Проєктування високонавантаженних систем. Лабораторна 2, звіт

Михайло Голуб

24 листопада 2025 р.

1 Завдання лабораторної роботи

1.1 Загальне

1. Встановити і налаштувати Hazelcast 5.4.x (у новіших версіях частина необхідного для виконання завдань функціоналу є платною)
2. Сконфігурувати і запустити 3 ноди (інстанси) об'єднані в кластер або як частину Java-застосування, або як окремі застосування У справжній системі кожна нода має запускатись на окремому сервері.
3. Далі, на основі прикладу з Distributed Map, напишіть код який буде емулювати інкремент значення для одного й того самого ключа у циклі до 10K. Це необхідно робити у 10 потоках.
4. Виходячи з того, що 10 потоків інкрементують каунтер 10K разів кожен, то остаточне значення каунтера має бути 10*10_000 = 100_000. Для імплементації спочатку скористаємось Distributed Map у Hazelcast
5. Реалізуйте каунтер без блокувань. Поміряйте час виконання, та подивиться чи коректне кінцеве значення каунтера ви отримаєте.
6. Реалізуйте каунтер з використанням песимістичного блокування. Поміряйте час виконання, та подивиться чи коректне кінцеве значення каунтера ви отримаєте.
7. реалізуйте каунтер з використанням оптимістичного блокування. Поміряйте час виконання, та подивиться чи коректне кінцеве значення каунтера ви отримаєте.
8. Реалізуйте каунтер з використанням IAtomicLong та увімкнувши підтимку CP Sysbsystem на основі трьох нод. УВАГА! Без CP Sysbsystem не гарантується коректність результату (у протоколі мають бути логи Hazelcast з яких видно, що CP Subsystem активована та складається з 3-х нод) Поміряйте час виконання, та подивиться чи коректне кінцеве значення каунтера ви отримаєте.

1.2 Вимоги до звіту та реалізації

- Мова реалізації будь-яка
- Має бути надано код програми/скрипта та результати виконання
- Приведно лог, який видають ноди Hazelcast, де буде видно що кластер складається з 3-х нод і що активована CP Subsystem

2 Хід роботи

2.1 Запуск Hazelcast

Hazelcast локально захощено через Docker. Створюються три контейнери контейнери Hazelcast 5.4.0: `hz-node1`, `hz-node2`, `hz-node3` на портах 5701, 5702 та 5703 відповідно. Та контейнер Hazelcast management-center 5.4.0: `hz-mz` на порті 8080

Лістинг 1: docker-compose.yml

```
version: '3.8'

services:
  hazelcast1:
    image: hazelcast/hazelcast:5.4.0
    container_name: hz-node1
    ports:
      - "5701:5701"
    volumes:
      - ./hazelcast.yaml:/opt/hazelcast/config/
    hazelcast.yaml
    environment:
      - JAVA_OPTS=-Dhazelcast.local.publicAddress=
    hazelcast1:5701 -Dhazelcast.config=/opt/hazelcast/
    config/hazelcast.yaml
    networks:
      - hz-network

  hazelcast2:
    image: hazelcast/hazelcast:5.4.0
    container_name: hz-node2
    ports:
      - "5702:5701"
    volumes:
      - ./hazelcast.yaml:/opt/hazelcast/config/
    hazelcast.yaml
    environment:
```

```
      - JAVA_OPTS=-Dhazelcast.local.publicAddress=hazelcast2:5701 -Dhazelcast.config=/opt/hazelcast/config/hazelcast.yaml
    depends_on:
      - hazelcast1
    networks:
      - hz-network

  hazelcast3:
    image: hazelcast/hazelcast:5.4.0
    container_name: hz-node3
    ports:
      - "5703:5701"
    volumes:
      - ./hazelcast.yaml:/opt/hazelcast/config/hazelcast.yaml
    environment:
      - JAVA_OPTS=-Dhazelcast.local.publicAddress=hazelcast3:5701 -Dhazelcast.config=/opt/hazelcast/config/hazelcast.yaml
    depends_on:
      - hazelcast1
    networks:
      - hz-network

  management-center:
    image: hazelcast/management-center:5.4.0
    container_name: hz-mc
    ports:
      - "8080:8080"
    environment:
      - MC_DEFAULT_CLUSTER=dev
      - MC_DEFAULT_CLUSTER_MEMBERS=hazelcast1,hazelcast2,hazelcast3
    depends_on:
      - hazelcast1
    networks:
      - hz-network

networks:
  hz-network:
    driver: bridge
```

Після запуску усі чотири контейнери запущені успішно і існує кластер з трьома нодами. Про це свідчить ця частина логу:

```
Members {size:3, ver:3} [
```

```
    Member [hazelcast1]:5701 - 360c484e-0413-4f01-b810-edd4f20fb514
    Member [hazelcast2]:5701 - 912b48a7-4b31-4303-9e30-06560e463ef1
    Member [hazelcast3]:5701 - ae62cc6a-26c6-4188-b174-ce60c234888c this
]
```

CP система не запущена:

```
CP Subsystem is not enabled. CP data structures will operate in UNSAFE mode!
Please note that UNSAFE mode will not provide strong consistency guarantees.
```

2.2 Клієнт

Майже ідентичний клієнту реалізованому в першій лабораторній роботі. Додано лог проходження певної частки запитів, для розуміння чи просувається виконання запитів.

Лістинг 2: client.py

```python
import requests
import threading
import time
import random
from tqdm import tqdm

SERVER = "http://127.0.0.1:5000"
CALLS_PER_CLIENT = 10_000
NUM_CLIENTS = 10

print_lock = threading.Lock()

def worker(client_id):
    bar = tqdm(range(CALLS_PER_CLIENT), position=client_id, desc =
    str(client_id), leave=False)
    for i in bar:
        trying = True
        while trying:
            try:
                requests.get(f"{SERVER}/inc")
                trying = False
            except requests.exceptions.RequestException:
                print(client_id, "worker, request denied, N =",i)
                time.sleep(random.random())
    with print_lock:
        bar.close()

def main():
    print(f"Starting {NUM_CLIENTS} clients x {CALLS_PER_CLIENT}
    calls each...")
    start = time.time()

    threads = []
    for i in range(NUM_CLIENTS):
        t = threading.Thread(target=worker, args=(i,))
        t.start()
        threads.append(t)
    print("MAIN: Threads created")
```

```
    for t in threads:
        t.join()
    print("MAIN: Threads joined")
    end = time.time()
    elapsed = end - start
    final_count = -1
    i = 0
    while final_count == -1 and i < 5:
        try:
            final_count = int(requests.get(f"{SERVER}/count").text)
            print(final_count)
        except:
            final_count = -1
            i+=1
            time.sleep(1)

    total_calls = CALLS_PER_CLIENT * NUM_CLIENTS
    throughput = total_calls / elapsed

    print(f"Final count: {final_count}")
    print(f"Total time:  {elapsed:.2f} s")
    print(f"Throughput:  {throughput:.2f} requests/sec")

if __name__ == "__main__":
    main()
    input()
```

Приклад логу під час роботи клієнта:

```
Starting 10 clients x 10000 calls each...
MAIN: Threads created
0:   7%| 709/10000 [01:17<05:28, 28.32it/s]
1:   7%| 673/10000 [01:17<06:52, 22.62it/s]
2:   7%| 677/10000 [01:17<06:45, 22.98it/s]
3:   7%| 666/10000 [01:17<06:13, 25.02it/s]
4:   7%| 675/10000 [01:17<06:10, 25.14it/s]
5:   7%| 675/10000 [01:17<06:03, 25.67it/s]
6:   7%| 681/10000 [01:17<05:54, 26.27it/s]
7:   7%| 673/10000 [01:17<07:18, 21.25it/s]
8:   7%| 675/10000 [01:17<06:47, 22.89it/s]
9:   7%| 673/10000 [01:17<06:26, 24.10it/s]
```

Приклад логу після завершення роботи клієнта:

```
Starting 10 clients x 10000 calls each...
MAIN: Threads created
MAIN: Threads joined
47221
Final count: 47221
Total time:  1105.88 s
Throughput:  90.43 requests/sec
```

2.3 Лічильник без блокування

Застосунок отримує значення лічильника, інкрементує його всередині потоку застосунку і записує назад в Hazelcast. Таким чином, блокування відсутні.

Лістинг 3: app_no_block.py

```python
from flask import Flask
from hazelcast.client import HazelcastClient

print("Starting with no block")

app = Flask(__name__)

HZ_ADDRESSES = [
    "127.0.0.1:5701",
    "127.0.0.1:5702",
    "127.0.0.1:5703"
]

COUNTER_NAME = "likes"

print("Connecting to Hazelcast...")
client = HazelcastClient(
    cluster_members=HZ_ADDRESSES
)

cp = client.cp_subsystem
counter = cp.get_atomic_long(COUNTER_NAME).blocking()

counter.set(0)
print(f"Hazelcast connected. Counter '{COUNTER_NAME}' reset to 0.")


@app.route("/inc")
def inc():
    new_value = counter.get() + 1
    counter.set(new_value)
    return str(new_value)


@app.route("/count")
def count():
    current = counter.get()
    return str(current)
```

Результати тестування:
Час роботи: 1106 секунд
Пропускна здатність: 90.43 запитів/с
Кінцеве значення лічильника: 47221

2.4 Лічильник з песимістичним блокуванням

Застосунок блокує блок (або очікує доки може заблокувати), отримує значення лічильника, інкрементує його всередині потоку застосунку, записує

назад в Hazelcast та відпускає блок.

Лістинг 4: app_pessimistic_block.py

```python
from flask import Flask
from hazelcast.client import HazelcastClient

print("Starting with pessimistic block")

app = Flask(__name__)

HZ_ADDRESSES = [
    "127.0.0.1:5701",
    "127.0.0.1:5702",
    "127.0.0.1:5703"
]

COUNTER_NAME = "likes"

print("Connecting to Hazelcast...")
client = HazelcastClient(cluster_members=HZ_ADDRESSES)
counter_map = client.get_map("map").blocking()



counter_map.put(COUNTER_NAME, 0)
print(f"Hazelcast connected. Counter '{COUNTER_NAME}' reset to 0.")


@app.route("/inc")
def inc():
    counter_map.lock(COUNTER_NAME)
    value = counter_map.get(COUNTER_NAME)
    new_value = value + 1
    counter_map.put(COUNTER_NAME, new_value)
    counter_map.unlock(COUNTER_NAME)
    return str(new_value)



@app.route("/count")
def count():
    value = counter_map.get(COUNTER_NAME)
    return str(value)
```

Результати тестування:
Час роботи: 3732 секунд
Пропускна здатність: 26.79 запитів/с
Кінцеве значення лічильника: 100к

2.5 Лічильник з оптимістичним блокуванням

Лістинг 5: app_optimistic_block.py

```python
from flask import Flask
from hazelcast.client import HazelcastClient
```

```
import time
import random

print("Starting with optimistic lock (CAS)")

app = Flask(__name__)

HZ_ADDRESSES = [
    "127.0.0.1:5701",
    "127.0.0.1:5702",
    "127.0.0.1:5703"
]

COUNTER_NAME = "likes"

client = HazelcastClient(cluster_members=HZ_ADDRESSES)
counter_map = client.get_map("map").blocking()


counter_map.put(COUNTER_NAME, 0)
print(f"Hazelcast connected. Counter '{COUNTER_NAME}' reset to 0.")


def cas_increment():
    backoff = 0.0005
    max_backoff = 0.05

    while True:
        old = counter_map.get(COUNTER_NAME)
        new = old + 1
        if counter_map.replace_if_same(COUNTER_NAME, old, new):
            return new
        jitter = random.uniform(0, backoff)
        time.sleep(backoff + jitter)
        backoff = min(backoff * 2, max_backoff)

@app.route("/inc")
def inc():
    new = cas_increment()
    return str(new)

@app.route("/count")
def count():
    return str(counter_map.get(COUNTER_NAME))
```

Результати тестування:
Час роботи: 1964 секунди
Пропускна здатність: 50.91 запитів/с
Кінцеве значення лічильника: 100к

2.6 CP Subsystem та IAtomicLong

Додано файл `hazelcast.yaml` який вказує що необхідно активувати CP Subsystem

Лістинг 6: hazelcast.yaml

```
hazelcast:
```

```
  cluster-name: dev

  cp-subsystem:
    cp-member-count: 3
    group-size: 3
    session-time-to-live-seconds: 30
    session-heartbeat-interval-seconds: 5
    missing-cp-member-auto-removal-seconds: 300

  network:
    join:
      multicast:
        enabled: true
      tcp-ip:
        enabled: false
```

В лозі нод наявне повідомлення:

```
CP Subsystem is enabled with 3 members.
```

Лістинг 7: app_CP.py

```
from flask import Flask
from hazelcast import HazelcastClient

print("Starting with IAtomicLong")

app = Flask(__name__)

HZ_ADDRESSES = [
    "127.0.0.1:5701",
    "127.0.0.1:5702",
    "127.0.0.1:5703"
]

COUNTER_NAME = "likes"

client = HazelcastClient(cluster_members=HZ_ADDRESSES)
counter = client.cp_subsystem.get_atomic_long(COUNTER_NAME).
    blocking()

counter.set(0)
print(f"Hazelcast connected. Counter '{COUNTER_NAME}' reset to 0.")


@app.route("/inc")
def inc():
    new_value = counter.increment_and_get()
    return str(new_value)


@app.route("/count")
def count():
    return str(counter.get())
```

Результати тестування:
Час роботи: 1126 секунд
Пропускна здатність: 88.74 запитів/с
Кінцеве значення лічильника: 100к

3 Результати

Варіант блокування	Час роботи	Пропускна здатність	Значення лічильника
Відсутнє	1106	90.43	47721
Песимістичне	3732	26.79	100k
Оптимістичне	1964	50.91	100k
CP Subsystem IAtomicLong	1126	88.74	100k