Κ23 - Ανάπτυξη Λογισμικού Για Πληροφοριακά Συστήματα

Χειμερινό Εξάμηνο 2018-2019
Καθηγητής : Ι. Ιωαννίδης

Υλοποίηση συστήματος αξιολόγησης ερωτημάτων join με στόχο τον βέλτιστο τρόπο με ελάχιστο προηγούμενο indexing

Τμήμα Παρακολούθησης : Τμήμα Τρίτης

Ονόματα Μελών Ομάδας

Ηλιάκης Μάρκος - 1115201500185
Παπαναστασίου Λεωνίδας 1115201500121

Τιμητική Αναφορά : Τσαπέλας Χρήστος - 1115201500161

Περιεχόμενα

Εισαγωγή

Σκοπός του φετινού project, ήταν η υλοποίηση ενός συστήματος διαχείρισης ερωτημάτων join από ένα σύνολο προκαθορισμένων σχέσεων, που θα τα αξιολογούσε αναλόγως με τα relations, predicates και selections που τα συνοδεύουν. Επιπροσθέτως, στα ενδιάμεσα βήματα δεν επιτρεπόταν να γίνεται κάποια ιδιαίτερη ευρετηρίαση.

Σημαντικό κομμάτι αυτού του συστήματος ήταν και η βέλτιστή του συμπεριφορά ως προς τις απαιτήσεις μνήμης και χρόνου απόκρισης, κομμάτι που χαρακτήρισε το τρίτο μέρος της εργασίας.

Όντας μεγάλο και απαιτητικό σύστημα προς υλοποίηση, το project αυτό χωρίστηκε από τους υπευθύνους σε τρία μέρη για την ευκολότερη πραγμάτωσή του. Το πρώτο μέρος αφορούσε την απλή υλοποίηση του αλγορίθμου Radix Hash Join, το δεύτερο την βελτίωση του και την προσθήκη ανάγνωσης και αποθήκευσης των δεδομένων (ερωτημάτων) και το τρίτο την παραλληλοποίησή του για καλύτερη απόδοση και την εύρεση της καλύτερης σειράς εκτέλεσης των join.

Στόχος του report αυτού, είναι η περιγραφή του τρόπου εργασίας της ομάδας κατά την υλοποίηση του προαναφερθέντως συστήματος, την αναλυτική περιγραφή των τριών μερών που χωρίστηκε, τις συμβάσεις που ακολουθήθηκαν και τον τρόπο που βγήκε το τελικό αποτέλεσμα.

Μέρος Πρώτο

Το πρώτο μέρος της εργασίας, ζητούσε την υλοποίηση ενός αλγορίθμου join πινάκων, του Radix Hash Join. Για την υλοποίηση του απαιτούνται διάφορα βήματα προετοιμασίας των δεδομένων, προτού γίνει η τελική ενοποίηση των πινάκων.

Για την πραγματοποίηση του πρώτου αυτού μέρους, έγιναν δύο συναντήσεις των μελών της ομάδας. Στην πρώτη έγινε η ένωση των πινάκων η υλοποίηση του histogram, του psum και της hash function.

Η ομάδα ακόμα περιλάμβανε και τα τρία αρχικά μέλη, όπου η συνεργασία έγινε όπως αναφέρθηκε σε κοινούς χρόνους, χωρίς ιδιαίτερη ατομική δουλειά.

Δυσκολίες εντοπίστηκαν στον αλγόριθμο του Radix Hash Join, κυρίως στον καθορισμό των δεδομένων εισόδου, για αυτό το λόγο δημιουργήθηκαν δύο πίνακες είκοσι γραμμών και τεσσάρων στηλών, οι οποίοι αρχικοποιήθηκαν με την εντολή rand. Στη συνέχεια, υλοποιήθηκε η ζεύξη των δύο πινάκων, σε τρίτη στήλη του καθενός. Πριν την εφαρμογή της ζεύξης, όπως υποδείχθηκε στην εκφώνηση, έγινε η προεπεξεργασία των δεδομένων. Αρχικά έγινε η απομόνωση της στήλης που συμμετέχει στη ζεύξη, και στα δεδομένα της εφαρμόστηκε η πρώτη συνάρτηση κατακερματισμού, η οποία χώρισε σε κάδους τα δεδομένα της στήλης αυτής. Η συνάρτηση αυτή απομονώνει τα 8 τελευταία ψηφία του αριθμού (μέγεθος το οποίο υποδείχθηκε σε φροντιστήριο).

Μετέπειτα, με βάση τους κάδους που δημιουργήθηκαν στο προηγούμενο βήμα, κατασκευάζονται το ιστόγραμμα των διαφορετικών τιμών που βρίσκονται στους κάδους και το ιστόγραμμα αθροίσματος έτσι ώστε να ταξινομηθούν τα δεδομένα του πίνακα με βάση τη τιμή της συνάρτησης κατακερματισμού και να αποθηκευτούν τα όρια του κάθε κάδου.

Ακολούθως, κατασκευάζονται τα ευρετήρια για κάθε κάδο με βάση τη δεύτερη συνάρτηση κατακερματισμού. Ο τύπος της συνάρτησης αυτής είναι το mod της τιμής με το 101 όπως υποδείχθηκε στην εκφώνηση. Το αποτέλεσμα των ευρετηρίων αυτών είναι για κάθε κάδο να φτιαχτούν “αλυσίδες”, όπου κάθε κόμβος της αλυσίδας δείχνει τις θέσεις των στοιχείων μέσα στο κάδο τα οποία έχουν την ίδια τιμή με βάση τη δεύτερη συνάρτηση κατακερματισμού. Αυτό συμβαίνει έτσι ώστε να μειωθεί το πλήθος των στοιχείων των οποίων θα ελεγχθούν για ζεύξη στη Radix Hash Join.

Τέλος, εφόσον δημιουργήθηκαν οι κατάλληλες δομές, εκτελείται ο αλγόριθμος Radix Hash Join για τη ζεύξη των δύο πινάκων. Για τον αλγόριθμο αυτό υλοποιήθηκε μια δομή για την αποθήκευση των αποτελεσμάτων σε μια μορφή buffer, του οποίου το μέγεθος κάθε κόμβου είναι ίσο με 1Mb. Για την εξασφάλιση ότι κάθε κόμβος θα έχει μέγεθος 1Mb, αποθηκεύεται σε κάθε κόμβο το πληθος των στηλών που βρίσκονται στα αποτελέσματα. Συνεπώς με βάση το τύπο: *1024 / (buffer_size * sizeof(int32_t)),* Εξάγεται κάθε φορά το μέγεθος του buffer. Κάθε φορά που προστίθεται ένας πίνακας στα αποτελέσματα αυξάνεται και το buffer_size.

Μετά την εξαγωγή των αποτελεσμάτων γίνονται οι κατάλληλες αποδεσμεύσεις μνήμης.

Μέρος Δεύτερο

Το δεύτερο μέρος της εργασίας, αφορούσε την ανάγνωση των δεδομένων πινάκων προς join, την αποθήκευσή τους στην μνήμη. Ακόμα απαιτούσε την ανάγνωση μερικών ερωτημάτων πάνω σε αυτά τα δεδομένα και την κατάλληλη επεξεργασία τους, ώστε να γίνει αντιληπτή η εντολή τους και τέλος η εκτέλεσή τους για την παραγωγή αποτελεσμάτων

Σε αυτό το σημείο οι αρμοδιότητες χωρίστηκαν σε τρία βασικά σημεία : πρώτο στην ανάγνωση και αποθήκευση των δεδομένων, δεύτερο στην ανάγνωση, επεξεργασία και αποθήκευση των εντολών και τρίτο στην εκτέλεση των εντολών αυτών.

Πλέον εδώ η ομάδα είχε τα δύο βασικά μέλη, με το τρίτο να έχει συνεισφέρει οικειοθελώς στην ολοκλήρωση του μέρους αυτού. Εδώ στην συνεργασία αποφασίστηκε να χωριστούν αρμοδιότητες για την ταχύτερη εκτέλεση της εργασίας.

Δυσκολίες εντοπίστηκαν στην διαχείριση ερωτημάτων που επαναλαμβάνεται ένας πίνακας παραπάνω από μία φορά, καθώς έπρεπε να γίνει διαχείρισή του σαν να ήταν ξεχωριστός πίνακας. Ευτυχώς, το parsing των εντολών το διαχειριζόταν εξ'ολοκλήρου αυτό, χωρίς να πρέπει να γίνει κάποια αλλαγή στον κώδικα.

Τροποποιήθηκε εξ ολοκλήρου το σύνολο εισόδου δεδομένων, καθώς έγινε διαθέσιμο το input του διαγωνισμού. Δεδομένου αυτού, υλοποιήθηκε η συνάρτηση loadTables της οποίας ρόλος ήταν να διαβάσει το αρχείο small.init το οποίο περιέχει τα ονόματα των αρχείων των πινάκων τα οποία βρίσκονται σε binary μορφή. Ακολούθως, με τη χρήση της συνάρτησης mmap() δεσμεύεται το κατάλληλο μέγεθος με βάση τα δεδομένα που υπάρχουν στο αρχείο πέρι πλήθους γραμμών και στηλών και τελικά γίνεται η αποθήκευση κατα στήλες των πινάκων.

Μετέπειτα, αποφασίστηκε να δημιουργηθούν όλα τα ευρετήρια που είχαν περιγραφθεί στο πρώτο μέρος για κάθε πίνακα και για κάθε στήλη του πίνακα, έτσι ώστε να μη γίνεται έλεγχος κατά τη διάρκεια της εκτέλεσης ερωτημάτων αν ο πίνακας που εμπλέκεται έχει ευρετήριο στη συγκεκριμένη στήλη.

Για τη μοντελοποίηση των ερωτημάτων στο πρόγραμμα δημιουργήθηκε μια λίστα από λίστες, όπου όλο το αρχείο με τα ερωτήματα αποτελεί τη λίστα queries η οποία περιέχει τη λίστα από batches, που κάθε κόμβος αυτής της λίστας αντιστοιχεί σε κάθε batch του αρχείου, όπου κάθε κόμβος περιέχει μία λίστα instruction που αντιστοιχεί σε κάθε ερώτημα του batch. Στη συνέχεια κάθε instruction περιέχει τρεις μικρές λίστες που αντιστοιχούν στα τρία μέρη ενός ερωτήματος, δηλαδή, τους πίνακες που εμπλέκονται στο ερώτημα, το τμήμα where και το τμήμα των προβολών. Τέλος για το τμήμα where κάθε κόμβος της λίστας περιέχει και μία πράξη μεταξύ των πινάκων όπου αποθηκεύονται οι πίνακες που συμμετέχουν σε κάθε πράξη και τη πράξη.

Ακόμα άλλαξε σε μεγάλο βαθμό η υλοποίηση της συνάρτησης RadixHashJoin() η οποία υλοποιεί τον αντίστοιχο αλγόριθμο προκειμένου να μπορεί να εκτελέσει ζεύξη πινάκων οι οποίοι δεν έχουν ξανασυμμετάσχει σε κάποια άλλη ζεύξη του ερωτήματος, είτε κάποιος από τους εμπλεκόμενους βρίσκεται ήδη στα ενδιάμεσα αποτελέσματα. Στη δεύτερη περίπτωση, διατρέχουμε κάθε στοιχείο που βρίσκεται σε κάθε κόμβο του buffer που περιγράφθηκε προηγουμένως, και με βάση τη τιμή που θα δώσει η πρώτη

συνάρτηση κατακερματισμού, εξετάζεται ο κάδος του καινούριου πίνακα του οποίου τα στοιχεία έχουν την ίδια τιμή σύμφωνα με τη συνάρτηση κατακερματισμού.

Για τις συναρτήσεις που εκτελούν κάποια πράξη φίλτρου (<,>,=) στην υλοποίηση μας διατρέχεται ο πίνακας κατα γραμμές, και οποίο στοιχείο ικανοποιεί τη συνθήκη αποθηκεύεται στο buffer. Στη ζεύξη στηλών του ίδιο πίνακα η λογική είναι ίδια, δηλαδή αποθηκεύονται στον buffer μόνο οι γραμμές οι οποίες έχουν ίδιες τιμές στις στήλες που υποδεικνύονται στη πράξη.

Ακόμα, υλοποιήθηκε μια ουρά προτεραιότητας για τη σειρά εκτέλεσης των πράξεων σε ένα ερώτημα. Η κατεύθυνση που ακολουθήθηκε είναι πως για κάθε ζεύξη μεταξύ πινάκων, θα ελέγχεται το ερώτημα αν υπάρχει φίλτρο για κάποιους από τους πίνακες που εμπλέκονται. Αν ναι, δίνεται προτεραιότητα στα φιλτρα.

Τέλος, για την εκτέλεση κάθε ερωτήματος υλοποιήθηκε η συνάρτηση executeQuery() η οποία δέχεται την ουρά προτεραιότητας και τους πίνακες μαζί με τα ευρετήρια τους, και με βάση την ουρά προτεραιότητας που περιγράφθηκε παραπάνω εκτελεί τη κάθε πράξη και κρατά δύο δείκτες, έναν για το αποτέλεσμα της προηγούμενης πράξης που εκτελέστηκε και έναν για αυτόν που πρόκειται να εκτελεστεί τώρα. Κατά την εκτέλεση ενός ερωτήματος, αποθηκεύονται κάποια μεταδεδομένα όσον αφορά ποιοι πίνακες έχουν εμφανιστεί μέχρι στιγμής στην εκτέλεση του, καθώς και τη σειρά εμφάνισης τους για να καλυφθεί και η περίπτωση δύο διαφορετικων μεταβλητών σε ένα πίνακα. Στο τέλος εκτέλεσης ενός ερωτήματος, υπολογίζονται οι προβολές που ζητούνται στις κατάλληλες στήλες των αντίστοιχων πινάκων.

Μέρος Τρίτο

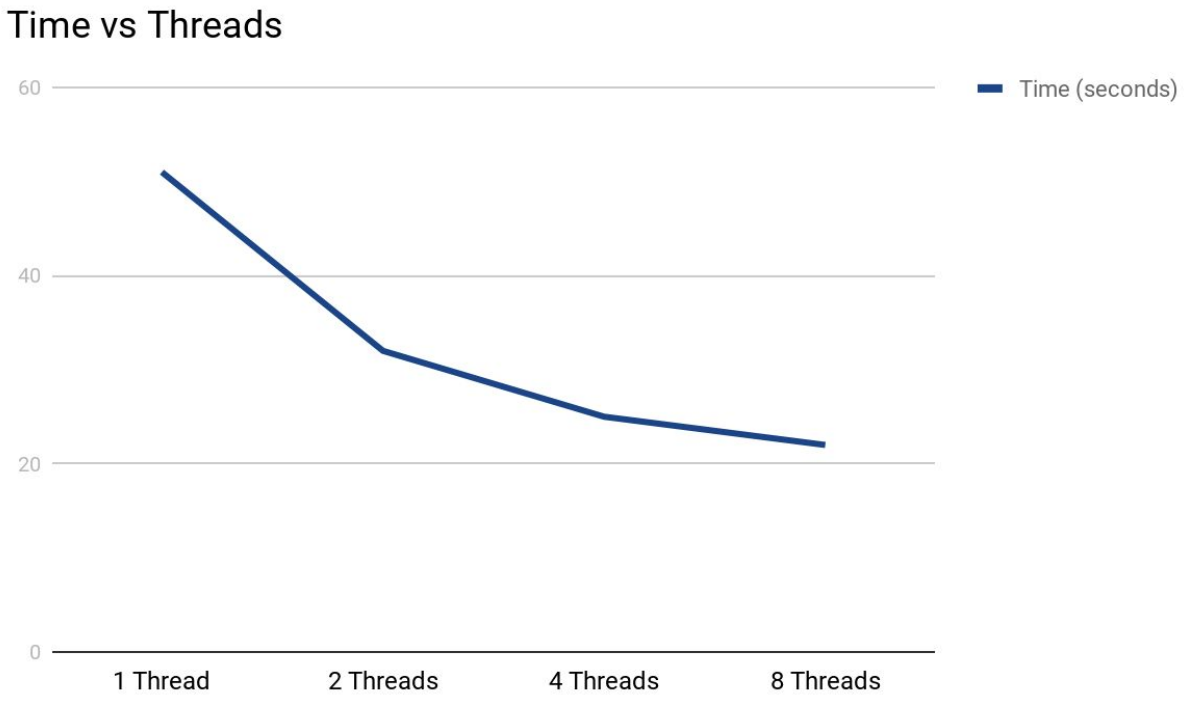
Το τρίτο μέρος της εργασίας, αφορούσε την βελτιστοποίηση της υπάρχουσας υλοποίησης σε δύο βασικά σημεία : στον χρόνο εκτέλεσης και στον χώρο αποθήκευσης. Για αυτό το λόγο η εκφώνηση της άσκησης

έδινε σαν ζητούμενα την παραλληλοποίηση της εκτέλεσης του κώδικα και την επιλογή της βέλτιστης σειράς εκτέλεσης των join ανά query με την χρήση του αλγορίθμου Join Enumeration.

Η ομάδα που ολοκλήρωσε το τρίτο και τελευταίο μέρος της εργασίες ήταν τα δύο άτομα, με το τρίτο να προσφέρει την βοήθειά του σε ορισμένα σημεία. Ξανά, όπως και στο προηγούμενο μέρος της εργασίας, έγινε καταμερισμός εργασιών, ώστε να γίνει καλύτερη διαχείριση της εργασίας και ταχύτερη ανάπτυξη.

[Παραλληλοποίηση]

Εδώ παραλληλοποιήσαμε 3 αλγοριθμους της διαδικασίας indexing kai εξυπηρέτησης των queries έτσι ώστε να πετύχουμε ταχύτερα αποτελέσματα. Πιο συγκεκριμένα φτιάξαμε έναν Job Scheduler ο οποίος έχει στη διάθεση του ενα thread pool με 1/2/4/8 threads που το κάθε ένα δέχεται και εκτελεί κάποια jobs. Τροποποιήσαμε κατάλληλα τις συναρτήσεις δημιουργίας Ιστογραμμάτων, μεταφοράς δεδομένων από τον r στον r' καθώς και την διαδικασία του join έτσι ώστε να μπορούν να διαμεριστούν σε τμήματα και να γίνουν παράλληλα. Τα αποτελέσματα ήταν αρκετά ικανοποιητικά αφού ενώ με 1 thread έχουμε τα αποτελέσματα όλων των queries σε χρονο 51s, αυξάνοντας σταδιακά τον αριθμό των thread παίρνουμε τελικά με 8 threads χρόνο 22s! Παρακάτω μπορούμε να δούμε καθαρά αυτή τη μετάβαση.



[Join Enumeration]

Για την βέλτιση σειρά εκτέλεσης των joins, χρησιμοποιήθηκαν ορισμένα στατιστικά και ο αλγόριθμος Join Enumeration. Αφού γίνεται δημιουργία της λίστας προτεραιότητας των predicates, εκτελείται υπολογισμός των στατιστικών για αυτά που αφορούν μόνο φίλτρα. Αφού γίνει αυτή η διαδικασία και ανανεωθούν τα

στατιστικά για τις στήλες και τους πίνακες που αφορούν το συγκεκριμένο query προς εκτέλεση, τότε καλείται η join enumeration.

Μέσα σε αυτή, και σύμφωνα με τον αλγόριθμο που έχει δωθεί, γίνεται κατηγοριοποίηση των joins. Για κάθε πίνακα που σχετίζεται με το query, δημιουργείται ένα hash table Best Tree, όπου αποθηκεύεται ανάλογα το hash function, το κόστος του συγκεκριμένου συνδυασμού και το κόστος του. Το hash function που υλοποιήθηκε εκμεταλλεύεται την εκθετική άνοδο του παραγωντικού προκειμένου να αποφευχθούν οποιαδήποτε collisions στο αποτέλεσμα της συνάρτησης.

Αφού αρχικοποιηθεί το hash table, για κάθε συνδυασμό αυξανόμενου μεγέθους από την λίστα των πινάκων προς join, ελέγχεται το κόστος και εντοπίζεται ο τελικός συνδυασμός που είναι ο βέλτιστος. Έτσι, επιστρέφεται στην parsing_unit η λίστα με την βέλτιστη σειρά εκτέλεσης των ερωτημάτων.