Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών

Πληροφορικής και Τηλεπικοινωνιών

Ανάπτυξη Λογισμικού για Πληροφοριακά Συστήματα

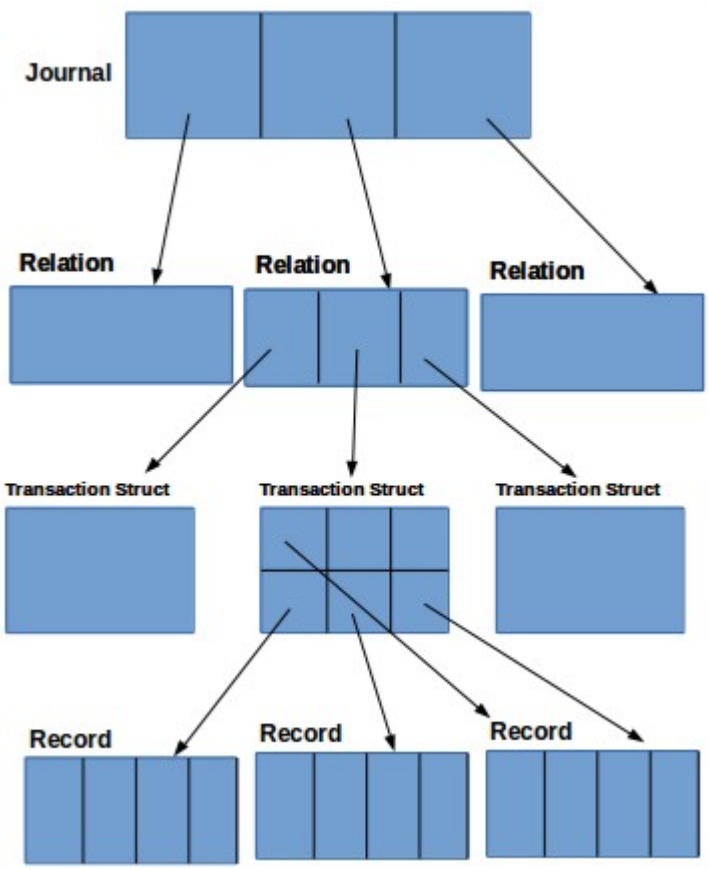
Έτος 2015-16

Νικόλαος ΜαρούληςΑ.Μ. :1115201000212
Οδυσσεύς Μπόρας Α.Μ. :1115201000096

ΠΕΡΙΕΧΟΜΕΝΑ

1. Σχεδιασμός

Το ημερολόγιο(**Journal**) διαχείριζεται τις σχέσεις που έχουμε και για κάθε μια δημιουργεί ένα αντικείμενο τύπου relation. Κάθε **relation** διαχειρίζεται αντικείμενα τύπου **transactionStruct**. Κάθε transactionStruct είναι μια δομή που κρατάει έναν αριθμό (transactionId), έναν πίνακα με όλα τα records που μπήκαν σε αυτό το transaction απο εισαγωγή, και έναν άλλο πίνακα με Record*, δηλαδή δείκτες στα αντίστοιχα records εισαγωγής για την διαγραφή. Κάθε **Record** είναι η εγγραφή που έχει έναν πίνακα με αριθμούς, ανάλογα με το αριθμό των πεδίων της σχέσης.



2. Ευρετήρια

2.1 Ευρετήριο με βάση το κλειδί.

Το ευρετήριο αυτό, είναι φτιαγμένο σύμφωνα με τις αρχές του επέκτατου κατακερματισμού. Χρησιμοποιείται σε δύο περιπτώσεις.
1. Όταν θέλουμε να διαγράψουμε κάποιο record,
2. Όταν θέλουμε να υπολογίσουμε κάποιο validation και το predicate αναφέρεται σε κλειδί.

Το ευρετήριο αυτό, μεταξύ άλλων, κρατάει και δείκτες στις θέσεις μνήμης των εγγραφών. Οπότε όταν εντοπίσουμε το predicate, κοιτάμε κατευθείαν στην πραγματική θέση μνήμης που είναι αποθηκευμένο.

2.2 Ευρετήριο για τα transactions struct.

Η υλοποίηση μας, κράταγε τα transactions struct σε έναν πίνακα. Σύμφωνα με τις απαιτήσεις του 2ο μέρους της εργασίας, φτιάξαμε ένα dummy hash, το οποίο αποθηκεύει το transaction id, και την θέση που βρίσκεται στον πίνακα.

Παρατηρήσαμε ότι η δυαδική αναζήτηση στον πίνακα που είναι τα transactions, είναι αρκετά πιο γρήγορη απο το να χρησιμοποιούμε αυτό το ευρετήριο.

2.3 Ευρετήριο για τα predicates.

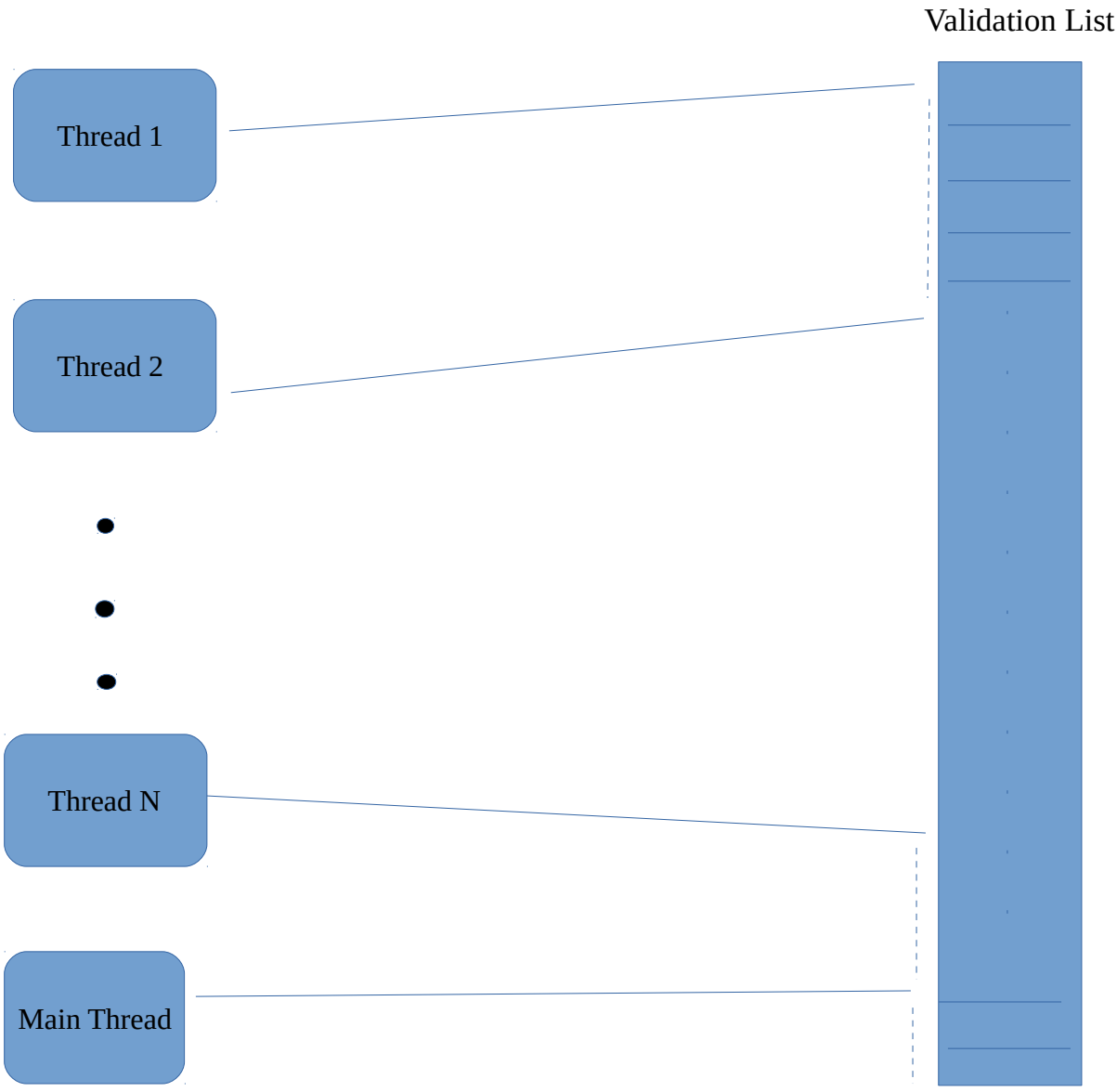
Επειδή τα predicates δεν είναι μοναδικά, και κάποιο μπορεί να εμφανιστεί παραπάνω απο μια φορά σε διαφορετικό validation, υλοποιήσαμε ενα ακόμα hash. Αν χρησιμοποιούμε αυτό το hash, τότε τα validations κρατάνε δείκτες σε predicates, οπότε για ενα predicate μπορεί παραπάνω απο ένα να δείχνουν σε αυτό.
Ο λόγος που χρησιμοποιούμε αυτή την λογική είναι για να υπολογίζουμε κάποιο predicate μόνο μία φορά και άμα ξαναέρθει να μην χρειαστεί να το ξαναυπολογίσουμε.Επίσης έχει υλοποιηθεί η forget, η οποία κρατάει σε μια άλλη λίστα τα predicates που είναι για διαγραφή, και μόλις έρθει η forget γίνεται προσπέλαση αυτής της λίστας ώστε να αποφασιστεί ποιο predicate θα διαφραφεί και ποιο όχι.

3. Threads

Για την υλοποίηση χρησιμοποιήθηκαν POSIX Threads, καθώς και για τον συγχρονισμό τους χρησιμοποιούμε conditionals και mutexes.Στην main γίνεται η δημιουργία των threads με link στην συνάρτησή τους threadController.Η δήλωση των threads είναι global και ο αριθμός τους γίνεται define, ένω καταλήξαμε ότι ο καλύτερος αριθμός threads είναι 7.
Διαμοιρασμός των Jobs στα Threads:

Το κάθε validation έχει ένα κόστος, το οποίο βρίσκεται μέσω ευρετικής συνάρτησης και το οποίο αντιπροσωπεύει το πόσο κόστος θέλει για να υπολογιστεί.Το κόστος υπολογισμού υπολογίζεται μέσω ευρετικής η οποία συνήθως λαμβάνει υπ'όψιν το εύρος των transactions που έχει να κοιτάξει, τον αριθμό των queries κτλ. Μία global μεταβλητή δείχνει το συνολικό κόστος που έχει μαζευτεί σε μία συγκεκριμένη flush, οπότε διαιρώντας το συνολικό κόστος με τον αριθμό των διεργασιών υπολογίζουμε το πόση δουλειά θα κάνει ο καθένας.

Η main έπειτα απ'αυτούς τους υπολογισμους όταν έρθει η flush, κατανέμει ομοιόμορφα τη δουλειά των threads και τους την δίνει, πάντα κλειδώνοντας mutexes για αφάλεια γραφής/ανάγνωσης, καθώς κρατάει και ένα 10% της δουλειάς για την ίδια, ώστε να μην περιμένει απλά τα threads να τελειώσουν.Μόλις όλα τα threads τελειώσουν τη δουλειά τους , δηλαδή να υπολογίσουν τα validations και να γράψουν το αποτέλεσμα στην λίστα τον validations, η main ξυπνάει και με μία προσπέλαση της λίστας εμφανίζει τα αποτελέσματα.



4. Βελτιστοποιήσεις

1. Όπου χρείαζεται να αναζητήσουμε κάτι κάνουμε πάντα δυαδική αναζήτηση.

2. Στην κατασκευή των validations, για κάθε query, βάζουμε πάντα το κλειδί πρώτο, με την λογική ότι θα ψάξει απο το hash και θα βρεί την απάντηση πιο γρήγορα.

3. Sort για τα queries. Τοποθετούμε τα μικρότερα πρώτα, επειδή έχουνε μεγαλύτερη πιθανότητα να απαντήσουν true μέσω quicksort.

4. Αν κάποια predicates είναι αμοιβαίος αποκλειόμενα μεταξύ τους σε ένα query τότε γράφουμε κατευθείαν false.

5. Χρήση ευρετικής συνάρτησης για τον υπολογισμό κόστους του κάθε validation, με αποτέλεσμα ίσο μοίρασμα της δουλειάς σε κάθε thread.

6. Διαμοιρασμός της δουλειάς στα threads με βάση τη σειρά που τα παίρνουν , ώστε το καθένα να χει διαφορετική βαρύτητα.

5. Συμπεράσματα - Χρόνοι

- Έχει χρησιμοποιηθεί valgrind για έλεγχο των memory leaks και άλλων πιθανών bugs.
- Η μνήμη που δεσμεύεται κατά την εκτέλεση του προγράμματος, αποδεσμεύεται στο τέλος.

Το makefile μπορεί με το κατάλληλο όρισμα να δημιουργήσει και το αντίστοιχο εκτελέσιμο, το default compile που γίνεται με το make χρησιμοποιεί τα flags -pthread και -03.Επίσης για να τρέξει το πρόγραμμα με κάποιες συγκεκριμένες λειτουργίες μπορούν να προστεθούν κάποια flags μετά το όνομα του εκτελέσιμου.
Τα flags της εκτέλεσης είναι τα εξής:

1. -rel_hash : Το πρόγραμμα δημιουργεί και χρησιμοποιεί για τον υπολογισμό των validations ένα hash στα transactions του κάθε relation
2. -pre_hash : Το πρόγραμμα χρησιμοποιεί το hash των predicates
3. -forget : Το πρόγραμμα όταν του έρθει forget απον parser,διαγράφει τα predicates που δεν χρειάζονται
4. -sort : Το πρόγραμμα ταξινομεί τα queries, με το μικρότερο πρώτο
5. -mutex : Το πρόγραμμα χρησιμοποιεί αμοιβαίο αποκλεισμό στα predicates τψν validations
6. -threads : Το πρόγραμμα χρησιμοποιεί threads
7. -time : Το πρόγραμμα εμφανίζει το user time εκτέλεσης

Χρόνοι Εκτέλεσης :

Μετρήθηκαν σε υπολογιστή με Arch Linux OS 64-bit και core i5 πέμπτης γενιάς στα 3.2 gHz 4 πυρήνων(χωρίς hyperthreading, δηλαδή 4 threads)και RAM 16 GB ddr4

Flags	Small.bin	Medium.bin
No flags	0.95 sec	21.8 sec
No flags (Threads)	0.74 sec	***7.55 sec***
Relation Hash	1.35 sec	26.9 sec
Relation Hash (Threads)	0.83 sec	9.01 sec
Predicate Hash	6.5 sec	5.41 min
Predicate Hash (Threads)	5.37 sec	1.54 min
Predicate Hash & Forget	6.5 sec	5.41 min
Predicate Hash & Forget(Threads)	5.37 sec	1.54 min
Sort	0.84 sec	22.3 sec
Sort (Threads)	0.69 sec	7.85 sec
Mutual Exclusion	0.86 sec	22.2 sec
Mutual Exclusion(Threads)	0.675 sec	7.95 sec
Sort & Mutual Exclusion	0.825 sec	22.7 sec
Sort & Mutual Exclusion(Threads)	***0.652 sec***	8.01 sec