Ανάπτυξη Λογισμικού για Πληροφοριακά Συστήματα

2015 - 2016

Ονοματεπώνυμο	Α.Μ
Γαμβρινός Σταυράγγελος	1115201100006
Καββαθάς Δημήτριος	1115201100161
Σεϊνταρίδης Δημήτριος	1115201100197

Οδηγίες Χρήσης

Για το compile του προγράμματος υπάρχει makefile. Εκτός από το main Project έχουμε υλοποιήσει και μια main μέσω της οποίας μπορούμε να δούμε στατιστικά για τα δεδομένα των αρχείων. Για το compile της main με τα στατιστικά υπάρχει η επιλογή “make stats” του makefile.

Οι επιπλέον δομές κάθε επιπέδου μπορούν να ενεργοποιηθούν/απενεργοποιηθούν μέσω του αρχείου *options.hpp.*

Επίπεδα Project

Παρακάτω, παρατίθενται συγκεκριμένες σχεδιαστικές επιλογές και βελτιώσεις που υλοποιήθηκαν σε κάθε επίπεδο του Project.

1ο Επίπεδο

1. Η δομή του Journal υλοποιείται μέσω των κλάσεων Journal και JournalRecord. Καθώς έρχονται τα transactions, κατασκευάζουμε τα JournalRecords και τα προσθέτουμε στο αντίστοιχο Journal object.

2. Καθώς έρχονται τα validations στην συνάρτηση processValidations, τα δεδομένα των validations αποθηκεύονται σε objects των κλάσεων ColumnClass, queryClass και ValClass. Η επιλογή αυτή έγινε διότι η χρήση wrapper κλάσεων γύρω από τα έτοιμα structs των processes δίνει τη δυνατότητα πέρα από την αποθήκευση των validations, να κρατάμε και επιπλέον πληροφορία, όπως αν ένα validation είναι validated κλπ.

3. Τα validations στην processValidations αποθηκεύονται σε έναν 1-διάστατο πίνακα έτσι ώστε να γίνει ο απαραίτητος υπολογισμός στην συνάρτηση processFlush. Αρχικά είχαμε υλοποιήσει την δομή αυτή ως list, αλλά στην συνέχεια αλλάξαμε την δομή σε πίνακα για να εξασφαλίσουμε περισσότερα cache hits και συνεπώς να βελτιώσουμε τον χρόνο όπως προτάθηκε στο μάθημα. Μετά την αλλαγή αυτή, υπήρξε πράγματι μια μικρή βελτίωση στον χρόνο.

4. Όταν έρχεται το flush, τα validations υπολογίζονται μέσω της συνάρτησης valOptimize. Στην συνάρτηση αυτή, το 1ο optimization που κάνουμε είναι να ταξινομήσουμε τα queries ενός validation με βάση το column count. Δηλάδη, τοποθετούμε τα queries από το μικρότερο στο μεγαλύτερο. Η επιλογή αυτή έγινε έτσι ώστε να ανιχνευθεί με όσο το δυνατόν λιγότερες πράξεις το coflict (εάν το validation προκαλεί conflict). Το 2ο optimization αποτελείται απο 3 στάδια. Πιο συγκεκριμένα, έχουμε υλοποιήσει 3 priority arrays. Στο 1ο αποθηκεύονται όλα τα columns της μορφής “c0=<x>”. Στο 2ο αποθηκεύονται όλα τα columns της μορφής “ci=<x> όπου i != 0” και στο 3ο αποθηκεύονται όλα τα υπόλοιπα, >, >=, <, <=, != . Τα columns ελέγχονται με την σειρά προτεραιότητας. Συγκεκριμένα στο priority1 εκμεταλλευόμαστε την άμεση εύρεση του journal record μέσω του keyhashtable ενώ στο priority2 η ισότητα είναι πιθανότερο να αποκλείσει περισσότερα journal records από ότι οι πράξεις που υπάρχουν στο priority3. Οι επιλογές αυτές οδήγησαν σε πολύ σημαντική μείωση του χρόνου υπολογισμού των validations.

2ο Επίπεδο

1. Στο πρώτο μέρος του 2ου επιπέδου υλοποιήθηκε ο Tid HashTable ο οποίος αποθηκεύει την πρώτη θέση του κάθε transaction επιστρέφοντας το offset του. Μπορεί να ενεργοποιηθεί και να απενεργοποιηθεί μέσω του options.hpp. Η χρήση του αντικαθιστά τη binary search που χρησιμοποιείται για να βρεθεί το offset του κάθε tid κάνοντας την εύρεση πιο γρήγορη σε αυτό το σημείο. Η ενεργοποίηση του μειώνει τον χρόνο ειδικά στο medium.

2. Στο δεύτερο μέρος υλοποιήθηκε ο Val HashTable βάση του οποίου αποθηκεύονται ως keys όλα τα subqueries και το range τους μέσω

μετατροπής τους σε string και ως data το bitset που αναπαριστά τα journal records στο range αυτό. Στόχος αυτής της υλοποίησης ήταν να εκμεταλλευτούμε την περίπτωση όπου θα εμφανιστεί ξανά ένα ίδιο subquery με ίδιο range στο ίδιο Relation. Δυστυχώς αυτό το ποσοστό ήταν πολύ μικρό και ενδεικτικά στο small που μετρήθηκε δεν ξεπερνούσε το 3%. Αυτό είχε ως αποτέλεσμα όχι μόνο να μη βελτιώσει τον χρόνο αλλά να τον αυξήσει κατά πολύ και να εκτοξεύσει τη χρήση της μνήμης. Μετά από αλλαγές στην προσέγγιση του forget και της διαγραφής στον hashtable ώστε να μη γεμίζει η μνήμη στο medium καταφέραμε να το υλοποιήσουμε αλλά σε μη συναγωνιστικό χρόνο.

3ο Επίπεδο

1. Στο τρίτο επίπεδο προστέθηκε η δυνατότητα παράλληλου υπολογισμού των validations πάνω στην υλοποίηση του 1ου επιπέδου. Η ενεργοποίηση της λειτουργίας αυτής γίνεται μέσω του αρχείου options.hpp.

   Η αρχική ιδέα με βάση την εκφώνηση ήταν να παίρνει κάθε thread ένα συγκεκριμένο υποσύνολο των validations και να ασχολείται μόνο με αυτό. Η υλοποίηση αυτή όμως έχει το αρνητικό ότι εάν ένα thread πάρει ένα “πακέτο” από validations το οποίο μπορεί να υπολογιστεί πολύ γρηγορότερα σε σχέση με τα υπόλοιπα, τότε το thread αυτό θα τελειώσει πιο γρήγορα και θα παραμείνει αδρανές, μη αξιοποιώντας έτσι την υπολογιστική του ικανότητα. Επιπλέον κάθε thread καταστρέφεται και δημιουργείται εκ νέου, γεγονός που οδηγεί σε επιπλέον αύξηση του χρόνου του προγράμματος

   Με αφορμή τα παραπάνω, ακολουθήσαμε μια λογική thread pool όπου τα threads τραβάνε συνεχώς validations. Τα threads δημιουργούνται και περιμένουν σε ένα condition variable μέχρις ότου το main thread προσθέσει validations προς υπολογισμό. Όταν το κάνει, ξυπνάει τα worker threads και εκείνα αρχίζουν και τραβάνε validations από την queue. Η υλοποίηση αυτή εξαλείφει το προηγούμενο πρόβλημα καθώς πλέον εάν ένα thread τελειώσει γρήγορα, θα πάρει άμεσα το επόμενο validation. Πρέπει να τονίσουμε ότι υπάρχει φυσικά “ανταγωνισμός” των threads για το mutex που επιτρέπει την ασφαλή αφαίρεση validations από το queue, αλλά αυτός δεν προσθέτει ιδιαίτερη χρονική επιβάρυνση. Η υλοποίηση αυτή οδήγησε σε ~40% μείωση του χρόνου συγκριτικά με το 1ο επίπεδο.

2. Τέλος προσθέσαμε τη δυνατότητα επιλογής των rounds στο flush, δηλαδή κάθε πόσα flush τα validations θα υπολογίζονται και θα εκτυπώνονται. Η επιλογή βρίσκεται στο options.hpp και η default τιμή είναι να εκτυπώνονται σε κάθε flush. Η δυνατότητα αυτή υλοποιήθηκε με στόχο τη μέγιστη

εκμετάλλευση των threads καθώς κρατώντας validations από κάθε flush τα δεδομένα αυξάνονται και τα threads ξυπνάνε λιγότερες φορές. Η βελτίωση στον χρόνο του small ήταν αμελητέα ενώ στο medium με 5 rounds υπήρχε μείωση περίπου ενός δευτερολέπτου.

Χρόνοι & Μνήμη

Χρόνοι:

	1ο & 3ο Επίπεδο			**2ο Επίπεδο**	
	default	Tid Hashtable	Threads	default	Tid Hashtable
small	1.9s	1.7s	1.14s	5.5s	5.2s
medium	51s	38.2s	19.6s	301s	284s

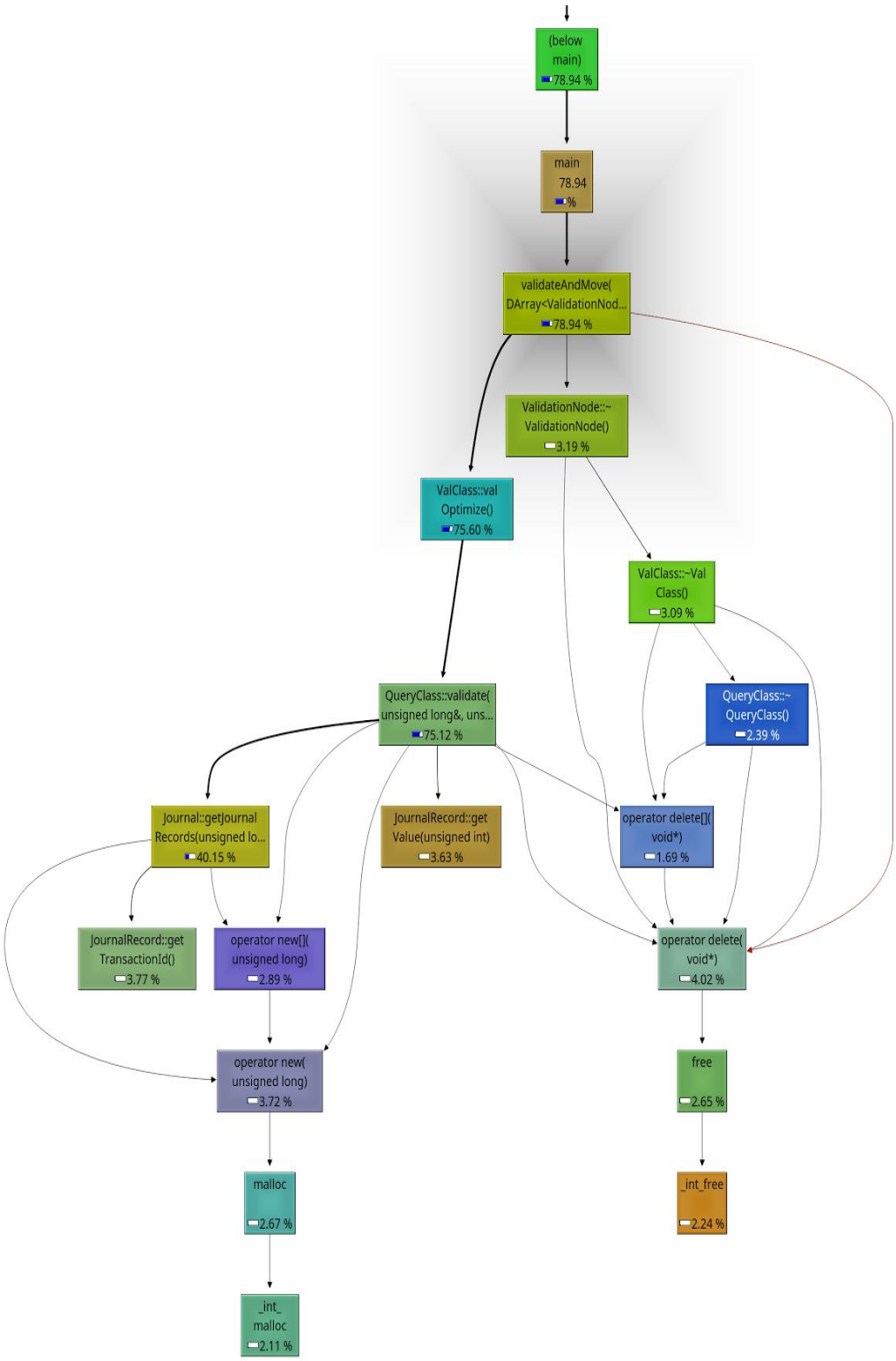
Μνήμη:

	1ο & 3ο Επίπεδο			**2ο Επίπεδο**	
	default	Tid Hashtable	Threads	default	Tid Hashtable
small	170mb	170mb	170mb	510mb	510mb
medium	800mb	830mb	830mb	5.7gb	5.7gb

PC Specs

CPU: Intel Core i5 4670k 3.40GHz
Ram: 12Gb 2400MHz

CPU Usage graph.



Προτάσεις για περαιτέρω βελτίωση

Κάποιες περαιτέρω βελτιώσεις που θα υλοποιηθούν μέχρι την προφορική εξέταση είναι:

1. Υλοποίηση hash table για τα c1 και c2 columns. Η υλοποίηση αυτή θεωρούμε ότι θα μειώσει επιπλέον τον χρόνο καθώς στα datasets που δίνονται παρατηρήσαμε ότι αθροιστικά το ποσοστό των ερωτήσεων ισότητας στα c1 και c2 είναι πολύ κοντά στο ποσοστό ερωτήσεων ισότητας στο c0 (key). Ενδεικτικά στο medium το ποσοστό των c1 και c2 είναι συνολικά ~11% και του c0, ~6.1%.

2. Σορτάρισμα των queries με βάση το relation, ώστε να κρατάμε τα journal records και να μην τα ξανα ψάχνουμε σε περίπτωση που το επόμενο query είναι στο ίδιο relation.

3. Χρήση πιο έξυπνου job scheduler ώστε να ομαδοποιεί τα δεδομένα στον πίνακα με στόχο περισσότερα cache hits.