ЗВІТ

Лабораторна робота №2
Реалізація каунтера з використанням Hazelcast (Distributed Map + CP IAtomicLong)

ПІБ	Одінцов Валерій Вячеславович
Група	ФБ-51мп
ОС / Середовище	Kali Linux, Docker + Java (Maven)

1. Мета роботи

1) Реалізувати розподілений каунтер на Hazelcast і продемонструвати проблему race condition при одночасному доступі.
2) Реалізувати коректний каунтер з песимістичним та оптимістичним блокуванням у Distributed Map.
3) Реалізувати каунтер на базі CP Subsystem за допомогою IAtomicLong (Raft) та перевірити коректність.
4) Поміряти час виконання та перевірити кінцеве значення каунтера.

2. Умови експерименту

Кількість потоків: 10

Кількість інкрементів на потік: 10 000

Очікуване кінцеве значення: 100 000

Кластер Hazelcast: 3 ноди (hz1, hz2, hz3), Hazelcast 5.4.x, CP Subsystem увімкнено

3. Підготовка середовища

3.1. Підняття 3 нод Hazelcast у Docker (файли конфігурації)

3.1.1 docker-compose.yml

```
services:
  hz1:
    image: hazelcast/hazelcast:5.4
    container_name: hz1
    hostname: hz1
    ports: ["5701:5701"]
    environment:
      JAVA_OPTS: "-Dhazelcast.config=/opt/hazelcast/config_ext/hazelcast.yaml"
    volumes:
      - ./hazelcast.yaml:/opt/hazelcast/config_ext/hazelcast.yaml:ro
```

```
  hz2:
    image: hazelcast/hazelcast:5.4
    container_name: hz2
    hostname: hz2
    ports: ["5702:5701"]
    environment:
      JAVA_OPTS: "-Dhazelcast.config=/opt/hazelcast/config_ext/hazelcast.yaml"
    volumes:
      - ./hazelcast.yaml:/opt/hazelcast/config_ext/hazelcast.yaml:ro

  hz3:
    image: hazelcast/hazelcast:5.4
    container_name: hz3
    hostname: hz3
    ports: ["5703:5701"]
    environment:
      JAVA_OPTS: "-Dhazelcast.config=/opt/hazelcast/config_ext/hazelcast.yaml"
    volumes:
      - ./hazelcast.yaml:/opt/hazelcast/config_ext/hazelcast.yaml:ro
```

3.1.2 hazelcast.yaml

```
hazelcast:
  cluster-name: counter-cluster

  network:
    join:
      multicast:
        enabled: false
      tcp-ip:
        enabled: true
        member-list:
          - hz1:5701
          - hz2:5701
          - hz3:5701

  map:
    counter-map:
      backup-count: 2

  cp-subsystem:
    cp-member-count: 3
    group-size: 3
```

3.2. Команди запуску кластера

```
docker-compose up -d
docker ps --filter "name=hz" --format "table {{.Names}}\t{{.Status}}\t{{.Ports}}"
```

```
└─$ docker ps --filter "name=hz" --format "table {{.Names}}\t{{.Status}}\t{{.Ports}}"

NAMES     STATUS         PORTS
hz3       Up 6 seconds   0.0.0.0:5703→5701/tcp, [::]:5703→5701/tcp
hz1       Up 6 seconds   0.0.0.0:5701→5701/tcp, :::5701→5701/tcp
hz2       Up 6 seconds   0.0.0.0:5702→5701/tcp, [::]:5702→5701/tcp
```

4. Підтвердження CP Subsystem та кількості нод

```
└─$ docker logs hz1 | grep -E "CP Subsystem is enabled with 3 members" | head -n 3

2025-12-17 10:14:24,004 [ INFO] [main] [c.h.c.CPSubsystem]: [hz1]:5701 [counter-cluster] [5.4.0] CP Subsystem is enabled with 3 members.
```

5. Реалізація (Java)

Проєкт зібрано за допомогою Maven. Запуск бенчмарка виконується командою:

```
mvn -q -DskipTests package
mvn -q exec:java -Dexec.args="--threads 10 --perThread 10000 --cpBatch 1"
```

Примітка: для стабільної роботи Java-клієнта з кластером у Docker вимкнено smart routing (cfg.getNetworkConfig().setSmartRouting(false);), оскільки ноди можуть рекламувати hostname (hz1/hz2/hz3), які не резолвляться на хостовій ОС.

5.1. Код програми (CounterBench.java)

```
package lab;

import com.hazelcast.client.HazelcastClient;
import com.hazelcast.client.config.ClientConfig;
import com.hazelcast.core.HazelcastInstance;
import com.hazelcast.cp.IAtomicLong;
import com.hazelcast.map.IMap;

import java.util.Arrays;
import java.util.concurrent.*;

public class CounterBench {

    static class Settings {
        int threads = 10;
        int perThread = 10_000;
        int cpBatch = 1; // 1 = incrementAndGet() (strict); >1 = addAndGet(batch) (faster)
        String clusterName = "counter-cluster";
        String[] addresses = {"127.0.0.1:5701","127.0.0.1:5702","127.0.0.1:5703"};
        String mapName = "counter-map";
        String key = "counter";
    }
```

```
    public static void main(String[] args) throws Exception {
        Settings s = parseArgs(args);
        long expected = (long) s.threads * s.perThread;

        System.out.println("Settings: threads=" + s.threads + ", perThread=" + s.perThread
+
                ", expected=" + expected + ", cpBatch=" + s.cpBatch);
        System.out.println("Cluster: " + s.clusterName + " @ " +
Arrays.toString(s.addresses));

        HazelcastInstance client = connect(s);

        try {
            IMap<String, Long> map = client.getMap(s.mapName);

            runScenario("1) Map / no locks", expected, () -> mapNoLocks(map, s));
            runScenario("2) Map / pessimistic lock()", expected, () -> mapPessimistic(map,
s));
            runScenario("3) Map / optimistic CAS replace()", expected, () ->
mapOptimistic(map, s));
            runScenario("4) CP / IAtomicLong", expected, () -> cpAtomic(client, s));

        } finally {
            client.shutdown();
        }
    }

    static HazelcastInstance connect(Settings s) {
        ClientConfig cfg = new ClientConfig();
        cfg.setClusterName(s.clusterName);

        var net = cfg.getNetworkConfig();
        net.addAddress(s.addresses);

        // Important for Docker: members advertise hostnames hz1/hz2/hz3 not resolvable on
host
        net.setSmartRouting(false);

        return HazelcastClient.newHazelcastClient(cfg);
    }

    static void runScenario(String name, long expected, Callable<Long> scenario) throws
Exception {
        long t0 = System.nanoTime();
        long value = scenario.call();
        double secs = (System.nanoTime() - t0) / 1_000_000_000.0;
        String ok = (value == expected) ? "OK" : "BAD";
        System.out.printf("%-38s time=%8.3fs  value=%d  expected=%d  => %s%n",
                name, secs, value, expected, ok);
    }

    static long mapNoLocks(IMap<String, Long> map, Settings s) throws Exception {
        map.put(s.key, 0L);
        runThreads(s.threads, () -> {
            for (int i = 0; i < s.perThread; i++) {
                Long v = map.get(s.key);
                map.put(s.key, v + 1);
            }
```

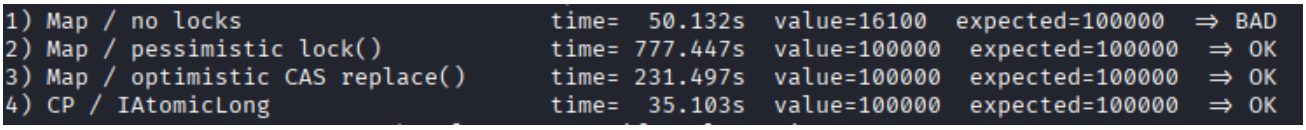
```
    });
    return map.get(s.key);
}

static long mapPessimistic(IMap<String, Long> map, Settings s) throws Exception {
    map.put(s.key, 0L);
    runThreads(s.threads, () -> {
        for (int i = 0; i < s.perThread; i++) {
            map.lock(s.key);
            try {
                Long v = map.get(s.key);
                map.put(s.key, v + 1);
            } finally {
                map.unlock(s.key);
            }
        }
    });
    return map.get(s.key);
}

static long mapOptimistic(IMap<String, Long> map, Settings s) throws Exception {
    map.put(s.key, 0L);
    runThreads(s.threads, () -> {
        for (int i = 0; i < s.perThread; i++) {
            while (true) {
                Long oldVal = map.get(s.key);
                if (map.replace(s.key, oldVal, oldVal + 1)) break;
            }
        }
    });
    return map.get(s.key);
}

static long cpAtomic(HazelcastInstance client, Settings s) throws Exception {
    IAtomicLong al = client.getCPSubsystem().getAtomicLong("counter");
    al.set(0L);

    if (s.cpBatch <= 1) {
        runThreads(s.threads, () -> {
            for (int i = 0; i < s.perThread; i++) {
                al.incrementAndGet();
            }
        });
    } else {
        int batch = s.cpBatch;
        int fullBatches = s.perThread / batch;
        int rem = s.perThread % batch;

        runThreads(s.threads, () -> {
            for (int i = 0; i < fullBatches; i++) {
                al.addAndGet(batch);
            }
            if (rem > 0) al.addAndGet(rem);
        });
    }

    return al.get();
}
```

```
    static void runThreads(int threads, Runnable worker) throws InterruptedException {
        ExecutorService pool = Executors.newFixedThreadPool(threads);
        CountDownLatch latch = new CountDownLatch(threads);

        for (int t = 0; t < threads; t++) {
            pool.submit(() -> {
                try { worker.run(); }
                finally { latch.countDown(); }
            });
        }

        latch.await();
        pool.shutdown();
    }

    static Settings parseArgs(String[] args) {
        Settings s = new Settings();
        for (int i = 0; i < args.length; i++) {
            switch (args[i]) {
                case "--threads" -> s.threads = Integer.parseInt(args[++i]);
                case "--perThread" -> s.perThread = Integer.parseInt(args[++i]);
                case "--cpBatch" -> s.cpBatch = Integer.parseInt(args[++i]);
                default -> { }
            }
        }
        return s;
    }
}
```

6. Результати експерименту

Очікуване кінцеве значення каунтера: 100 000

Сценарій	Час, с	Значення	Очікуване	Статус
1) Map / no locks	50.132	16100	100000	BAD
2) Map / pessimistic lock()	777.447	100000	100000	OK
3) Map / optimistic CAS replace()	231.497	100000	100000	OK
4) CP / IAtomicLong incrementAndGet()	35.103	100000	100000	OK



7. Аналіз та висновки

1) Map / no locks: результат BAD через race condition - потоки одночасно читають одне й те саме значення та перезаписують його після інкременту, внаслідок чого частина інкрементів втрачається.
2) Map / pessimistic lock(): результат OK, але найбільший час виконання через lock/unlock на кожній операції.
3) Map / optimistic CAS: результат OK та швидше за песимістичне блокування, однак можливі повтори при конфліктах.
4) CP / IAtomicLong: результат OK. CP Subsystem забезпечує узгодженість та коректність за допомогою протоколу Raft.