Міністерство освіти і науки України
НТУУ «Київський політехнічний інститут»
Фізико-технічний інститут

Проектування високонавантажених систем
Лабораторна робота No6
Кешування з використанням Hazelcast

Виконав:
Студент 4-го курсу
групи ФІ-21
Климентьєв Максим
Перевірив:
_____

Зміст

1 Код реалізації

Counter.py

```
import hazelcast
import time
import threading
import psycopg2


class Counter:
    def __init__(self, database_params, info: bool = False):
        if info:
            print("[INFO] Connecting to Hazelcast...")

        self.database_params = database_params
        self.info = info

        self.client = hazelcast.HazelcastClient(
            cluster_name="dev",
            cluster_members=["localhost:5701"]
        )
        self.counters = self.client.get_map("counters").blocking()
        self._clear_database()
        self.get_info()

    def _get_connection(self):
        return psycopg2.connect(**self.database_params)

    def _clear_database(self):
        if self.info:
            print("[INFO] Clearing Counters...")

        self.counters.clear()
        for i in range(1, 5):
            self.counters.put(i, 0)
        time.sleep(1)

    def close(self):
        self.client.shutdown()

    def get_info(self):
        connection = self._get_connection()
        cursor = connection.cursor()
        cursor.execute("SELECT id, value FROM counters ORDER BY id")
        data = cursor.fetchall()
        cursor.close()
        connection.close()

        if self.info:
            print(f"[INFO] Data: {data}")
        return data

    def client_task(self, counter_id, requests_count):
        for _ in range(requests_count):
            self.counters.lock(counter_id)
            try:
                value = self.counters.get(counter_id)
                self.counters.put(counter_id, value + 1)
            finally:
                self.counters.unlock(counter_id)

    def run(self, name, counter_id, clients, increments_per_client):
        if self.info:
            print(f"[INFO] Running {name}")
            print(f"[INFO] Counter ID: {counter_id}")
            print(f"[INFO] Clients: {clients} | Increments per client: {increments_per_client}")

        expected = clients * increments_per_client

        threads = []
        start_time = time.time()
        for _ in range(clients):
            t = threading.Thread(target=self.client_task, args=(counter_id, increments_per_client))
            threads.append(t)
            t.start()

        for t in threads:
```

```
            t.join()

        duration = time.time() - start_time
        throughput = expected / duration

        result = self.counters.get(counter_id)
        if self.info:
            print(f"[RESULT] Time: {duration:.2f} sec")
            print(f"[RESULT] Throughput: {throughput:.2f} increments/sec")
            print(f"[RESULT] Received: {result:_} ({result/expected*100:.2f}% from Expected: {expected:_})")


if __name__ == "__main__":
    database_params = {
        "host": "localhost",
        "port": 5432,
        "dbname": "counter_db",
        "user": "postgres",
        "password": "postgres"
    }

    counter = Counter(database_params, info=True)
    counter.run("Counter 1", 1, 1, 10000)
    counter.run("Counter 2", 2, 2, 10000)
    counter.run("Counter 3", 3, 5, 10000)
    counter.run("Counter 4", 4, 10, 10000)
    counter.get_info()
    counter.close()
```

CounterMapStore.java

```
import com.hazelcast.map.MapStore;
import java.sql.*;
import java.util.*;


public class CounterMapStore implements MapStore<Integer, Integer>
{
    private Connection connection;

    public CounterMapStore()
    {
        String url = "jdbc:postgresql://postgres:5432/counter_db";
        String user = "postgres";
        String password = "postgres";

        for (int i = 0; i < 20; i++)
            {
            try
            {
                connection = DriverManager.getConnection(url, user, password);

                System.out.println("[INFO] Connected to PostgreSQL");

                Statement statement = connection.createStatement();
                statement.executeUpdate("CREATE TABLE IF NOT EXISTS counters (id INT PRIMARY KEY, value INT)");
                return;
            }
            catch (SQLException e)
            {
                System.out.println(">>> Waiting for PostgreSQL... Attempt " + (i + 1));
                try
                {
                    Thread.sleep(2000);
                }
                catch (InterruptedException ex)
                {
                    ex.printStackTrace();
                }
            }
        }

        throw new RuntimeException("Could not connect to PostgreSQL after multiple attempts");
    }

    @Override
    public synchronized void store(Integer key, Integer value)
    {
        try
        {
```

```
            PreparedStatement statement = connection.prepareStatement("INSERT INTO counters (id, value) VALUES (?, ?)
     ON CONFLICT (id) DO UPDATE SET value = EXCLUDED.value");
            statement.setInt(1, key);
            statement.setInt(2, value);
            statement.executeUpdate();
        }
        catch (SQLException e)
        {
            e.printStackTrace();
        }
    }

    @Override
    public synchronized void storeAll(Map<Integer, Integer> map)
    {
        for (Map.Entry<Integer, Integer> entry : map.entrySet())
        {
            store(entry.getKey(), entry.getValue());
        }
    }

    @Override
    public synchronized void delete(Integer key)
    {
        try
        {
            PreparedStatement statement = connection.prepareStatement("DELETE FROM counters WHERE id = ?");
            statement.setInt(1, key);
            statement.executeUpdate();
        }
        catch (SQLException e)
        {
            e.printStackTrace();
        }
    }

    @Override
    public synchronized void deleteAll(Collection<Integer> keys)
    {
        for (Integer key : keys) delete(key);
    }

    // Read-through
    @Override
    public synchronized Integer load(Integer key)
    {
        try
        {
            PreparedStatement statement = connection.prepareStatement("SELECT value FROM counters WHERE id = ?");
            statement.setInt(1, key);
            ResultSet rs = statement.executeQuery();
            if (rs.next())
            {
                return rs.getInt("value");
            }
        }
        catch (SQLException e)
        {
            e.printStackTrace();
        }
        return null;
    }

    @Override
    public synchronized Map<Integer, Integer> loadAll(Collection<Integer> keys)
    {
        Map<Integer, Integer> result = new HashMap<>();
        for (Integer key : keys)
        {
            Integer value = load(key);
            if (value != null) result.put(key, value);
        }
        return result;
    }

    @Override
    public Iterable<Integer> loadAllKeys()
    {
        return null;
    }
}
```

docker-compose.yaml

```yaml
services:
  postgres:
    image: postgres:18.1
    container_name: postgres
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
      POSTGRES_DB: counter_db
    volumes:
      - postgre-data:/var/lib/postgresql
    ports:
      - "5432:5432"
    healthcheck:
      test: ["CMD", "pg_isready", "-U", "postgres"]
      interval: 0.5s
      retries: 30

  hazelcast:
    build: .
    container_name: hazelcast
    ports:
      - "5701:5701"
    environment:
      - HZ_NETWORK_PUBLICADDRESS=localhost:5701
      - JAVA_OPTS=-Dhazelcast.config=/opt/hazelcast/config/hazelcast.
   yaml
    depends_on:
      postgres:
        condition: service_healthy

volumes:
  postgre-data:
```

Dockerfile

```
FROM ubuntu:22.04 AS builder

RUN apt-get update && apt-get install -y openjdk-17-jdk wget
RUN wget https://jdbc.postgresql.org/download/postgresql-42.7.7.jar -O /postgresql.jar
RUN wget https://repo1.maven.org/maven2/com/hazelcast/hazelcast/5.6.0/hazelcast-5.6.0.jar -O /hazelcast.jar

COPY CounterMapStore.java /
RUN javac -encoding UTF-8 -cp /hazelcast.jar CounterMapStore.java


FROM hazelcast/hazelcast:5.6

COPY --from=builder /postgresql.jar /opt/hazelcast/lib/
COPY --from=builder /CounterMapStore.class /opt/hazelcast/lib/
COPY hazelcast.yaml /opt/hazelcast/config/
```

hazelcast.yaml

```yaml
hazelcast:
```

```yaml
map:
  counters:
    map-store:
      enabled: true
      class-name: CounterMapStore
      write-delay-seconds: 0 # Write-Through
```

2 Результати

На основі першого завдання з Web-counter, додати до нього Read-through та Write-through кешування.

У якості системи кешування, використати кластер Hazelcast з відповідними налаштуваннями https://docs.hazelcast.com/hazelcast/5.6/mapstore/working-with-external-data

Ідея полягає у тому, що значення каунтера зберігається у базі даних, але операції з ним відбувались не на пряму, а крізь кеш.

1. Налаштувати Read-through та Write-through на основі generic MapLoader та generic MapStore для БД PostgreSQL (або MySQL)

2. Припускаючи, що у нас буде 4 різних каунтерів, додати в Hazelcast Map, яка відповідає таблиці БД, 4 окремих записи з різними ID (1-4) та 0 у якості початкового значення. Перевірити через прямий запит до БД, що ці записи в ній з'явились

3. Для окремих створених ключів (каунтерів) повторити тести 1) - 4) з першої роботи, забезпечуючи коректність результатів (тобто щоб кінцевий результат був коректним) та міряючи час. Тобто тест 1 має використовувати каунтер 1, тест 2 - каунтер 2, і т.д.

   (1) Один клієнт робить послідовно 10K викликів, кінцеве значення count = 10K - порахувати кількість запитів в секунду

   (2) Два клієнти одночасно роблять по 10K викликів кожен, кінцеве значення count = 20K - порахувати кількість запитів в секунду

   (3) 5 клієнтів одночасно роблять по 10K викликів кожен, кінцеве значення count = 50K - порахувати кількість запитів в секунду

   (4) 10 клієнтів одночасно роблять по 10K викликів кожен, кінцеве значення count = 100K - порахувати кількість запитів в секунду

4. Перевірити кінцеві значення каунтерів у БД та порівняти отримані результати з продуктивності з отриманими у першій роботі.

```
[INFO] Connecting to Hazelcast...
[INFO] Clearing Counters...
[INFO] Data: [(1, 0), (2, 0), (3, 0), (4, 0)]
[INFO] Running Counter 1
[INFO] Counter ID: 1
[INFO] Clients: 1 | Increments per client: 10000
[RESULT] Time: 44.36 sec
[RESULT] Throughput: 225.41
[RESULT] Received: 10_000 (100.00% from Expected: 10_000)
[INFO] Running Counter 2
[INFO] Counter ID: 2
[INFO] Clients: 2 | Increments per client: 10000
[RESULT] Time: 69.05 sec
```

```
[RESULT] Throughput: 289.64
[RESULT] Received: 20_000 (100.00% from Expected: 20_000)
[INFO] Running Counter 3
[INFO] Counter ID: 3
[INFO] Clients: 5 | Increments per client: 10000
[RESULT] Time: 170.70 sec
[RESULT] Throughput: 292.91
[RESULT] Received: 50_000 (100.00% from Expected: 50_000)
[INFO] Running Counter 4
[INFO] Counter ID: 4
[INFO] Clients: 10 | Increments per client: 10000
[RESULT] Time: 354.13 sec
[RESULT] Throughput: 282.39
[RESULT] Received: 100_000 (100.00% from Expected: 100_000)
[INFO] Data: [(1, 10000), (2, 20000), (3, 50000), (4, 100000)]
```

```
[INFO] Один клієнт
[Name] Memory
[Sent] 10000 requests in 3.29 seconds
[Throughput]: 3040.52 requests/sec
[Name] DataBase
[Sent] 10000 requests in 10.75 seconds
[Throughput]: 929.95 requests/sec
[Name] Hazelcast Cash
[RESULT] Time: 44.36 sec
[RESULT] Throughput: 225.41

[INFO] Два клієнти
[Name] Memory
[Sent] 20000 requests in 6.67 seconds
[Throughput]: 2996.28 requests/sec
[Name] DataBase
[Sent] 20000 requests in 22.13 seconds
[Throughput]: 903.66 requests/sec
[Name] Hazelcast Cash
[RESULT] Time: 69.05 sec
[RESULT] Throughput: 289.64

[INFO] П''ять клієнтів
[Name] Memory
[Sent] 50000 requests in 17.83 seconds
[Throughput]: 2804.70 requests/sec
[Name] DataBase
[Sent] 50000 requests in 57.78 seconds
[Throughput]: 865.40 requests/sec
[Name] Hazelcast Cash
[RESULT] Time: 170.70 sec
[RESULT] Throughput: 292.91

[INFO] Десять клієнтів
\textbf{}
[Name] Memory
[Sent] 100000 requests in 39.65 seconds
[Throughput]: 2521.88 requests/sec
[Name] DataBase
[Sent] 100000 requests in 117.59 seconds
[Throughput]: 850.39 requests/sec
[Name] Hazelcast Cash
[RESULT] Time: 354.13 sec
[RESULT] Throughput: 282.39
```

Виходить повільний запис, але надійний та можна дуже швидко читати