**Improving the data access using threading**

**Performance without using threading:**

1. Database Insertion:

- Takes approximately 7.33 minutes.

2. Database Retrieval:

- Takes around 1.85 minutes.

3. L1 Cache Insertion and Retrieval:

- Insertion: 0.0018 minutes.

- Retrieval: 0.00063 minutes.

4. L2 Cache Insertion and Retrieval:

- Insertion: 0.0118 minutes.

- Retrieval: 0.006 minutes.

5. Multilevel Cache Retrieval:

- Takes around 0.93 minutes.

**Performance using threading:**

1. Database Operations:

- Insertion Time: Approximately 0.025 minutes (1.5 seconds).

- Retrieval Time: Approximately 0.0031 minutes (0.19 seconds).

2. L1 Cache Operations:

- Insertion Time: Approximately 0.0000885 minutes (5 milliseconds).

- Retrieval Time: Approximately 0.0000283 minutes (1.7 milliseconds).

3. L2 Cache Operations:

- Insertion Time: Approximately 0.0000694 minutes (4 milliseconds). - Retrieval Time: Approximately 0.0000165 minutes (0.99 milliseconds).

**Performance difference:**

* Database insertion using threading is 79.5% faster than the insertion without using threading.
* Database retrieval using threading is 99.83% faster than the retrieval without using threading.
* L1 cache insert using threading is 95.1% faster than the insertion without using threading.
* L1 cache retrieval using threading is 95.56% faster than the retrieval without using threading.
* L2 cache insert using threading is 99.41% faster than the insertion without using threading.
* L2 cache retrieval using threading is 99.73% faster\*\* than the retrieval without using threading.

**CODE:**

package com.example;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

import java.util.Map;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

import java.util.concurrent.TimeUnit;

import com.google.common.cache.Cache;

import com.google.common.cache.CacheBuilder;

import java.util.LinkedHashMap;

public class AdvancedDatabaseCachingBenchmark {

private static final String DB\_URL = "jdbc:mysql://localhost:3306/testdb";

private static final String DB\_USER = "root";

private static final String DB\_PASSWORD = "root";

private static final int NUM\_ELEMENTS = 100000;

private static final int L1\_CACHE\_SIZE = 10000;

private static final int L2\_CACHE\_SIZE = 10000;

private static final int L2\_CACHE\_DURATION\_MINUTES = 10;

private static Connection connection;

private static Map<Integer, String> l1Cache;

private static Cache<Integer, String> l2Cache;

private static final ExecutorService executorService = Executors.newFixedThreadPool(10);

public static void main(String[] args) {

try {

setupDatabase();

setupCaches();

long dbInsertTime = benchmarkDatabaseInsert();

long dbRetrieveTime = benchMarkDatabaseRetrieve();

long l1CacheInsertTime = benchMarkL1CacheInsert();

long l1CacheRetrieveTime = benchMarkL1CacheRetrieve();

long l2CacheInsertTime = benchMarkL2CacheInsert();

long l2CacheRetrieveTime = benchMarkL2CacheRetrieve();

long multilevelCacheRetrieveTime = benchmarkMultilevelCacheRetrieve();

printResults(dbInsertTime, dbRetrieveTime, l1CacheInsertTime, l1CacheRetrieveTime, l2CacheInsertTime, l2CacheRetrieveTime, multilevelCacheRetrieveTime);

} catch (Exception e) {

e.printStackTrace();

} finally {

try {

connection.close();

executorService.shutdown();

} catch (SQLException e) {

e.printStackTrace();

}

}

}

private static void setupDatabase() throws SQLException {

connection = DriverManager.getConnection(DB\_URL, DB\_USER, DB\_PASSWORD);

try (Statement statement = connection.createStatement()) {

statement.executeUpdate("CREATE TABLE IF NOT EXISTS test\_table (id INT PRIMARY KEY, value VARCHAR(255))");

}

}

private static void setupCaches() {

l1Cache = new LinkedHashMap<Integer, String>(L1\_CACHE\_SIZE, 0.75f, true) {

@Override

protected boolean removeEldestEntry(Map.Entry<Integer, String> eldest) {

return size() > L1\_CACHE\_SIZE;

}

};

l2Cache = CacheBuilder.newBuilder()

.maximumSize(L2\_CACHE\_SIZE)

.expireAfterAccess(L2\_CACHE\_DURATION\_MINUTES, TimeUnit.MINUTES)

.build();

}

private static long benchmarkDatabaseInsert() throws SQLException {

long startTime = System.nanoTime();

String sql = "INSERT INTO test\_table (id, value) VALUES (?, ?)";

try (PreparedStatement statement = connection.prepareStatement(sql)) {

for (int i = 0; i < NUM\_ELEMENTS; i++) {

statement.setInt(1, i);

statement.setString(2, "Value" + i);

statement.addBatch();

if (i % 1000 == 0) {

statement.executeBatch();

}

}

statement.executeBatch();

}

long endTime = System.nanoTime();

return endTime - startTime;

}

private static long benchMarkDatabaseRetrieve() throws SQLException {

long startTime = System.nanoTime();

String sql = "SELECT \* FROM test\_table WHERE id = ?";

try (PreparedStatement statement = connection.prepareStatement(sql)) {

for (int i = 0; i < NUM\_ELEMENTS; i++) {

statement.setInt(1, i);

try (ResultSet resultSet = statement.executeQuery()) {

if (resultSet.next()) {

resultSet.getString("value");

}

}

}

}

long endTime = System.nanoTime();

return endTime - startTime;

}

private static long benchMarkL1CacheInsert() {

long startTime = System.nanoTime();

for (int i = 0; i < NUM\_ELEMENTS; i++) {

l1Cache.put(i, "Value" + i);

}

long endTime = System.nanoTime();

return endTime - startTime;

}

private static long benchMarkL1CacheRetrieve() {

long startTime = System.nanoTime();

for (int i = 0; i < NUM\_ELEMENTS; i++) {

l1Cache.get(i);

}

long endTime = System.nanoTime();

return endTime - startTime;

}

private static long benchMarkL2CacheInsert() {

long startTime = System.nanoTime();

for (int i = 0; i < NUM\_ELEMENTS; i++) {

l2Cache.put(i, "Value" + i);

}

long endTime = System.nanoTime();

return endTime - startTime;

}

private static long benchMarkL2CacheRetrieve() {

long startTime = System.nanoTime();

for (int i = 0; i < NUM\_ELEMENTS; i++) {

l2Cache.getIfPresent(i);

}

long endTime = System.nanoTime();

return endTime - startTime;

}

private static long benchmarkMultilevelCacheRetrieve() throws SQLException {

long startTime = System.nanoTime();

for (int i = 0; i < NUM\_ELEMENTS; i++) {

String value = l1Cache.computeIfAbsent(i, k -> l2Cache.getIfPresent(k));

if (value == null) {

String sql = "SELECT \* FROM test\_table WHERE id = ?";

try (PreparedStatement statement = connection.prepareStatement(sql)) {

statement.setInt(1, i);

try (ResultSet resultSet = statement.executeQuery()) {

if (resultSet.next()) {

value = resultSet.getString("value");

l2Cache.put(i, value);

}

}

}

}

}

long endTime = System.nanoTime();

return endTime - startTime;

}

private static void printResults(long dbInsertTime, long dbRetrieveTime, long l1CacheInsertTime, long l1CacheRetrieveTime, long l2CacheInsertTime, long l2CacheRetrieveTime, long multilevelCacheRetrieveTime) {

System.out.println("Database Insert Time: " + dbInsertTime + " nanoseconds");

System.out.println("Database Retrieve Time: " + dbRetrieveTime + " nanoseconds");

System.out.println("L1 Cache Insert Time: " + l1CacheInsertTime + " nanoseconds");

System.out.println("L1 Cache Retrieve Time: " + l1CacheRetrieveTime + " nanoseconds");

System.out.println("L2 Cache Insert Time: " + l2CacheInsertTime + " nanoseconds");

System.out.println("L2 Cache Retrieve Time: " + l2CacheRetrieveTime + " nanoseconds");

System.out.println("Multilevel Cache Retrieve Time: " + multilevelCacheRetrieveTime + " nanoseconds");

}

}