και στα αποτελέσματα παρακάτω το κομμάτι της ζεύξης τρέχει γρηγορότερα με λιγότερα νήματα από το κομμάτι της ταξινόμησης.

Στο κομμάτι της ταξινόμησης, δημιουργείται το ιστόγραμμα για κάθε byte των δεδομένων που ταξινομούνται. Μόλις στο ιστόγραμμα εμφανιστεί διαφοροποίηση, και τα δεδομένα κατανεμηθούν σε περισσότερα από 1 buckets, τότε σπάμε το sorting σε n sort jobs(ή λιγότερες σε κάποιες περιπτώσεις), όπως έχουν οριστεί από το χρήστη, προσπαθώντας παράλληλα κάθε job να λάβει περίπου το ίδιο πλήθος δεδομένων.

Για την ταξινόμηση χρησιμοποιούνται όπως στην αρχή μόνο 2 Relations R και R', και ο αριθμός των threads δεν αλλάζει την απαίτηση σε μνήμη.

Στο κομμάτι της ζεύξης σπάμε τις 2 σχέσεις A, B που θα ζευχθούν σε m κομμάτια την κάθε μια. Στη συνέχεια αναθέτουμε m join jobs, όπου κάθε job αναλαμβάνει τη ζεύξη ενός κομματιού της σχέσης A με ένα κομμάτι της σχέσης B.

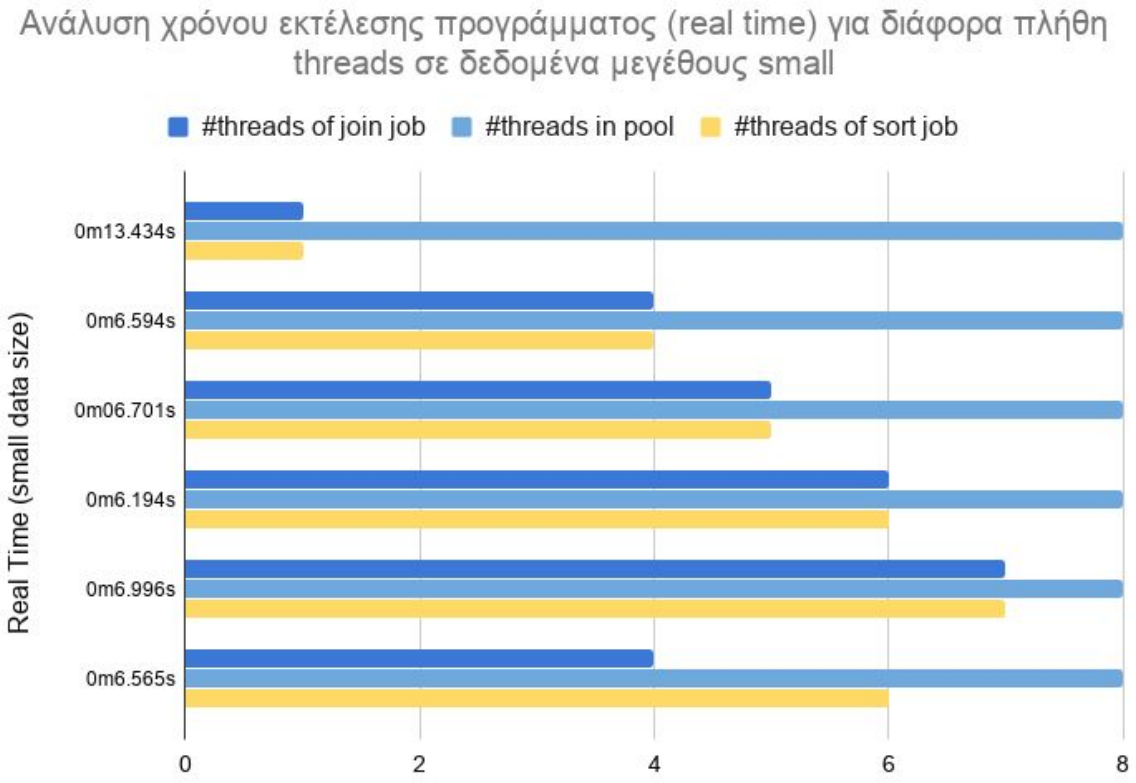
Είναι σημαντικό όταν ένα job λάβει ένα κομμάτι A_i, το αντίστοιχο κομμάτι B_i να περιλαμβάνει όλες τις τιμές της σχεσης B που υπάρχουν μέσα στο A_i. Υπάρχουν λοιπόν οι απαραίτητοι έλεγχοι για να σιγουρευτούμε ότι θα ισχύει πάντα η παραπάνω προϋπόθεση.

Για το κομμάτι της ζεύξης, αρκούν οι αρχικές σχέσεις A και B και δεν απαιτείται περισσότερη μνήμη για κάθε ένα κομμάτι που αναθέτουμε στα jobs.

- **Δοκιμές**

Εκτέλεση με χρήση query optimizer και παραλληλία βάσει threads από το thread pool του job scheduler, αναλαμβάνοντας jobs για εκτέλεση των ερωτημάτων και ταξινόμησης και ζεύξης πινάκων αυτού.

- δεδομένα εισόδου από το **small dataset**

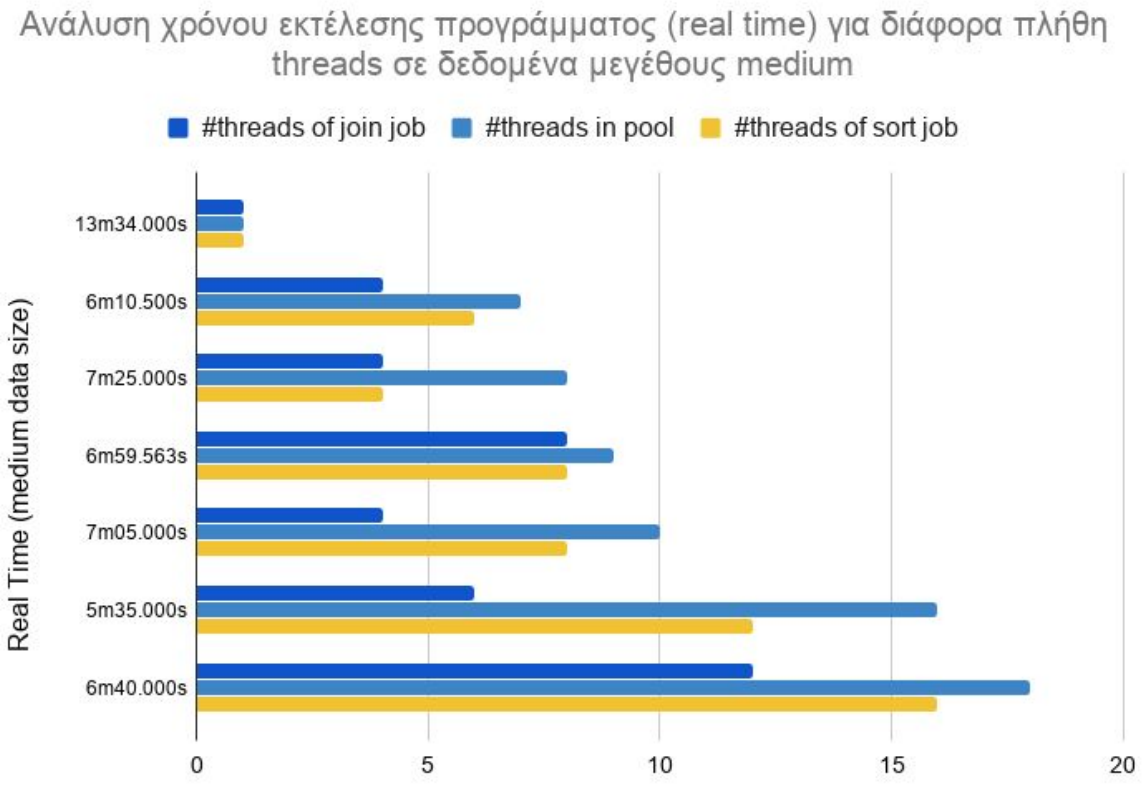


Peak RAM usage: ~750MB
Data load: < 0.1s
Machine Specs:
CPU: Intel i7-8550U (4 cores/ 8 threads)
RAM: 8GB DDR4 2400MHz
Δίσκος: 256GB SSD

Best performance:
execution (real) time: 0m06.194s
#threads in pool: 8

#threads of sort jobs: 6
#threads of join jobs: 6

- δεδομένα εισόδου από το **medium dataset**



Peak RAM usage: ~15.2GB
Data load: 4-5s
Machine Specs:
CPU: Intel i7-3770 (4 cores/ 8 threads)
RAM: 24GB DDR3 1600MHz
Δίσκος: 256GB SATA3 SSD

Best performance:
execution (real) time: 5m.35s
#threads in pool: 16
#threads of sort jobs: 12
#threads of join jobs: 6