1η Εργασία - Τεχνολογίες Ανάλυσης Μεγάλων Δεδομένων

Κωνσταντίνα Μαρίνα Μπλέτσα, ΑΕΜ 243

Αναφορά – Επεξήγηση Κώδικα

Υποεργασία 1 (Αρχείο κώδικα MultiplicationThreads(Y1))

Το πρόγραμμα στην υποεργασία 1 υλοποιεί τον πολλαπλασιασμό ενός πίνακα με ένα διάνυσμα χρησιμοποιώντας πολυνηματισμό (multithreading) στη Java. Σκοπός της είναι να συγκρίνουμε τον χρόνο εκτέλεσης για διαφορετικό αριθμό νημάτων (threads) και έτσι να φανεί η επίδραση του ταυτοχρονισμού στην απόδοση του κώδικα. Στον κώδικα αρχικά όρισα το μέγεθος του πίνακα A (N*M) και του διανύσματος v (μεγέθους M) όπου τα N και M τα βάζω ότι μέγεθος θέλω πχ 200 και 100. Στην main() γεμίζω τον πίνακα και το διάνυσμα με τυχαίους αριθμούς από το 0 έως το 10 έπειτα για κάθε αριθμό νημάτων (1, 2, 4, 8): Εκτυπώνω πόσα νηματα θα χρησιμοποιηθούν θα χρησιμοποιηθούν, Καταγράφω την ώρα έναρξης με System.nanoTime() και την ώρα λήξης, Καλώ τη μέθοδο *multiplyWithExecutor*(numThreads) για να εκτελέσει τον πολλαπλασιασμό, Υπολογίζω τον χρόνο εκτέλεσης σε χιλιοστά του δευτερολέπτου (ms) δηλαδή τα μετατρέπω από τα nanoseconds και εκτυπώνω το αποτέλεσμα και τον χρόνο. Στην multiplyWithExecutor() δημιουργώ ένα σύνολο “pool” από numThreads νήματα, αν π.χ. numThreads = 4, τότε φτιάχνει 4 threads που είναι έτοιμα να εκτελούν εργασίες αυτά τα threads δημιουργούνται μία φορά και μετά ξαναχρησιμοποιούνται**,** το αντικείμενο executor είναι υπεύθυνο να αναθέτει εργασίες στα threads**.** Ο πίνακας A χωρίζεται σε υποσύνολα γραμμών και κάθε νήμα επεξεργάζεται ένα τμήμα του πίνακα, καθοριζόμενο από τα όρια, start: πρώτη γραμμή που θα υπολογίσει το νήμα και end: τελευταία γραμμή που θα υπολογίσει. Μετά κάνω τις πράξεις του πολλαπλασιασμού για κάθε γραμμή i ξεκινάει με sum = 0, πολλαπλασιάζει κάθε στοιχείο της γραμμής A[i][j] με το στοιχείο v[j] του διανύσματος, προσθέτει όλα τα γινόμενα και αποθηκεύει το αποτέλεσμα στο result[i]. Τέλος εκτυπώνω το αποτέλεσμα διάνυσμα του πολλαπλασιασμού.

Υποεργασία 2 (Αρχείο κώδικα HospitalManagment(Y2))

Για την υλοποίηση του κώδικα της 2ης υποεργασίας χρησιμοποιήθηκαν:

Hospital, το αντικείμενο που κρατά την κατάσταση των κλινών (bedsOccupied) και στατιστικά (totalRecovered, totalTurnedAway). Οι μέθοδοι admitNewCases(int n) και releasePatients(int n) είναι συγχρονισμένες (synchronized) ώστε να εξασφαλίζουν αμοιβαίο αποκλεισμό κατά την πρόσβαση και τροποποίηση της κοινής κατάστασης τους. Το DiseaseThread: κάθε χ δευτερόλεπτα παράγει τυχαίο

πλήθος νέων κρουσμάτων (0..K) και καλεί hospital.admitNewCases(). Το HospitalThread: κάθε y δευτερόλεπτα θεραπεύει τυχαίο αριθμό (0..H) και καλεί hospital.releasePatients(...) για να απελευθερώσει τις αντίστοιχες κλίνες. Η κύρια συνάρτηση (main) εκκινεί τα δύο νήματα και τα περιμένει (join) να τερματίσουν, τέλος εκτυπώνει την τελική αναφορά (πόσοι ανάρρωσαν, πόσοι δεν βρήκαν θέση και πόσες κλίνες χρησιμοποιούνται). Για να πετύχει ο συγχρονισμός το αντικείμενο Hospital χρησιμοποιεί synchronized (this) για να προστατεύσει τις κρίσιμες περιοχές του (τα οποία είναι ο έλεγχος/ενημέρωση bedsOccupied και ενημέρωση στατιστικών). Αυτό αποτρέπει τα race conditions όταν και τα δύο νήματα προσπαθούν ταυτόχρονα να αλλάξουν την κατάσταση. Οι μετρήσεις totalRecovered και totalTurnedAway είναι AtomicInteger για thread-safe ατομικές προσθαφαιρέσεις (κάτι που είναι προαιρετικό διότι οι κλήσεις γίνονται μέσα σε synchronized μπλοκ). Εκτός από την τελική αναφορά για κάθε επανάληψη ο κώδικας εκτυπώνει το πόσοι εισήχθησαν, απελευθερώθηκαν και απέτυχαν να εισαχθούν, και πόσες κλίνες χρησιμοποιούνται. Όλες οι βασικές παράμετροι (ITERATIONS, ICU_CAPACITY, K, H, διαστήματα) είναι static final στην αρχή του αρχείου για να ρυθμίζονται εύκολα (έκανα πειραματισμό με διάφορες τιμές).

Μετατροπή της λύσης για 3 νοσοκομεία που μοιράζονται την ίδια ουρά κρουσμάτων (περιγραφή προσέγγισης)

Στην νέα εκδοχή του κώδικα θα υπάρχει μία κοινή ουρά (queue) των νέων περιστατικών που απαιτούν νοσηλεία και για τα τρία νοσοκομεία. Τα τρία νοσοκομεία θα είναι ανεξάρτητα (νήματα νοσοκομείων αναπαράσταση με threads) και θα παίρνουν ασθενείς από την ίδια ουρά. Κάθε νοσοκομείο έχει τη δική του χωρητικότητα (π.χ. capacityPerHospital) και μπορεί να δεχθεί ασθενείς όσο έχει διαθέσιμες κλίνες. Οι ασθενείς περιμένουν στην ουρά (σε σειρά προτεραιότητας) έως ότου κάποιο νοσοκομείο τους πάρει και, εάν η ουρά γίνει πολύ μεγάλη ή ξεπεραστεί κάποιος χρόνος αναμονής μπορούμε να τους σταματάμε από την αναμονή και να τους παραπέμπουμε ας πούμε σε άλλο νοσοκομείο. Για τον συγχρονισμό η ουρά που θα χρησιμοποιούσα είναι η BlockingQueue η οποία προσφέρει ασφάλεια (thread-safe).

Υποεργασία 3 (Αρχεία κώδικα Server(Y3) και Client(Y3))

Η υποεργασία έχει δύο αρχεία το Server.java και το Client.java τα οποία έχουν επικοινωνία μεταξύ τους. Στον server δημιουργούμε έναν πίνακα κατακερματισμού Hashtable<Integer, Integer> για να αποθηκεύει τα ζεύγη (κλειδί, τιμή). Ανοίγω ένα ServerSocket στη δοσμένη πόρτα (8888) και περιμένει τον client, για κάθε αίτημα του client, διαβάζει την τριάδα (A, B, C) όπου αν A = 0 → τερματισμός επικοινωνίας,

αν A = 1 → εισαγωγή (B,C) στο hashtable, αν A = 2 → διαγραφή του B και αν A = 3 → αναζήτηση του B στον πίνακα κατακερματισμού. Ο server έπειτα επιστρέφει απάντηση (1 ή 0 ή την τιμή) πίσω στον client. Ο client συνδέεται στον server με Socket(host, 8888), διαβάζει από το πληκτρολόγιο τις εντολές (A, B, C) και τις στέλνει στον server με DataOutputStream.writeInt() ενώ λαμβάνει απάντηση με DataInputStream.readInt().

Για να τρέξει το πρόγραμμα (στο Intellij) και να γίνει η σύνδεση Server-Client αρχικά τρέχω τον Server πατάω το Run → Edit Configurations και στο πεδίο Program arguments γράφω 8888, έπειτα μου εμφανίζει το μήνυμα «Ο server ξεκίνησε στην πόρτα: 8888 Αναμονή για σύνδεση...» και τώρα για να συνδεθεί ο client πατάω στο αρχείο Client.java Run → Edit Configurations και στο πεδίο Program arguments γράφω localhost 8888. Τώρα αφού τρέχουν και τα δύο και έχουν συνδεθεί μου εμφανίζεται μήνυμα στην κονσόλα «Συνδέθηκε με τον server localhost:8888 Δώσε εντολή (A B C). A=0(exit),1(insert),2(delete),3(search):» και δίνουμε όποια εντολή θέλουμε.

Μετατροπή της παραπάνω λύσης για έναν Server που θα εξυπηρετεί πολλούς clients ταυτόχρονα

Για να μπορεί ο server να εξυπηρετεί πολλούς clients ταυτόχρονα, θα πρέπει να μετατραπεί σε πολυνηματικό, δηλαδή να δημιουργεί ένα ξεχωριστό νήμα (thread) για κάθε client που συνδέεται. Κάθε νήμα θα λειτουργεί ανεξάρτητα και θα χειρίζεται την επικοινωνία με τον συγκεκριμένο client, θα διαβάζει τις εντολές του και θα εκτελεί τις αντίστοιχες λειτουργίες σε έναν κοινό πίνακα κατακερματισμού. Ο πίνακας αυτός είναι κοινός για όλα τα νήματα, ώστε οι αλλαγές που κάνει ένας client (όπως εισαγωγή, διαγραφή ή αναζήτηση στοιχείου) να είναι ορατές και στους υπόλοιπους (η κλάση Hashtable της Java είναι ασφαλής για χρήση από πολλά νήματα οπότε δεν υπάρχει πρόβλημα ταυτοχρονισμού). Με αυτήν την προσέγγιση, ο server μπορεί να εξυπηρετεί πολλούς clients ταυτόχρονα και να διατηρεί την απόδοσή του.

Υποεργασία 4 (Αρχεία κώδικα Server_CP(Y4), Producer(Y4) και Consumer(Y4))

Το πρόγραμμα της υποεργασίας 4 υλοποιεί ένα σύστημα επικοινωνίας μεταξύ διεργασιών (interprocess communication) με χρήση TCP sockets. Υπάρχουν τρεις τύποι διεργασιών server, producer, consumer. Ο server δέχεται συνδέσεις και από producers και από consumers, χρησιμοποιώντας δύο διαφορετικές θύρες τις 8881, 8882 και 8883 για producers και τις 9991, 9992, 9993 για consumers. Η κλάση Server_CP δημιουργεί για κάθε instance δύο ServerSocket (μία θύρα για producers και μία για consumers) και διατηρεί μια ακέραια μεταβλητή storage που αντιπροσωπεύει το απόθεμα (τύπου int, αρχικοποιείται τυχαία 1–1000). Για κάθε εισερχόμενη σύνδεση ο server δημιουργεί έναν handler (ProducerHandler ή ConsumerHandler) που υλοποιεί Runnable και χειρίζεται την εισερχόμενη τιμή μέσω BufferedReader, PrintWriter (I/O streams του socket) ενώ οι handlers τρέχουν σε

ExecutorService (cached thread pool) ώστε να υποστηρίζονται πολλαπλές ταυτόχρονες συνδέσεις του πολυνηματισμού. Ο ταυτοχρονισμός με ασφάλεια της κοινής μεταβλητής storage γίνεται με synchronized(lock) πάνω σε ένα final Object lock. Οι producers προσπαθούν να προσθέσουν τυχαία τιμή 10–100 και η προσθήκη απορρίπτεται αν το νέο απόθεμα θα ξεπεράσει 1000, ενώ οι consumers αφαιρούν τυχαία 10–100 και η αφαίρεση απορρίπτεται αν το απόθεμα πέσει κάτω από 1, σε κάθε περίπτωση αποστέλλεται απάντηση στον χρήστη που τρέχει το πρόγραμμα. Οι κλάσεις Producer και Consumer (οι οποίες δεν είναι public) τρέχουν σε βρόχο επιλέγουν τυχαία server, ανοίγουν Socket, στέλνουν την τιμή και διαβάζουν την απάντηση και τέλος πηγαίνουν στην κατάσταση Thread.sleep για τυχαίο διάστημα 1–10 δευτερολέπτων.

Τα αρχεία για αυτήν την υποεργασία είναι το Server_CP, το Consumer και το Producer και για να τρέξουν στο Intellij τα βάζω αυτά τα τρία σε ένα project και πατάω run στο Server_CP, στο Consumer και στο Producer και βλέπω τα αποτελέσματα (δηλαδή τις προσθαφαιρέσεις στο απόθεμα) στην κονσόλα.