ΑΝΑΠΤΥΞΗ ΛΟΓΙΣΜΙΚΟΥ ΓΙΑ ΠΛΗΡΟΦΟΡΙΑΚΑ ΣΥΣΤΗΜΑΤΑ
PROJECT

ΑΝΑΛΥΤΙΚΗ ΤΕΧΝΙΚΗ ΕΚΘΕΣΗ
REPORT

ΜΕΛΗ ΟΜΑΔΑΣ:

ΚΑΤΣΟΡΙΔΑΣ ΙΩΑΝΝΗΣ
ΕΞΑΜΗΝΟ: 9ο
ΑΡΙΘΜΟΣ ΜΗΤΡΩΟΥ: 1115201400066

ΜΗΤΡΑΚΗΣ ΓΕΩΡΓΙΟΣ
ΕΞΑΜΗΝΟ: 9ο
ΑΡΙΘΜΟΣ ΜΗΤΡΩΟΥ: 1115201400107

ΠΕΡΙΟΔΟΣ: ΧΕΙΜΕΡΙΝΟ ΕΞΑΜΗΝΟ 2018-2019

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ
ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΓΕΝΙΚΕΣ ΠΛΗΡΟΦΟΡΙΕΣ

- *Όνομα εκτελέσιμου: ./Caramel*
- *Μνήμη που καταναλώνεται κατά την εκτέλεση: 1,029,186,074 bytes (περίπου 981 MB)*
- *Εντολή μεταγλώττισης: make*
- *Διάρκεια εκτέλεσης: Βλέπε παράγραφο “Μετρήσεις” παρακάτω*
- *Ψηφία αλγορίθμου Radix: 12*
- *Unit Tests Framework: CUnit*

ΜΕΡΟΣ ΠΡΩΤΟ
ΑΛΓΟΡΙΘΜΟΣ RADIX HASH JOIN

Στο πρώτο μέρος της εργασίας αυτής, υλοποιήθηκε ο τελεστής Radix Hash Join για την βέλτιστη εκτέλεση κατηγορημάτων μεταξύ σχέσεων οι οποίες ενώνονται μεταξύ τους. Η λογική του αλγορίθμου είναι σχετικά απλή και εστιάζεται στο κατακερματισμό των σχέσεων προς σύζευξη με σκοπό τη γρηγορότερη προσπέλασή τους. Αυτό πραγματοποιήθηκε σε 3 συνολικά βήματα. Τα βήματα αυτά, όπως και η γενική απόδοση του αλγορίθμου εξαρτάται από τον αριθμό RADIX_N του αλγορίθμου με τον οποίο κατακερματίζονται οι τιμές.

Ο κατακερματισμός των τιμών πραγματοποιείται με βάση τον αριθμό RADIX_N. Πιο συγκεκριμένα, η κάθε τιμή οδηγείται στον κάδο με δείκτη τα τελευταία RADIX_N ψηφία της δυαδικής μορφής του. Για παράδειγμα, για τον αριθμό 45 και για τιμή RADIX_N 4, τότε θα η συγκεκριμένη τιμή θα κατακερματίζονταν στον κάδο 13, αφού η δυαδική μορφή του αριθμού 45 είναι 101101, και τα τελευταία 4 ψηφία, δηλαδή ο δυαδικός αριθμός 1101 αντιστοιχεί στον δεκαδικό 13.

Η βέλτιστη, θεωρητικά, τιμή για τον αλγόριθμο αυτό θα εξαρτώταν από το μέγεθος της κρυφής μνήμης του επεξεργαστή (cache). Αυτό συμβαίνει καθώς θα πρέπει ο κάθε κάδος να χωράει στη κρυφή μνήμη ώστε η προσπέλαση να γίνεται πολύ πιο γρήγορα, όταν έρθει η στιγμή της σύγκρισης. Με δεδομένο αυτό, δοκιμάστηκε ένας αναδρομικός αλγόριθμος ο οποίος παίρνοντας σαν είσοδο το μέγεθος της κρυφής μνήμης και το μέγεθος του πίνακα, επιστρέφει τον βέλτιστο αριθμό RADIX_N, σύμφωνα με τον οποίο, κάθε κάδος που δημιουργείται μετά τον κατακερματισμό θα χωράει στη κρυφή μνήμη. Κάτι τέτοιο, ωστόσο, θα προκαλούσε πρόβλημα στη περίπτωση που ένας πίνακας αποτελείται αποκλειστικά από τον ίδιο αριθμό, ή πιο γενικά αν ένας αριθμός εμφανίζεται περισσότερες φορές από το μέγεθος της κρυφής μνήμης, καθώς τότε το πρόγραμμα θα έτρεχε αναδρομικά χωρίς να φτάσει ποτέ στη τελική συνθήκη, με αποτέλεσμα να σκάσει η στοίβα. Για το λόγο αυτό, σαν αριθμός RADIX_N επιλέχθηκε στατικά ο αριθμός 12, καθώς δημιουργεί πίνακες κατακερματισμού 4096 θέσεων, που μοιάζει πιθανό να χωράνε σε κάθε κρυφή μνήμη, ειδικά στη σημερινή εποχή των αρκετών Gigabyte μνήμης.

Η λογική του αλγορίθμου, λοιπόν, ξεκινάει με τη δημιουργία ενός ιστογράμματος 4096 (2^{RADIX_N}) θέσεων και τον κατακερματισμό όλων των τιμών του κάθε πίνακα με σκοπό την εύρεση του μεγέθους κάθε κάδου. Πιο συγκεκριμένα, για κάθε τιμή που ανήκει σε έναν κάδο, η τιμή του εκάστοτε κάδου στο ιστόγραμμα αυξάνεται κατά ένα. Στο τέλος, δηλαδή, κάθε θέση του ιστογράμματος θα περιέχει το σύνολο των τιμών της σχέσης που κατακερματίζονται στον αντίστοιχο κάδο. Η διαδικασία αυτή πραγματοποιείται και για τις δύο σχέσεις που ενώνονται.

Χρησιμοποιώντας το εκάστοτε ιστόγραμμα, ο αλγόριθμος δημιουργεί τα αθροιστικά ιστογράμματα της κάθε σχέσης, τα οποία αντί να δείχνουν πόσες τιμές περιέχει ο κάθε κάδος, περιέχουν σε κάθε κελί τη θέση του πίνακα από την οποία θα ξεκινούσαν να τοποθετούνται οι τιμές του συγκεκριμένου κάδου, αν οι πίνακες ήταν ταξινομημένοι ανά κάδο. Αυτό γίνεται απλά προσθέτοντας σε κάθε τιμή του αθροιστικού ιστογράμματος, την αμέσως προηγούμενή της και την αντίστοιχη τιμή του απλού ιστογράμματος. Για παράδειγμα, για το ιστόγραμμα [2, 1, 3, 4, 2], το αθροιστικό ιστόγραμμα θα ήταν το [0, 2, 3, 6, 10].

Με βάση το αθροιστικό ιστόγραμμα, ξεκινάει μια δεύτερη προσπέλαση της κάθε σχέσης, κατά την οποία κατακερματίζονται μια δεύτερη φορά όλες οι τιμές. Κατά τον κατακερματισμό, ωστόσο, οι τιμές τοποθετούνται σε μία νέα σχέση, στη θέση που υποδεικνύει το αθροιστικό ιστόγραμμα του εκάστοτε κάδου. Έπειτα, η τιμή στη συγκεκριμένη θέση του αθροιστικού ιστογράμματος αυξάνεται κατά ένα. Αυτό οδηγεί στην ταξινόμηση, πρακτικά των σχέσεων με βάση

τους κάδους. Πρόκειται για μια διαδικασία η οποία μπορεί να είναι χρονοβόρα, γλυτώνει ωστόσο σημαντικό χρόνο από την μετέπειτα σύγκριση των τιμών, καθώς δε χρειάζεται πλέον να συγκριθούν όλες οι τιμές από όλους τους κάδους αλλά μόνο οι τιμές από τους κάδους στους οποίους οδηγεί η εκάστοτε τιμή κατακερματισμού. Με άλλα λόγια, αποφεύγει την πολυπλοκότητα δεύτερης τάξης.

Το δεύτερο επίπεδο κατακερματισμού, γίνεται σε επίπεδο κουβάδων των ήδη κατακερματισμένων relations R' και S' . Γνωρίζοντας τα R' και S' και τον αύξων αριθμό των κάδων μεταξύ των οποίων θα γίνει το Join, δημιουργείται ένα ευρετήριο για το relation με τον μικρότερο κάδο. Το ευρετήριο αυτό χρησιμοποιεί δύο πίνακες , τον bucket και τον chain, με τις λειτουργίες που περιγράφονται στην εκφώνηση. Η δεύτερη συνάρτηση κατακερματισμού επιστρέφει το υπόλοιπο της διαίρεσης του ήδη κατακερματισμένου payload με το 10. Τέλος, προσπελαύνοντας τα ζευγάρια του μεγαλύτερου κάδου από το τέλος προς την αρχή, χρησιμοποιεί το ευρετήριο για να βρει τα payloads του μικρότερου που είναι ίσα με αυτά του μεγαλύτερου. Αποθηκεύει τα ζεύγη των keys των κάδων που έχουν ίσα payloads σε μια λίστα αποτελεσμάτων.

ΜΕΡΟΣ ΔΕΥΤΕΡΟ
ΣΥΖΕΥΞΗ ΚΑΙ ΕΦΑΡΜΟΓΗ ΦΙΛΤΡΟΥ ΣΕ ΕΠΕΡΩΤΗΣΕΙΣ

Αποθήκευση Σχέσεων Στη Μνήμη:

Το πρόγραμμα δέχεται ως είσοδο μια σειρά από ονόματα αρχείων, κάθε ένα από τα οποία περιέχει σε δυαδική μορφή έναν πίνακα με τις τιμές του. Χρησιμοποιώντας τη συνάρτηση loadRelation του αρχείου Relation.cpp του πακέτου submission του διαγωνισμού SIGMOD (κατάλληλα τροποποιημένου ώστε να υποστηρίζει τη γλώσσα C στην οποία είναι γραμμένος ο υπόλοιπος κώδικας αντί για τη C++ που είναι το πακέτο submission), φορτώνονται στη μνήμη όλες οι σχέσεις που έχουν δοθεί σαν input. Οι σχέσεις φορτώνονται σε μια δομή τύπου: table** όπου κρατώνται τόσο ο αριθμός των στηλών και των κολόνων, όσο και οι ίδιες οι τιμές του πίνακα.

Έπειτα, γεμίζει μια δομή Metadata* για κάθε κολόνα του πίνακα, η οποία περιέχει στατιστικά όπως ο αριθμός των μοναδικών τιμών (distincts) ανά κολόνα ή η μέγιστη και η ελάχιστη τιμή. Αυτά τα στατιστικά θα χρησιμοποιηθούν για την επιλογή της ένωσης ή σύγκρισης που θα πραγματοποιηθεί πρώτα σε ένα query, ανάλογα με το ποιο αναμένεται ότι θα επιστρέψει τα λιγότερα αποτελέσματα.

Επεξεργασία των Queries:

Μετά την αποθήκευση των σχέσεων στη μνήμη, το πρόγραμμα αναμένει να εισέλθουν ως είσοδος κάποιες επερωτήσεις, από τις οποίες θα προκύψουν τα επιθυμητά αποτελέσματα. Διαβάζει τις επερωτήσεις ανά ομάδα, όπου η κάθε ομάδα τελειώνει όταν διαβαστεί ο χαρακτήρας "F". Μετά το διάβασμα μιας ομάδας, εκτελεί το κάθε query με τη σειρά εισόδου του και εκτυπώνει το άθροισμα των τιμών που θα ήταν το αποτέλεσμα της επερώτησης. Έπειτα, περιμένει εκ νέου να δοθούν νέα queries.

Για την επεξεργασία των επερωτήσεων, χρησιμοποιείται η δομή Query η οποία περιέχει δείκτες σε δομές όπως Column_t, Comparison_t, Query_relation_t οι οποίες περιέχουν τις απαραίτητες πληροφορίες σχετικά με την επερώτηση. Έπειτα, η δομή αυτή δίνεται ως είσοδος στη συνάρτηση ExecuteQueries η οποία ανάλογα με τον τύπο του, πραγματοποιεί αντίστοιχες πράξεις.

Εκτέλεση Κατηγορημάτων:

Προτού εκτελεστούν τα κατηγορήματα, δίνεται σε κάθε ένα από αυτά ένας βαθμός προτεραιότητας. Αυτός ο βαθμός προκύπτει από τον αναμενόμενο αριθμό αποτελεσμάτων που θα επιστρέψει αυτό το κατηγόρημα, σε περίπτωση που τα στοιχεία ήταν ομοιόμορφα κατανεμημένα. Το πραγματοποιεί αυτό χρησιμοποιώντας τα στοιχεία που σύλλεξε μετά την αποθήκευση των σχέσεων. Όσο λιγότερα αποτελέσματα αναμένεται να επιστραφούν, τόσο πι γρήγορα θα εκτελεστεί το επερώτημα. Αυτό γίνεται καθώς, με δεδομένο πως όπως και να εκτελεστούν θα επιστρέψουν τα ίδια αποτελέσματα, αν από την αρχή υπάρχει μικρός αριθμός αποτελεσμάτων, τα υπόλοιπα επερωτήματα θα εκτελεστούν πιο γρήγορα.

Για τη διαδοχική εκτέλεση των κατηγορημάτων, χρησιμοποιείται μια συνδεδεμένη λίστα, κάθε κόμβος της οποίας περιέχει έναν πίνακα από ενδιάμεσα αποτελέσματα. Κάθε φορά που εκτελείται ένα predicate, θα υπάρχει (αν τα αποτελέσματα δεν είναι μηδέν) ένας αριθμός από σειρές οι οποίες ικανοποιούν τη σχέση που έχει δοθεί. Αυτές οι σειρές αποθηκεύονται σε έναν ενδιάμεσο πίνακα αποτελεσμάτων, ώστε την επόμενη φορά που μία από τις σχέσεις, που βρίσκονται σε κάποιο πίνακα, βρεθεί σε άλλο ένα κατηγόρημα, να χρησιμοποιηθούν οι τιμές που βρίσκονται στα ενδιάμεσα αποτελέσματα και όχι κάποιες άλλες οι οποίες έχουν ήδη αποκλειστεί λόγω κάποιας προηγούμενης συνθήκης. Στη περίπτωση που ένα κατηγόρημα περιλαμβάνει σχέση ή σχέσεις των οποίων καμία δεν βρίσκεται σε κάποιο ενδιάμεσο αποτέλεσμα, δημιουργείται ένας νέος πίνακας ενδιάμεσων αποτελεσμάτων που προστίθεται στη λίστα. Αν βρεθεί κατηγόρημα το οποίο περιλαμβάνει σχέσεις από 2 ξεχωριστούς πίνακες, τότε αυτοί ενώνονται σε έναν.

Έτσι, ανάλογα με τη προτεραιότητα, εκτελούνται ένα-ένα όλα τα κατηγορήματα. Οι κατηγορίες στις οποίες μπορεί να ανήκει ένα κατηγόρημα είναι τρεις: να είναι σύγκριση με έναν αριθμό (>, <, =), να είναι ένωση (Join) μεταξύ κολόνων της ίδιας σχέσης και να είναι join μεταξύ ανεξάρτητων σχέσεων.

Στη πρώτη κατηγορία, ελέγχονται όλες οι τιμές της κολόνας της σχέσης (ή αντίστοιχα οι τιμές που βρίσκονται στις σειρές που υπάρχουν σε κάποιο πίνακα ενδιάμεσων αποτελεσμάτων, αν έχει ήδη πραγματοποιηθεί κάποια πράξη στην ίδια σχέση) και αυτές που ικανοποιούν τη συνθήκη, αποθηκεύονται σε ένα πίνακα αποτελεσμάτων.

Στη δεύτερη κατηγορία, σε αντίθεση με τις άλλες ενώσεις, δεν εκτελείται ο αλγόριθμος του Radix Hash Join, καθώς πρόκειται για κολόνες τις ίδιας σχέσης. Αντίθετα, ελέγχονται όλες οι σειρές που βρίσκονται είτε σε κάποιο ενδιάμεσο αποτέλεσμα, είτε από τον αρχικό πίνακα αν δεν έχει προηγηθεί κατηγόρημα στη συγκεκριμένη σχέση, ώστε να δοθούν σαν αποτελέσματα μόνο οι σειρές της σχέσης, των οποίων οι εν λόγω κολόνες έχουν ίδια τιμή. Κάτι παρόμοιο γίνεται και στη περίπτωση που δύο σχέσεις βρίσκονται στον ίδιο πίνακα ενδιάμεσων αποτελεσμάτων και φτάνει ένα κατηγόρημα για αυτές.

Στη τελευταία περίπτωση πρέπει να εκτελεστεί ο αλγόριθμος Radix Hash Join που υλοποιήθηκε στο προηγούμενο μέρος. Ο αλγόριθμος παίρνει σαν είσοδο δύο σχέσεις και επιστρέφει τις σειρές που έχουν ίδια τιμή. Για να χρησιμοποιηθεί, λοιπόν, ο RHJ, πρώτα φτιάχνονται σχέσεις από τις κολόνες που ενώνονται. Αν κάποια υπάρχει σε κάποιο ενδιάμεσο αποτέλεσμα, τότε ο αριθμός της εκάστοτε σειράς δίνεται από την θέση στον ενδιάμεσο πίνακα. Αν καμία από τις δοθείσες σχέσεις δε βρίσκεται σε κάποιο ενδιάμεσο αποτέλεσμα, τότε δημιουργείται ένας νέος πίνακας και προστίθεται στη λίστα. Αν από την άλλη υπάρχει μόνο μία από τις δύο σχέσεις, τότε ο πίνακας στον οποίο βρίσκεται η άλλη μεγαλώνει κατά μία στήλη, και προστίθενται εκεί οι τιμές της νέας σχέσης. Αν από την άλλη και οι δύο σχέσεις βρίσκονται σε διαφορετικά ενδιάμεσα αποτελέσματα, τότε, όπως έχει ήδη αναφερθεί, οι δύο πίνακες ενώνονται σε έναν κοινό.

Εκτύπωση Αποτελεσμάτων:

Αφού εκτελεστούν όλα τα κατηγορήματα, θα προκύψει μια σειρά από πίνακες ενδιάμεσων αποτελεσμάτων. Σε εκείνο τη σημείο λαμβάνονται υπόψη οι επιθυμητές κολόνες προς εκτύπωση. Αν και οι δύο κολόνες βρίσκονται στον ίδιο πίνακα, τότε από αυτόν εκτυπώνονται οι κολόνες ως έχουν. Αν από την άλλη οι κολόνες βρίσκονται σε πίνακες ενδιάμεσων αποτελεσμάτων που δεν

είναι ενωμένοι, τότε δημιουργείται ένας νέος πίνακας ενδιάμεσων αποτελεσμάτων που αποτελεί το καρτεσιανό γινόμενο των δύο προηγούμενων. Και από αυτόν τον πίνακα, εκτυπώνονται οι επιθυμητές κολόνες.

ΜΕΡΟΣ ΤΡΙΤΟ
ΠΑΡΑΛΛΗΛΟΠΟΙΗΣΗ ΚΑΙ ΒΕΛΤΙΩΣΗ ΑΠΟΔΟΣΗΣ

Παραλληλοποίηση:

Ο βασικότερος τρόπος για την αποδοτικότερη εκτέλεση των επερωτήσεων, σε συνάρτηση πάντα με τον χρόνο, είναι η παραλληλοποίηση κομματιών του αλγορίθμου, με σκοπό την παράλληλη εκτέλεσή τους και τον μικρότερο χρόνο εκτέλεσης. Για να επιτευχθεί αυτή η μέθοδος, χρειάζεται να μοιραστεί ο φόρτος εργασίας ανάμεσα στα διαθέσιμα νήματα (threads) του προγράμματος. Λεπτομέρειες σχετικά με την απόδοση του προγράμματος σε σχέση με τον αριθμό των νημάτων βρίσκονται στις επόμενες σελίδες. Ένα ακόμα σημαντικό στοιχείο που πρέπει να παρατηρηθεί είναι πως δεν μπορεί να εφαρμοστεί παντού η παραλληλοποίηση, παρά μόνο στον τελεστή Radix Hash Join. Αυτό συμβαίνει καθώς σε πολλά σημεία απαιτείται η ταυτόχρονη προσπέλαση στα ίδια δεδομένα από διαφορετικά νήματα, κάτι το οποίο θα καθυστερούσε σημαντικά την εκτέλεση, ενώ σε άλλα σημεία θα την έκανε αδύνατη.

Το πρώτο σημείο του αλγορίθμου που μπορεί να βελτιωθεί η απόδοση είναι η δημιουργία του ιστογράμματος (Histogram). Αντί να προσπελάσει το πρόγραμμα όλους τους πίνακες στο βασικό νήμα, χωρίζει τη κάθε σχέση σε ίσα κομμάτια, τόσα όσος είναι ο αριθμός των διαθέσιμων νημάτων, και αναθέτει σε κάθε ένα, ένα κομμάτι από τη σχέση. Αυτό οδηγεί το κάθε νήμα στη δημιουργία ενός αυτόνομου ιστογράμματος, για το κομμάτι της σχέσης που προσπελαύνει. Το κάθε ιστόγραμμα θα είναι ίδιου μεγέθους με το αρχικό, απλά θα έχει λιγότερες τιμές. Όπως γίνεται αντιληπτό, η πρόσθεση όλων των κελιών της ίδιας θέσης από όλα τα ιστογράμματα θα οδηγήσει στο αρχικό ιστόγραμμα, σε αρκετά μειωμένο, ωστόσο χρόνο.

Αντίστοιχα, το δεύτερο κομμάτι του αλγορίθμου αφορά στη δημιουργία των νέων σχέσεων κατά τη διάρκεια του αλγορίθμου του Radix Hash Join, και πιο συγκεκριμένα στην αντιγραφή των τιμών στις νέες θέσεις (Partition). Όταν υπήρχε ένα μοναχά νήμα, δημιουργούταν το αθροιστικό ιστόγραμμα, και από αυτό, δημιουργούταν οι δύο νέες ταξινομημένες σχέσεις. Εκμεταλλευόμενοι, ωστόσο, την παραλληλοποίηση, είναι δυνατό να αποφευχθεί η δημιουργία ενός και μόνο αθροιστικού ιστογράμματος, αλλά θα δημιουργηθεί ένα αθροιστικό ιστόγραμμα ανά νήμα. Ορίζεται σαφώς, επομένως, το πλήθος των θέσεων που μπορεί να γράψει το κάθε νήμα και αντιγράφει μόνο τις τιμές που του έχουν ανατεθεί. Στο τέλος, δημιουργείται σε ελάχιστο χρόνο μια σχέση αντίστοιχη της πρώτης, ταξινομημένης, ωστόσο, ανά κάδο.

Τέλος, το τρίτο κομμάτι αφορά στην παράλληλη σύγκριση των κάδων ώστε να προκύψουν οι σειρές που θα αποτελέσουν στο τέλος το αποτέλεσμα (Join). Στο συγκεκριμένο είδος εργασίας, μεταφέρεται ουσιαστικά όλη η λειτουργία του αλγορίθμου Radix Hash Join όπως αυτός υλοποιήθηκε στο πρώτο μέρος με μόνη διαφορά ότι στο τέλος, δε επιστρέφεται η λίστα των αποτελεσμάτων που δημιουργήθηκε αλλά προστίθεται στη γενική λίστα που περιέχει τα αποτελέσματα όλων των νημάτων. Αυτή είναι και η λίστα που επιστρέφεται από τον αλγόριθμο και το αποτέλεσμα ουσιαστικά της σύζευξης.

Για τη σωστή υλοποίηση της παραπάνω λειτουργίας απαιτείται τόσο σωστός συγχρονισμός μεταξύ των διάφορων εργασιών, όσο κάποια δομή η οποία θα μοιράζει τις εργασίες στα νήματα. Η δομή αυτή ονομάζεται JobScheduler και είναι υπεύθυνη για τη σωστή λειτουργία της παραλληλοποίησης. Οι διαφορετικές εργασίες που πρέπει να υλοποιήσουν τα νήματα χωρίζονται σε

τρεις κατηγορίες, τα HistogramJobs, PartitionJobs και JoinJobs, το κάθε ένα από τα οποία πραγματοποιεί την εργασία που αναφέρει το όνομά του. Τα Histogram και Partition Jobs που δημιουργούνται είναι στο σύνολο τόσα, όσα είναι και τα νήματα, καθώς όπως προαναφέρθηκε, κάθε νήμα θα πάρει ένα ίσο κομμάτι της σχέσης για να επιτελέσει τις πράξεις του. Τα JoinJobs πραγματοποιούνται σε επίπεδο κουβάδων, οπότε θα φτιαχτούν τόσα, όσοι οι διαθέσιμοι κουβάδες, δηλαδή τα αποτελέσματα της συνάρτησης κατακερματισμού. Όλες οι εργασίες, αφού δημιουργηθούν, τοποθετούνται σε μια ουρά, από την οποία το κάθε νήμα δέχεται την εργασία που θα εκτελέσει. Πρέπει, ωστόσο, να έχουν τελειώσει όλα τα HistogramJobs για να αρχίσουν τα PartitionJobs και αντίστοιχα να έχουν τελειώσει όλα τα PartitionJobs για να αρχίσουν τα JoinJobs, γιατί διαφορετικά υπάρχει περίπτωση να απαιτήσει κάποιο νήμα δεδομένα που δεν έχουν γραφεί ακόμα από κάποιο άλλο. Αυτό πραγματοποιείται από τον JobScheduler, ο οποίος περιμένει να τελειώσουν όλα τα νήματα τις δουλειές τους προτού αρχίσει να τοποθετεί στη ουρά τις επόμενες δουλειές. Εν τω μεταξύ, τα νήματα “κοιμούνται” μέχρις ότου να υπάρχει κάποια εργασία στην ουρά ώστε να την εκτελέσουν. Αφού εκτελεστούν όλες οι εργασίες, ο JobScheduler είναι υπεύθυνος για την απελευθέρωση της μνήμης όλων των νημάτων, για την έξοδο όλων των νημάτων και τέλος για την απελευθέρωση της μνήμης της ίδια της δομής. Σε κάθε ζεύξη, δημιουργείται και ένας νέος JobScheduler μαζί με νέα νήματα.

Πολλές φορές, τα νήματα πρέπει να προσπελάσουν ή και να αλλάξουν μνήμη που είναι κοινή για όλους. Μια τέτοια περίπτωση είναι η προσθήκη της λίστας αποτελεσμάτων του κάθε νήματος στη συνολική λίστα. Για να πραγματοποιηθεί μια τέτοια εργασία, χωρίς να υπάρχει ταυτόχρονο γράψιμο σε ίδια θέση μνήμης, απαιτείται κατάλληλος συγχρονισμός. Αυτό επιτυγχάνεται με τη χρήση κλειδαριών (mutexes) αλλά και σημαφόρων (semaphores). Οι κλειδαριές χρησιμεύουν στη προστασία δομών όπως η ουρά των εργασιών ενώ οι σημαφόροι βοηθούν στο μπλοκάρισμα των νημάτων, όταν δεν υπάρχει διαθέσιμη εργασία, και αντίστοιχα στην αφύπνισή τους όταν μια εργασία τοποθετηθεί στην ουρά. Τέλος, σημαφόροι χρησιμεύουν και για την αναμονή του JobScheduler μέχρις ότου ολοκληρωθούν όλες οι εργασίες της ουράς από τα νήματα.

Join Enumeration Algorithm:

Ένας άλλος τρόπος, πέρα από την παραλληλοποίηση, ο οποίος μπορεί να οδηγήσει σε βελτιωμένη απόδοση είναι η αλλαγή της σειράς των ζεύξεων, ώστε να προκύψουν τα ελάχιστα δυνατά ενδιάμεσα αποτελέσματα. Για να πραγματοποιηθεί αυτό, θα χρησιμοποιηθούν τα στατιστικά που υπολογίστηκαν στο δεύτερο μέρος, μαζί με τον αλγόριθμο Join Enumeration. Θα προκύψει έτσι η καλύτερη δυνατή σειρά ζεύξεων, από την οποία θα προκύψει η καλύτερη δυνατή σειρά εκτέλεσης κατηγορημάτων. Ο αλγόριθμος πρακτικά υπολογίζει το πλήθος περίπου των αποτελεσμάτων που θα προέκυπταν αν οι τιμές των σχέσεων ήταν ομοιόμορφα κατανεμημένες, με βάση κάποιους δεδομένους τύπους. Έτσι γνωρίζει στο περίπου πόσα είναι τα αναμενόμενα αποτελέσματα για κάθε πιθανό συνδυασμό συζεύξεων και επιστρέφει το συνδυασμό που θα δημιουργήσει τα λιγότερα ενδιάμεσα αποτελέσματα.

Αυτό επιτυγχάνεται δημιουργώντας δύο πίνακες κατακερματισμού (hash tables), έναν που για κάθε δυνατό συνδυασμό σχέσεων επιστρέφει τη βέλτιστη σειρά ζεύξεων τους (BestTree) και ένα που για κάθε δυνατό συνδυασμό σχέσεων επιστρέφει το κόστος της ζεύξης τους με τη συγκεκριμένη σειρά (Cost). Ο δεύτερος πίνακας, δεν περιέχει μονάχα τα κόστη (δηλαδή τα αναμενόμενα αποτελέσματα) αλλά όλα τα στατιστικά των στηλών κάθε πίνακα μετά από τη σύζευξη των σχέσεων της εκάστοτε θέσης. Αυτό είναι απαραίτητο ώστε να χρησιμοποιούνται τα ανανεωμένα στατιστικά σε κάθε υπολογισμό κόστους και όχι τα αρχικά της κάθε κολόνας. Η συνάρτηση κατακερματισμού που χρησιμοποιήθηκε για την τοποθέτηση των συνόλων στον πίνακα είναι η προσθήκη του παραγοντικού του δείκτη της κάθε σχέσης αυξανόμενου κατά ένα. Για παράδειγμα για το σύνολο: [1, 3, 4] η κατακερματισμένη τιμή θα ήταν: (1+1)! + (3+1)! + (4+1)! =

146. Αυτή η τιμή είναι μοναδική για κάθε διαφορετικό υποσύνολο των σχέσεων αλλά ίδια για κάθε διαφορετική εμφάνιση του ίδιου υποσυνόλου (πχ BestTree[{1, 2, 4}] == BestTree[{4, 1, 2}] κλπ).

Σε κάθε σύζευξη, υπολογίζονται τα στατιστικά των σχέσεων μετά τη σύζευξη και αποθηκεύονται σε μια νέα δομή Cost, η οποία αντιγράφεται από τη δομή των σχέσεων χωρίς τη συζευγμένη. Άμα το κόστος, δηλαδή τα αναμενόμενα αποτελέσματα, είναι λιγότερο από το κόστος της βέλτιστης σειράς μέχρις εκείνης της στιγμής, τότε η σειρά αντικαθιστά της άλλη ως BestTree και ανανεώνεται ο πίνακας κόστους. Στο τέλος, επιστρέφεται η βέλτιστη σειρά εκτέλεσης των ζεύξεων, από την οποία προκύπτει η βέλτιστη σειρά εκτέλεσης των κατηγορημάτων. Η σειρά εκτέλεσης κατηγορημάτων που ενώνουν τους ίδιους πίνακες (είτε αυτούσιους είναι ως μέρος ενδιάμεσων αποτελεσμάτων) είναι τυχαία.

Μετρήσεις:

Παρακάτω παρατίθεται ένας πίνακας με τους χρόνους που πήρε το πρόγραμμα να εκτελέσει το small workload:

Thread Amount	1	2	4	8	10	12
small workload	3m 49s	4m 13s	4m 1s	4m	4m 2s	4m 5s
small workload without 2 queries (*)	0m 21s	0m 25s	0m25s	0m 26s	0m 27s	0m 28s

(*)Σημείωση: η δεύτερη μέτρηση έγινε ενδεικτικά χωρίς τα ακόλουθα 2 queries, λόγω μεγάλης καθυστέρησης στην εκτέλεσή τους:

- 0 13 7 10 | 0.0=1.2&0.0=2.1&0.0=3.2&1.2>295 | 3.2 0.0
  (batch #4, query #2)
- 8 0 13 13 | 0.2=1.0&1.0=2.2&2.1=3.2&0.1>7860 | 3.3 2.1 3.6
  (batch #5, query #5)

Unit Tests:

Χρησιμοποιούμε το CUnit Framework για να φτιάξουμε κάποια Unit Tests για να ελέγξουμε την ορθότητα των μεθόδων μας, κυρίως για το πρώτο και δεύτερο μέρος της εργασίας.