

Проектування високонавантажених систем.

Лабораторна 2, звіт

Михайло Голуб

20 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 скористаємося 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 пессимістичного 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
- Приведено лог, який видають ноди Hazelcast, де буде видно що кластер складається з 3-х нод і що активована CP Subsystem

2 Запуск Hazelcast

Hazelcast локально захочено 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

```
1 version: '3.8'
2
3 services:
4   hazelcast1:
5     image: hazelcast/hazelcast:5.4.0
6     container_name: hz-node1
7     ports:
8       - "5701:5701"
9     environment:
10      - HZ_NETWORK_JOIN_MULTICAST_ENABLED=false
11      - HZ_NETWORK_JOIN_TCPIP_ENABLED=true
12      - HZ_NETWORK_JOIN_TCPIP_MEMBERS=hazelcast1,
13        hazelcast2,hazelcast3
14      - HZ_CP_MEMBER_COUNT=3
15      - JAVA_OPTS=-Dhazelcast.local.publicAddress=
16        hazelcast1:5701
17     networks:
18       - hz-network
19
20   hazelcast2:
21     image: hazelcast/hazelcast:5.4.0
22     container_name: hz-node2
23     ports:
24       - "5702:5701"
25     environment:
26      - HZ_NETWORK_JOIN_MULTICAST_ENABLED=false
27      - HZ_NETWORK_JOIN_TCPIP_ENABLED=true
28      - HZ_NETWORK_JOIN_TCPIP_MEMBERS=hazelcast1,
29        hazelcast2,hazelcast3
```

```

27         - HZ_CP_MEMBER_COUNT=3
28         - JAVA_OPTS=-Dhazelcast.local.publicAddress=
29 hazelcast2:5701
30     depends_on:
31         - hazelcast1
32     networks:
33         - hz-network
34
35 hazelcast3:
36     image: hazelcast/hazelcast:5.4.0
37     container_name: hz-node3
38     ports:
39         - "5703:5701"
40     environment:
41         - HZ_NETWORK_JOIN_MULTICAST_ENABLED=false
42         - HZ_NETWORK_JOIN_TCPIP_ENABLED=true
43         - HZ_NETWORK_JOIN_TCPIP_MEMBERS=hazelcast1,
44             hazelcast2,hazelcast3
45         - HZ_CP_MEMBER_COUNT=3
46         - JAVA_OPTS=-Dhazelcast.local.publicAddress=
47 hazelcast3:5701
48     depends_on:
49         - hazelcast1
50     networks:
51         - hz-network
52
53 management-center:
54     image: hazelcast/management-center:5.4.0
55     container_name: hz-mc
56     ports:
57         - "8080:8080"
58     environment:
59         - MC_DEFAULT_CLUSTER=dev
60         - MC_DEFAULT_CLUSTER_MEMBERS=hazelcast1,
61             hazelcast2,hazelcast3
62     depends_on:
63         - hazelcast1
64     networks:
65         - hz-network
66
67 networks:
68     hz-network:
69         driver: bridge

```

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

```
[ WARN] [main] [c.h.c.CPSubsystem]: [hazelcast3]:5701 [dev] [5.4.0] CP Subsystem is not enabled
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
]
```

3 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

```
1 import requests
2 import threading
3 import time
4 import random
5 from tqdm import tqdm
6
7 SERVER = "http://127.0.0.1:8080"
8 CALLS_PER_CLIENT = 10_000
9 NUM_CLIENTS = 10
10
11 print_lock = threading.Lock()
12
13 def worker(client_id):
14     bar = tqdm(range(CALLS_PER_CLIENT), position=client_id, desc =
15                str(client_id), leave=False)
16     for i in bar:
17         trying = True
18         while trying:
19             try:
20                 requests.get(f"{SERVER}/inc")
21                 trying = False
22             except requests.exceptions.RequestException:
23                 print(client_id, "worker, request denied, N =", j *
24                       CALLS_PER_CLIENT // N_REPORTS + i)
25                 time.sleep(random.random())
26         with print_lock:
27             bar.close()
28
29 def main():
30     print(f"Starting {NUM_CLIENTS} clients x {CALLS_PER_CLIENT}
31 calls each...")
32     start = time.time()
33
34     threads = []
35     for i in range(NUM_CLIENTS):
36         t = threading.Thread(target=worker, args=(i,))
37         t.start()
38         threads.append(t)
39     print("MAIN: Threads created")
```

```

37     for t in threads:
38         t.join()
39     print("MAIN: Threads joined")
40     end = time.time()
41     elapsed = end - start
42     final_count = -1
43     i = 0
44     while final_count == -1 and i < 5:
45         try:
46             final_count = int(requests.get(f"{SERVER}/count").text)
47             print(final_count)
48         except:
49             final_count = -1
50             i+=1
51             time.sleep(1)
52
53     total_calls = CALLS_PER_CLIENT * NUM_CLIENTS
54     throughput = total_calls / elapsed
55
56     print(f"Final count: {final_count}")
57     print(f"Total time: {elapsed:.2f} s")
58     print(f"Throughput: {throughput:.2f} requests/sec")
59
60 if __name__ == "__main__":
61     main()
62     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
50663
Final count: 50663
Total time: 1184.66 s
Throughput: 84.41 requests/sec

```

4 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

```
 1 from flask import Flask
 2 from hazelcast.client import HazelcastClient
 3
 4 print("Starting with no block")
 5
 6 app = Flask(__name__)
 7
 8 HZ_ADDRESSES = [
 9     "127.0.0.1:5701",
10     "127.0.0.1:5702",
11     "127.0.0.1:5703"
12 ]
13
14 COUNTER_NAME = "likes-counter"
15
16 print("Connecting to Hazelcast...")
17 client = HazelcastClient(
18     cluster_members=HZ_ADDRESSES,
19     cluster_name="dev",
20 )
21
22 cp = client.cp_subsystem
23 counter = cp.get_atomic_long(COUNTER_NAME).blocking()
24
25 counter.set(0)
26 print(f"Hazelcast connected. Counter '{COUNTER_NAME}' reset to 0.")
27
28
29 @app.route("/inc")
30 def inc():
31     new_value = counter.get() + 1
32     counter.set(new_value)
33     return str(new_value)
34
35
36 @app.route("/count")
37 def count():
38     current = counter.get()
39     return str(current)
40
41
42 if __name__ == "__main__":
43     try:
44         app.run(host="0.0.0.0", port=5000, debug=False)
45     finally:
46         client.shutdown()
```

Результати тестування:
Час роботи: 1184 секунди

Пропускна здатність: 84.41 запитів/с
Кінцеве значення лічильника: 32582

5 Лічильник з пессимістичним 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

```
1 from flask import Flask
2 from hazelcast.client import HazelcastClient
3
4 print("Starting with pessimistic block")
5
6 app = Flask(__name__)
7
8 HZ_ADDRESSES = [
9     "127.0.0.1:5701",
10    "127.0.0.1:5702",
11    "127.0.0.1:5703"
12 ]
13
14 client = HazelcastClient(cluster_members=HZ_ADDRESSES)
15 counter_map = client.get_map("likes-map").blocking()
16
17 COUNTER_KEY = "likes"
18
19 counter_map.put(COUNTER_KEY, 0)
20 print("Hazelcast connected. Counter reset.")
21
22
23 @app.route("/inc")
24 def inc():
25     counter_map.lock(COUNTER_KEY)
26     value = counter_map.get(COUNTER_KEY)
27     new_value = value + 1
28     counter_map.put(COUNTER_KEY, new_value)
29     counter_map.unlock(COUNTER_KEY)
30     return str(new_value)
31
32
33
34 @app.route("/count")
35 def count():
36     value = counter_map.get(COUNTER_KEY)
37     return str(value)
38
39
40 if __name__ == "__main__":
41     app.run(host="0.0.0.0", port=5000, debug=False)
```

Задля економії часу, зменшено кількість запитів клієнтів з 10000 до 10.
Результати тестування:

Час роботи: 1290 секунд
Пропускна здатність: 0.08 запитів/с
Кінцеве значення лічильника: 100