ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Κ23α – Ανάπτυξη Λογισμικού Για Πληροφοριακά Συστήματα

Χειμερινό Εξάμηνο 2018-2019

Ζήσης Μπεληγιάννης 1115201400118

Παναγιώτης Στασινός 1115201400191

ΠΕΡΙΕΧΟΜΕΝΑ

Thread Pool

Το πληθος των thread που δημιουργει η συναρτηση thread_pool_create() καθοριζεται με #define μεσα στο αρχείο thread_pool.h .

Οποιαδήποτε στιγμη μόνο ένα thread έχει πρόσβαση για να εισάγει ή να εξαγει ενα job στην ουρά , αυτο επιτυγχάνεται με την χρηση του mutex queue_mutex

Τα thread με την δημιουργία τους εκτελουν την συναρτηση ready() στην οποια οσο το thread_pool ειναι awake ζητούν απο την ουρά jobs

Χρονομέτρηση

Οι χρονομετρήσεις έγιναν στα workstation της σχολής

Χρονομέτρηση Σειριακού

Χρονομέτρηση Παραλληλοποιημένου

Συναρτησεις

parse.c :

- void information_storing(struct file_info* info, uint64_t * data, struct file_stats * fstats);
  λειτουργία :
- void print_query_info(struct query_info* query);
  λειτουργία :
- void insert_pred(struct query_info* query, char* pred, int index);
  λειτουργία :

query.c :

- void calculate_query();
- result* comparison_query();
- int calculate_priority();
- void calculate_sum();
- void print_sum();
- int predicate_exists();

utils.c :

- void update_results();
  λειτουργία :
- void create_relation();
  λειτουργία :
- void create_rel_from_list();
  λειτουργία :
- void create_interlist();
  λειτουργία :
- void list_to_rel_with_indexes();
  λειτουργία :
- void copy_result();
  λειτουργία :
- void update_interlists();
  λειτουργία :
- void create_rel_from_list_distinct();
  λειτουργία :
- void update_results_filter();
  λειτουργία :
- void update_interlists_filter();
  λειτουργία :
- void update_existing_interlists();
  λειτουργία :
- void relation_similarity();
  λειτουργάα :
- void updateDifferCol();
  λειτουργία :
- void update_interlists_new();
  λειτουργία :
- void list_to_relation();
  λειτουργία :

thread_pool.c :

- **int thrd_pool_wait(struct thrd_pool *temp);**
  λειτουργία : *μπολκαρει το main thread μεχρι να μην εχει μείνει καμία δουλειά στην job_queue και κανένα thread να μην εκτελεί ακόμα καποιο job.*
- **Int thrd_pool_create(struct thrd_pool *temp);**
  λειτουργία : *αρχικοποιεί τα mutexes της job queue και του μετρητή των active thread, το flag end σε μηδεν, δημιουγεί την job queue και στο τελος δημιουργει τα thread τα οποία εκτελούν την συναρτηση ready().*
- **Int thrd_pool_destroy(struct thrd_pool *temp);**

λειτουργία : *δίνει στην flag μεταβλητή end τιμή 1 ωστε να τερματίσουν ολα τα threads και στην συνέχεια περιμένει ολα τα threads να τερματίσουν. Τέλος κάνει destroy τα mutexes και τα semaphores.*

- **int thrd_pool_sleep(struct thrd_pool *temp);**
  λειτουργία : *μπλοκαρει τα thread μέσω του semaphore που αντιστοιχεί στο καθένα*
- **int thrd_pool_wake(struct thrd_pool *temp);**
  λειτουργία : *ξεμπλοκάρει τα thread ώστε να είναι έτοιμα να πάρουν jobs*

job_scheduler.c :

- **void *ready(void *args);**
  λειτουργία : *εκτελείται από τα thread κατα την δημιουργία τους. Κάθε thread εκτελεί την επανάληψη ζητώντας για job να εκτελέσει. Αν του επιστραφεί NULL δεν υπάρχει κάποια εργασία να κάνει και επιστρεφει στην αρχη της επανάληψης. Αν έχει κληθεί η thrd_poool_sleep() τότε μπλοκάρεται στο semaphore του και περιμενει για κληση της thrd_pool_wake() για να συνεχίσει, αλλιώς συνεχιζει να ζητάει για job μεχρι να πάρει. Αν λαβει job, αυξάνει την μεταβλητή των ενεργών thread κατά ένα, εκτελεί το job και στην συνέχεια μειώνει κατά ένα την μεαβλητή.*
- **int job_queue_create(struct job_queue *queue);**
  λειτουργία : *αρχικοποιεί την job queue με μήκος μηδέν*
- **int job_queue_push(struct thrd_pool *pool, int (*func)(void**), void **args);**
  λειτουργία : *καλείται απο το main thread για να εισάγει μια νεα δουλειά στην job_queue*
- **int job_queue_pull(struct job_queue *queue, struct job *temp_job);**
  λειτουργία : *καλείται από τα thread του pool για να λάβουν μια εγασια να κανουν, επιστρέφει στο temp_job->func NULL αν δεν υπαρχει job για εκτέλεση.*
- **int HistogramJob(void **args);**
  (int)args[0] , start_index για την αρχική relation*
  (int)args[1] , end_index για την αρχική relation*
  (struct histogram*)args[2] , *δείκτης του ιστογράμματος στο οποίο πρέπει να γράψει το συγκεκριμένο job*
  (struct relation*)args[3] , *δείκτης στην αρχική σχέση*
  (struct histogram*)args[4] , *δείκτης του αθροιστικού ιστογράμματος στο οποίο πρέπει να γράψει το συγκεκριμένο job*
  λειτουργία : *αρχικοποιεί το ιστογραμμα και το αθροιστικό ιστογραμμα και στην συνεχεια υπολογίζει το ιστόγραμμα για το κομμάτι απο το start μέχρι και το end*
- **int PartitionJobs(void **args);**
  (struct histogram*)args[0] , *δείκτης του ιστογράμματος από το οποίο πρέπει να διαβάσει το συγκεκριμένο job*
  (struct histogram*)args[1] , *δείκτης του αθροιστικού ιστογράμματος από το οποίο πρέπει να διαβάσει το συγκεκριμένο job*
  (struct relation*)args[2] , *δείκτης στην παλιά σχέση*
  (struct relation*)args[3] , *δείκτης στην ordered relation*
  (int)args[4] , start_index για την παλιά relation*
  (int)args[5] , end_index για την παλιά relation*

λειτουργία : *αντιγράφει τα tuples της παλιάς relation που βρίσκονται μεταξύ start και end στο κατάλληλο bucket της ordered relation με βάση τα hist και τα psum του συγκεκριμένου job.*

- **int JoinJobs(void **args);**
  (int)args[0] , start του συγκεκριμένου bucket για την ordered relation R*
  (int)args[1] , end του συγκεκριμένου bucket για την ordered relation R*
  (struct relation*)args[2] , *δείκτης στην ordered relation R*
  (int)args[3] , start του συγκεκριμένου bucket για την ordered relation S*
  (int)args[4] , end του συγκεκριμένου bucket για την ordered relation S*
  (struct relation*)args[5] , *δείκτης στην ordered relation S*
  (struct result*)args[6] , *δείκτης στο result στο οποίο γραφει το συγκεκριμένο job*
  λειτουργία : *εκτελεί το radix hash join για ένα συγκεκριμένο bucket*
- **int Sum(void **args);**
  (struct result*)args[0] , *δείκτης στο result που έχει τα indexes για συγκεκριμένο relation*
  (struct file_info*)args[1] , *δείκτης στις πληροφορίες των relation*
  (int)args[2] , αριθμός relation*
  (int)args[3] , αριθμός column*
  (uint64_t*)args[4] , *δείκτης στην μεταβλητή όπου θα αποθηκευτεί το άθροισμα*
  λειτουργία : *υπολογίζει το άθροισμα για την συγκεκριμένη σχέση και το συγκεκριμένο column και το γυρνάει πίσω μέσω του (uint64_t*)args[4*]

str.c :

- **result* RadixHashJoin(relation *relR, relation* relS, struct thrd_pool *temp);**
  λειτουργία: *επιστρέφει τον δείκτη των result της radix hash join. Κάνει wake up το thread pool και στην συνέχεια καλεί τις συναρτήσεις που δίνουν εργασίες για την στο thread pool για την δημιουργία των hist, psum και των τελικών αποτελεσμάτων.*
- **int Hist_and_Psum();**
  λειτουργία : *μοιράζει την δουλειά της δημιουργίας των hist και της αρχικοποίησης των psum. Στην συνέχεια αφού όλα τα thread έχουν τερματίσει και είναι έτοιμα τα ιστογράμματα δημιουργεί σε σειριακό κομμάτι τα psum.*
- **int ReOrdered();**
  λειτουργία **:** *μοιράζει το αρχικό relation σε πλήθος τμημάτων όσο και το πλήθος των thread και αναθέτει σε κάθε thread την συμπλήρωση διαφορετικού τμήματος της reordered. Τέλος περιμένει ολα τα thread να τερματίσουν ώστε να είναι έτοιμο το reodered relation και τερματίζει.*
- **int Join();**
  λειτουργία : *δημιουργεί πλήθος job όσο και το πλήθος των bucket των relation και περιμένει μέχρι να τερματίσουν. Αφού έχει τα αποτελέσματα από όλα τα job τα ενώνει σε ένα και τερματιζει. Αυτό το αποτέλεσμα είναι και το τελικό που επιστρέφει η RadixHashJoin().*

result.c :

- void result_init(result* result);
  λειτουργία :

- void insert_result(int rowR, int rowS ,result* result);
  λειτουργία :
- void print_result(result* result);
  λειτουργία :
- void free_result(result* res);
  λειτυοργία :
- void insert_inter(int row, result* result);
  λειτουργία :

query_selection.c :

- **void QueryOptimization (qinfo*, finfo*, fstats*, int *qselect);**
  λειτουργία : *κατασκευάζει το hashTable BestTree και εκτελεί τον δυναμικό αλγόριθμο ο οποίος επιστρέφει το καλύτερο συνδυασμό που χρησιμοποιεί όλες τις σχέσεις του query.*
- **void JoinEstimation(BestTree*, tree_node*, tree_node*, nodedata*);**
  λειτουργία : *βάση τα στατιστικά των δύο δέντρων που γίνονται Join κάνει την εκτίμηση για το νέο συνδυασμό σχέσεων που θα δημιουργηθεί από τα δύο προηγούμενα*.
- **void GreaterFilterEstimation(tree_node*);**
  λειτουργία: *υπολογίζει τα νέα στατιστικά της σχέσης που συμμετέχει σε predicate τύπου φίλτρου μεγαλύτερο (>) και ανανεώνει τις αντίστοιχες τιμές.*
- **void LessFilterEstimation(tree_node*);**
  λειτουργία: *ίδια λειτουργικότητα με την GreaterFilterEstimation αλλά για φίλτρα τύπου μικρότερο(<).*
- **void EqualFilterEstimation(treenode*);**
  λειτουργία: και *πάλι ίδια λειτουργικότητα με τις δύο προηγούμενες συναρτήσεις αλλά για φίλτρα τύπου ισότητας(=).*
- **void CreateJoinTreee(BestTree*, tree_node* , tree_node*);**
  λειτουργία: *Δημιουργία νέου δεντρού από την ένωση του παλιού δέντρου και του νέου. Στην υλοποίηση μας το νέο δέντρο αποτελείται από μία σχέση.Αρχικοποιεί δεδομένα του νέου δέντρου.*
- **void TreeInsert(BestTree*, char* key, nodedata*,);**
  λειτουργία: *εκχωρεί, με βάση το κλειδί που επιστρέφει η hash function, στο hash Table τον νέο συνδυασμό σχέσεων αφού πρώτα δεσμεύσει τον απαραίτητο χώρο.*
- **Tree_node* TreeSearch(BestTree*, char* key);**
  λειτουργία: *κάνει αναζήτηση στο hast Table το στοιχείο με το συγκεκριμένο key.*