Ανάπτυξη Λογισμικού Πληροφορικής
Χειμερινό Εξάμηνο 2015-2016

“Σύστημα ανίχνευσης συγκρούσεων σε επερωτήσεις βάσεων δεδομένων”
Επίπεδο 3

Ημερομηνία παράδοσης: 28/02/2016

Πίνακας Περιεχομένων

Γενική περιγραφή

Σκοπός αυτού του επιπέδου είναι η προσθήκη πολυνηματισμού κατά τη φάση των υπολογισμών. Η παραλληλία θα εφαρμοστεί στον υπολογισμό των validations, ο οποίος θα βασίζεται στην προσέγγιση του 1ου επιπέδου. Επιπλέον, στο επίπεδο αυτό θα γράψετε τη τελική αναφορά της εργασίας σας, η οποία θα συγκεντρώνει και θα τεκμηριώνει όλες τις σχεδιαστικές επιλογές σας.

Περιγραφή παραδοτέων τρίτου επιπέδου

1. Πολυνηματισμός στον υπολογισμό των validations

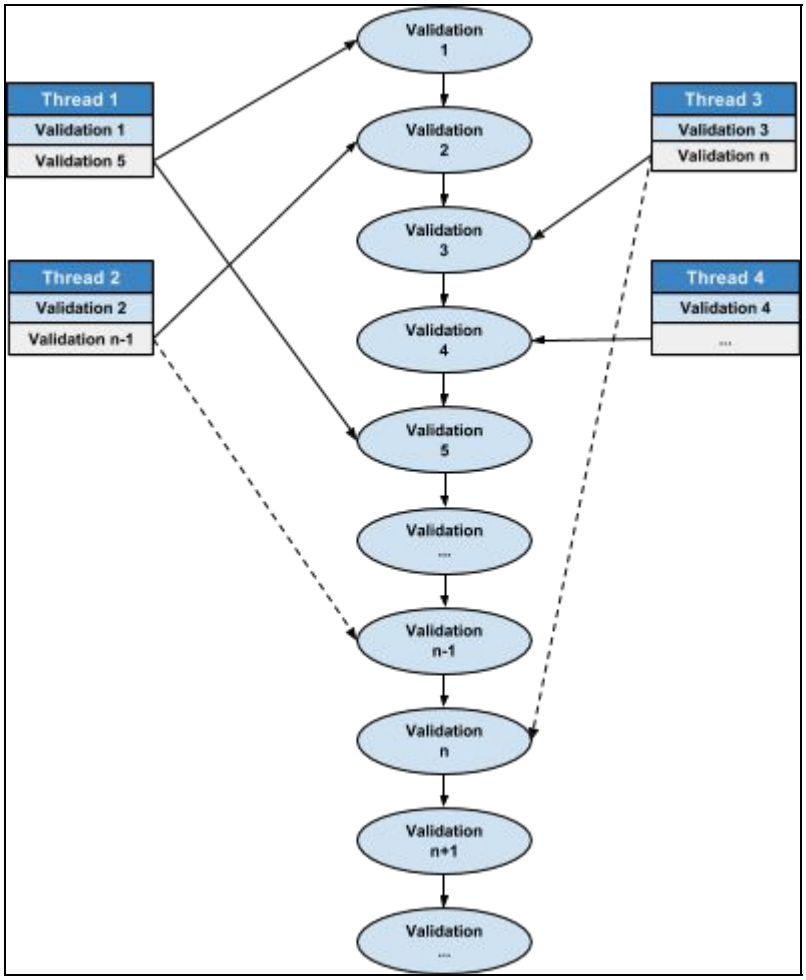
Τη στιγμή που το πρόγραμμα λαμβάνει την εντολή ***flush***, τα validations θα μοιράζονται πλέον στα νήματα και θα αποτιμάται η τιμή τους παράλληλα. Η χρήση των νημάτων μπορεί να γίνει με δύο τρόπους είτε δημιουργώντας καινούργια νήματα για κάθε παράλληλο κομμάτι, είτε με την υλοποίηση ενός job scheduler [Παράρτημα 1]

Διαχωρισμός σε νήματα

Έστω ότι ορίζουμε ***k*** νήματα για την παράλληλη επεξεργασία και πρέπει να υπολογιστούν ***n*** validations , τότε το κάθε νήμα θα πρέπει να επεξεργαστεί κατά μέσο όρο ***n/k*** validations. Ωστόσο επειδή ο αριθμός ***n*** δεν είναι γνωστός με βάση τη δομή λίστας που αναπαριστώνται τα validations, τότε η διαμοίραση μπορεί να πραγματοποιηθεί με τον ακόλουθο τρόπο.

Το 1ο νήμα θα υπολογίζει το validation1, το 2ο νήμα το validation2, το 3ο νήμα το validation3, το 1ο νήμα το validation4, … , το (n mod k) νήμα το (τελευταίο) validationN της τρέχουσας ριπής.

Για να υλοποιηθεί ο παραπάνω τρόπος διαμοιρασμού είναι απαραίτητο το νήμα να λάβει μια δομή που να του περιγράφει τα validations που πρέπει να υπολογίσει. Η δομή αυτή μπορεί να είναι ένας πίνακας δεικτών που θα δείχνει στα validations που θα υπολογίσει το συγκεκριμένο νήμα και θα τον λαμβάνει ως παράμετρο. Ο πίνακας αυτό θα έχει παραμετρικό μέγεθος. Σε περίπτωση που τα validations που πρέπει να εισαχθούν στον πίνακα είναι περισσότερα από το μέγεθος του, τότε θα μεγαλώνει δυναμικά. Το παρακάτω σχήμα συνοψίζει την προσέγγιση αυτή.



Σχήμα 1.

Υπολογισμός validations με χρήση νημάτων

Όταν ολοκληρωθεί ο διαμοιρασμός, τότε το κάθε νήμα / job θα ξεκινήσει να υπολογίζει τη τιμή του κάθε validation που του έχει ανατεθεί και θα αποθηκεύει το αποτέλεσμα (*true, false*) στο ίδιο το validation.

Εξαγωγή αποτελεσμάτων

Αφότου τα νήματα / jobs ολοκληρώσουν την εργασία τους, τότε πρέπει να συγχρονιστούν. Στη συνέχεια το πρόγραμμα θα διατρέξει μια ακόμη φορά τη λίστα των validations μέχρι το σημείο που ορίζει η ***flush***, ώστε να τυπώσει τις τιμές που υπολογίστηκαν καθώς και να απελευθερώσει τους κόμβους αυτούς.

Σημείωση:

Είναι πιθανό η επεξεργασία που απαιτεί μια ριπή να είναι αρκετά σύντομη και άρα η προσθήκη των νημάτων να μην προσδώσει κάποιο ιδιαίτερο όφελος. Για αυτό το λόγο ρυθμίστε την εφαρμογή σας, ώστε να πραγματοποιεί τον υπολογισμό των validations ανά μεταβλητό αριθμό (***rounds***) από εντολές ***flush***. Για παράδειγμα για *round=1*, τότε το πρόγραμμα θα λειτουργεί κανονικά όπως τώρα, για *round=2* θα υπολογίζει τα validations ανά δύο ριπές, για *round=3* θα υπολογίζει τα validations ανά τρεις ριπές, κοκ. Με αυτό το τρόπο θα αυξάνεται η ωφέλιμη εργασία των νημάτων και τελικά η παραλληλία.

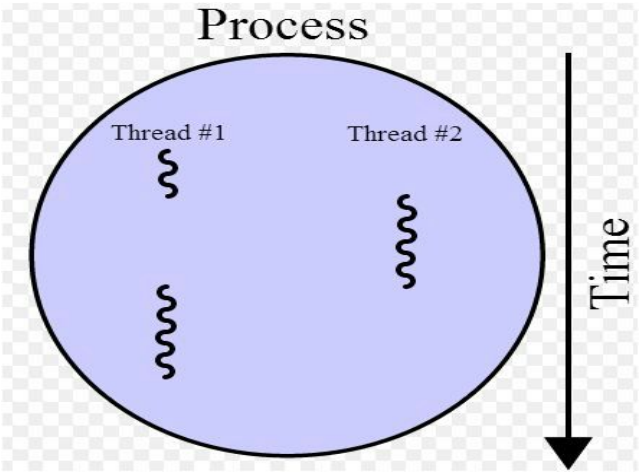
2. Τελική αναφορά εργασίας

Στη τελική αναφορά θα παρουσιάσετε μια σύνοψη ολόκληρη της εφαρμογής που υλοποιήσατε. Μπορείτε να αναφέρετε πράγματα που παρατηρήσατε κατά την μοντελοποίηση / υλοποίηση της εφαρμογής σας, με αποτέλεσμα να σας οδηγήσουν σε συγκεκριμένες σχεδιαστικές επιλογές που βελτίωσαν την εφαρμογή σας σε επίπεδο χρόνου, μνήμης, κτλ.
Στην αναφορά πρέπει ακόμη να παρουσιάσετε ένα σύνολο από πειράματα, τα οποία θα δείχνουν το χρόνο εκτέλεσης για όλες τις επιλογές των δομών που αναπτύξατε στα τρία επίπεδα για τα δύο datasets. Για παράδειγμα μπορείτε να αναφέρετε (π.χ. διαγράμματα) το χρόνο εκτέλεσης, κατανάλωση μνήμης με τις δομές του κάθε επιπέδου ξεχωριστά καθώς και με συνδυασμό τους, όπου φυσικά είναι δυνατό. Τέλος, είναι απαραίτητο να παρουσιάσετε τις δομές που σας έδωσαν τους καλύτερους χρόνους εκτέλεσης για τα δύο datasets και το τελικό χρόνο/μνήμη, μαζί με τις προδιαγραφές του μηχανήματος που τρέξατε τα πειράματα. Η αναφορά δεν πρέπει να ξεπερνά τις 5 σελίδες.

Παράρτημα 1

Πολυνηματισμός

Ένα νήμα είναι μια ελαφριά διεργασία. Η υλοποίηση των νημάτων και των διεργασιών διαφέρει από το ένα λειτουργικό σύστημα στο άλλο. Σε κάθε διεργασία υπάρχει τουλάχιστον ένα νήμα. Αν μέσα σε μια διεργασία υπάρχουν πολλαπλά νήματα, τότε αυτά διαμοιράζονται την ίδια μνήμη και πόρους αρχείων.



Σχήμα 2.

Ένα νήμα διαφέρει από μια διεργασία ενός πολυεπεξεργαστικού λειτουργικού συστήματος στα εξής:

- οι διεργασίες είναι τυπικώς ανεξάρτητες, ενώ τα νήματα αποτελούν υποσύνολα μιας διεργασίας.
- οι διεργασίες περιέχουν σημαντικά περισσότερες πληροφορίες κατάστασης από τα νήματα, ενώ πολλαπλά νήματα μιας διεργασίας μοιράζονται την κατάσταση της διεργασίας, όπως επίσης μνήμη και άλλους πόρους.
- οι διεργασίες έχουν ξεχωριστούς χώρους διευθυνσιοδότησης, ενώ τα νήματα μοιράζονται το σύνολο του χώρου διευθύνσεων που τους παραχωρείται
- η εναλλαγή ανάμεσα στα νήματα μιας διεργασίας είναι πολύ γρηγορότερη από την εναλλαγή ανάμεσα σε διαφορετικές διεργασίες.

Η πολυνημάτωση αποτελεί ένα ευρέως διαδεδομένο μοντέλο προγραμματισμού και εκτέλεσης διεργασιών το οποίο επιτρέπει την ύπαρξη πολλών νημάτων μέσα στα πλαίσια μιας και μόνο διεργασίας. Τα νήματα αυτά μοιράζονται τους πόρους της διεργασίας και μπορούν να εκτελούνται ανεξάρτητα. Το γεγονός ότι επιτρέπει μια διεργασία να εκτελείται παράλληλα σε ένα σύστημα με πολλαπλούς πυρήνες αποτελεί ίσως την πιο ενδιαφέρουσα εφαρμογή της συγκεκριμένης τεχνολογίας.

Το πλεονέκτημα ενός πολυνηματικού προγράμματος είναι ότι του επιτρέπει να εκτελείται γρηγορότερα σε υπολογιστικά συστήματα που έχουν πολλούς επεξεργαστές, επεξεργαστές με πολλούς πυρήνες, ή κατά μήκος μιας συστοιχίας υπολογιστών. Για να μπορούν να χειραγωγηθούν σωστά τα δεδομένα, τα νήματα θα πρέπει ορισμένες φορές να συγκεντρωθούν σε ένα ορισμένο χρονικό διάστημα έτσι ώστε να επεξεργασθούν τα δεδομένα στη σωστή σειρά.

Ένα ακόμα πλεονέκτημα της πολυδιεργασίας (και κατ' επέκταση της πολυνημάτωσης) ακόμα και στους απλούς επεξεργαστές (με έναν πυρήνα επεξεργασίας) είναι η δυνατότητα για μια εφαρμογή να ανταποκρίνεται άμεσα. Έχοντας ένα πρόγραμμα που εκτελείται μόνο σε ένα νήμα, αυτό θα φαίνεται ότι κολλάει ή ότι παγώνει, στις περιπτώσεις που εκτελείται κάποια μεγάλη διεργασία που απαιτεί πολύ χρόνο. Αντίθετα σε ένα πολυνηματικό σύστημα, οι χρονοβόρες διεργασίες μπορούν να εκτελούνται παράλληλα με άλλα νήματα που φέρουν άλλες εντολές για το ίδιο πρόγραμμα, καθιστώντας το άμεσα ανταποκρίσιμο.

Για την υλοποίηση των πολυνηματικών λειτουργιών θα χρησιμοποιήσετε τη Pthreads που είναι μια POSIX API C βιβλιοθήκη. Για τη χρήση της πρέπει να συμπεριλάβετε στα αρχεία του κώδικα σας το <pthread.h> και να χρησιμοποιήσετε κατα το compilation το -pthread flag.

Οι βασικές ρουτίνες των PThreads είναι οι εξής:

- int **pthread_create**(pthread_t *thread, const pthread_attr_t *attr, void*(*start_routine)(void), void *arg)
- void **pthread_exit**(void *value_ptr)
- int **pthread_join**(pthread_t thread, void **value_ptr)

Ο στόχος της ρουτίνας ***pthread_create*** είναι να δημιουργήσει ένα νέο νήμα και αφού αρχικοποιήσει τα χαρακτηριστικά αυτού, το κάνει διαθέσιμο για χρήση. Στην μεταβλητή *thread* επιστρέφετε το αναγνωριστικό του νήματος που μόλις δημιουργήθηκε. Με βάση τη τιμή στο πεδίο *attr* θα αρχικοποιηθούν τα χαρακτηριστικά του νέου νήματος. Στο όρισμα *start_routine* ορίζεται η ρουτίνα που θα εκτελέσει το νέο thread όταν δημιουργηθεί και στο πεδίο *arg* θα ορισθούν οι παράμετροι αυτής.

Η ρουτίνα ***pthread_exit*** θα τερματίσει ένα ήδη υπάρχον νήμα και θα αποθηκεύσει την κατάσταση τερματισμού για όσα άλλα νήματα θα προσπαθήσουν να συνενωθούν με αυτό. Επιπρόσθετα ελευθερώνει όλα τα δεδομένα του νήματος, συμπεριλαμβανομένων και της στοίβα του νήματος. Είναι σημαντικό τα αντικείμενα συγχρονισμού νημάτων ,όπως τα mutexes και οι μεταβλητές κατάστασης (condition variables), που κατανέμονται στη stack του νήματος, να καταστραφούν πριν κλήση της ρουτίνας pthread_exit.

Η ρουτίνα ***pthread_join*** μπλοκάρει το τρέχον thread μέχρι να τερματίσει το thread που προσδιορίζεται από το πεδίο *thread*. Η κατάσταση τερματισμού του thread αυτού επιστρέφεται στο πεδίο *value_ptr*. Αν το συγκεκριμένο thread έχει ήδη τερματίσει (και δεν είχε προηγουμένως αποσπασθεί), το τρέχον thread δεν θα μπλοκαριστεί.

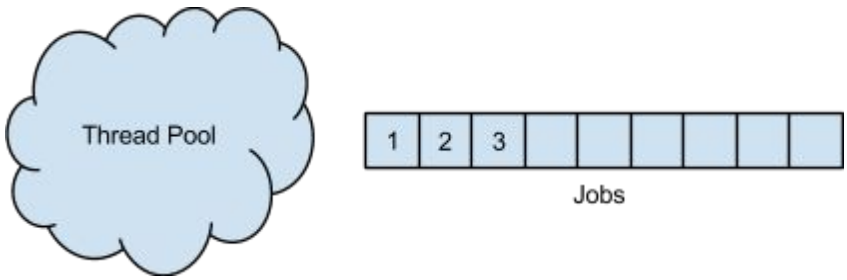
Job Scheduler

Εργασία (Jobs)

Εργασία (Job) είναι μια ρουτίνα κώδικα η οποία θέλουμε να εκτελεστεί από κάποιο thread πιθανότατα παράλληλα με κάποια άλλη. Μπορούμε να ορίσουμε οτιδήποτε θέλουμε ως job και να το αναθέσουμε στον χρονοπρογραμματιστή.

Χρονοπρογραμματιστής Εργασιών (Scheduler) και Δεξαμενή Νημάτων (Thread Pool)

Ο χρονοπρογραμματιστής ουσιαστικά δέχεται δουλειές και αναλαμβάνει την ανάθεση τους σε νήματα, για προσωρινή αποθήκευση των εργασιών χρησιμοποιεί μια ουρά προτεραιότητας. Έστω ότι έχουμε μια Δεξαμενή νημάτων (thread pool) και μια συνεχόμενη ροή από ανεξάρτητες εργασίες (jobs). Όταν δημιουργείται μια εργασία, μπαίνει στην ουρά προτεραιότητας του χρονοπρογραμματιστή και περιμένει να εκτελεστεί. Οι εργασίες εκτελούνται με την σειρά που δημιουργήθηκαν (first-in-first-out - FIFO). Κάθε νήμα περιμένει στην ουρά μέχρι να του ανατεθεί μια εργασία και, αφού την εκτελέσει, επιστρέφει στην ουρά για να αναλάβει νέα εργασία. Για την ορθή λειτουργία ενός χρονοπρογραμματιστή είναι απαραίτητη η χρήση σημαφόρων στην ουρά, ώστε να μπλοκάρουν εκεί τα νήματα, και κάποια κρίσιμη περιοχή (critical section), ώστε να γίνεται σωστά εισαγωγή και απολαβή εργασιών από την ουρά.



Σχήμα 3.

Condition Variables

΄Ενα ακόμα εργαλείο συγχρονισμού που μπορεί να σας φανεί χρήσιμο είναι τα condition variables. Ένα condition variable είναι ένα εργαλείο που επιτρέπει στα POSIX threads να αναβάλουν την εκτέλεση τους μέχρι μια έκφραση να γίνει αληθής. Δύο είναι οι βασικές πράξεις πάνω σε ένα condition variable, wait που αδρανοποιεί το thread που την κάλεσε και η signal που ξυπνά ένα από τα thread που είναι απενεργοποιημένα πάνω στο ίδιο condition variable. Ένα condition variable χρησιμοποιείται ζευγάρι με ένα mutex.

Οι βασικές ρουτίνες των POSIX condition variable είναι οι εξής:

- int **pthread_cond_init**(pthread cond)t *cond, pthread_condattr_t *cond attr)
- int **pthread_cond_wait**(pthread_cond_t *cond, pthread_mutex_t *mutex)
- int **pthread_cond_signal**(pthread_cond_t *cond)
- int **pthread_cond_broadcast**(pthread_cond_t *cond)
- int **pthread_cond_destroy**(pthread_cond_t *cond)

Οι condition variables χρησιμοποιούνται μέσα σε while loops:

```
mutex_lock(mtx1)
while( canRun ){
        cond_var_wait(condvar1, mtx1)
}
//do some work
cond_var_signal(condvar1)
mutex_unlock(mtx1)
```

Πριν τη χρήση του condvar1 κλειδώνουμε το mtx1. Μετά ελέγχουμε σε μια while μια συνθήκη που αν αποτιμηθεί ως αληθής σημαίνει ότι το thread δεν μπορεί να κάνει τη δουλειά του και πρέπει να αδρανοποιηθεί με τη χρήση της cond_var_wait. Αλλιώς το thread δεν εισέρχεται στο εσωτερικό της while και εκτελεί τη δουλειά του. Χρησιμοποιεί τη signal για να ξυπνήσει ένα ακόμα thread, αν υπάρχει, που είναι αδρανοποιημένο στο ίδιο condition variable και ξεκλειδώνει το mtx1.