Task 2 Hazelcast

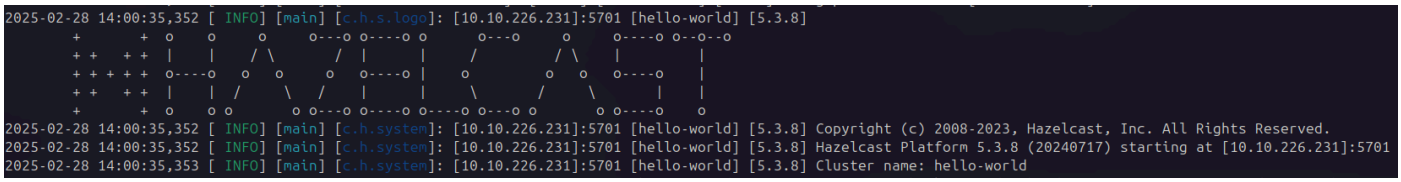
Автор - Тарас Лисун
Репозиторій: https://github.com/taraslysun/SA-labs

1. *Встановити і налаштувати Hazelcast*

Вирішив використовувати docker-версію hazelcast для цього завдання, тому створив мережу

```
docker network create hazelcast-network
```

В цій мережі за допомогою команди з документації успішно створив ноду



2. *Сконфігурувати і запустити 3 ноди (інстанси) об'єднані в кластер або як частину Java-застосування, або як окремі застосування*

Додатково додав ще 2 інстанси, але потрібно було зробити port mapping в докері, щоб уникнути конфліктів

```
Members {size:3, ver:3} [
        Member [10.10.226.231]:5701 - 51cc7b0f-a157-44ef-9aeb-9214ebb61910
        Member [10.10.226.231]:5702 - 68195965-cc4f-4c69-9fdd-d8fd8feb05b1
        Member [10.10.226.231]:5703 - 3bd191db-df74-4199-bdd9-dc050f21124a this
]
```

3. *Продемонструйте роботу Distributed Map*

Створив мапу використовуючи пайтон-клієнт і записав туди 1000 значень

main.py > ...

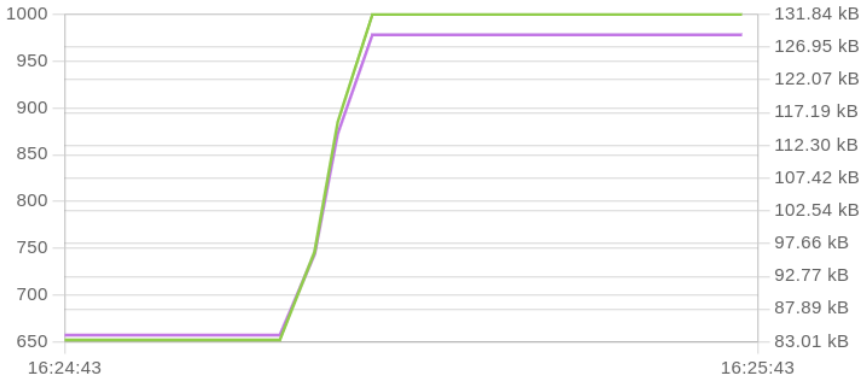
```python
import hazelcast

if __name__ == "__main__":
    client = hazelcast.HazelcastClient(
    cluster_name="hello-world",
    )

    # Create a Distributed Map in the cluster
    map = client.get_map("my-distributed-map").blocking()


    map.put_all({ i:"hello"+str(i) for i in range(1000) })
```

Розподіл ключів по трьох нодах:

Map Statistics (In-Memory Format: BINARY)

Member	Entries	Gets	Puts
10.10.226.231:5701	320	0	320
10.10.226.231:5702	332	0	332
10.10.226.231:5703	348	0	348
TOTAL	1,000	0	1,000

Якщо вимкнути одну ноду через ctrl+c, дані розподіляться по двох інших:

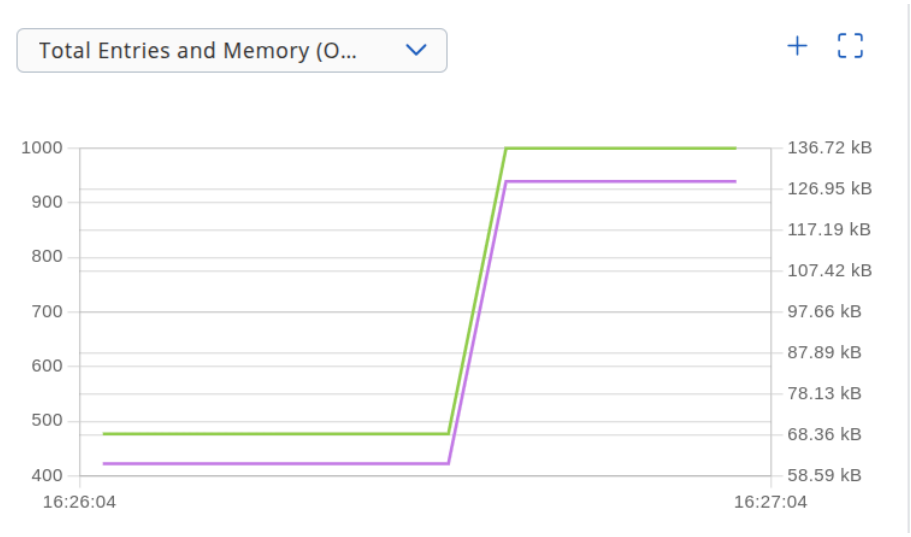


Map Statistics (In-Memory Format: BINARY)

Member	Entries	Gets
10.10.226.231:5701	477	0
10.10.226.231:5702	523	0
TOTAL	1,000	0

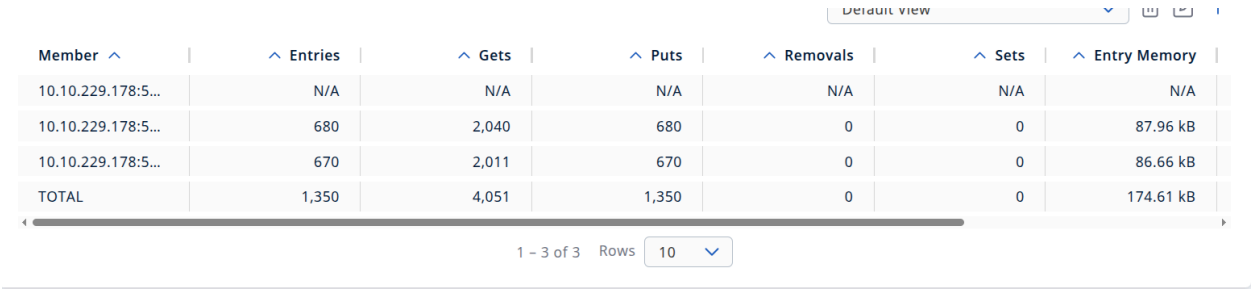
1 – 2 of 2 Ro

Якщо вимкнути ще одну, дані відповідно перейдуть на останні.



Member	Entries	Gets
10.10.226.231:5701	1,000	0
TOTAL	1,000	0

Якщо вимикати ноди через **kill -9**, то тоді hazelcast не встигає обробити припинення роботи ноди, і дані втрачаються (додав ще 1000 значень, тому загалом мало б бути 2000, а є 1350)



Member	Entries	Gets	Puts	Removals	Sets	Entry Memory
10.10.229.178:5...	N/A	N/A	N/A	N/A	N/A	N/A
10.10.229.178:5...	680	2,040	680	0	0	87.96 kB
10.10.229.178:5...	670	2,011	670	0	0	86.66 kB
TOTAL	1,350	4,051	1,350	0	0	174.61 kB

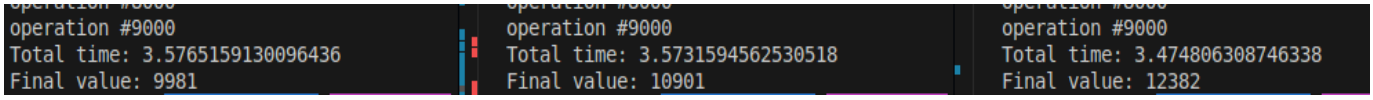
1 – 3 of 3 Rows 10

4. *Продемонструйте роботу Distributed Map without locks*

Зробив файл для наповнення non-blocking map використовуючи python-клієнта і запустив в 3-х окремих процесах відносно одночасно

```python
import hazelcast
import time

if __name__ == "__main__":
    client = hazelcast.HazelcastClient(
    cluster_name="hello-world",
    )

    map = client.get_map("my-distributed-map")
    map.put_if_absent("key", 0)
    start_time = time.time()
    for i in range(10000):
        val = int(map.get('key').result())
        val = val+1
        map.put('key', val)
        if i % 1000 == 0:
            print("operation #"+str(i))

    final_value = map.get("key").result()
    print("Total time:", time.time() - start_time)
    print("Final value:", final_value)

    client.shutdown()
```

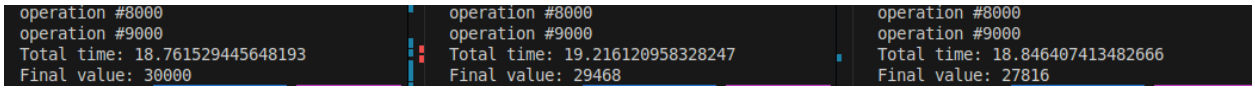
Оскільки доступ до мапи був неблокуючим, то, очевидно, виникнув data race і результат був неточним



5. *Зробіть те саме з використанням песимістичним блокування та поміряйте час*

```python
import hazelcast
import time

if __name__ == "__main__":
    client = hazelcast.HazelcastClient(
        cluster_name="hello-world",
    )

    map = client.get_map("my-distributed-map")
    map.put_if_absent("key", 0).result()
    start_time = time.time()
    for i in range(10000):
        map.lock("key").result()
        try:
            val = map.get("key").result()
            new_val = int(val) + 1
            map.put("key", new_val).result()
        finally:
            map.unlock("key").result()

        if i % 1000 == 0:
            print("operation #"+str(i))

    final_value = map.get("key").result()
    print("Total time:", time.time() - start_time)
    print("Final value:", final_value)

    client.shutdown()
```

Зробив з песимістичним блокуванням. Також запустив на трьох різних клієнтах. Результат, як і очікувалось, був точним, але досить довгим



Тобто середній час близько 18.9 секунди.

6. *Зробіть те саме з використанням оптимістичним блокуванням та поміряйте час*

```python
import hazelcast
import time

if __name__ == "__main__":
    client = hazelcast.HazelcastClient(
        cluster_name="hello-world",
    )

    map = client.get_map("my-distributed-map")
    map.put_if_absent("key", 0).result()
    start_time = time.time()

    for i in range(10000):
        while True:
            val = map.get("key").result()
            new_val = int(val) + 1

            if map.replace_if_same("key", val, new_val).result():
                break
        if i % 1000 == 0:
            print("operation #"+str(i))

    final_value = map.get("key").result()
    print("Total time:", time.time() - start_time)
    print("Final value:", final_value)

    client.shutdown()
```

Результати:

```
operation #8000
operation #9000
Total time: 11.347018003463745
Final value: 27393
```

```
operation #8000
operation #9000
Total time: 11.765206098556519
Final value: 29510
```

```
operation #8000
operation #9000
Total time: 11.362004041671753
Final value: 30000
```

Як можна побачити, час набагато менший за запуск з песимістичним блокуванням, майже вдвічі.

7. *Порівняйте результати кожного з запусків*

Як було видно з попередніх запусків, найшвидше працював метод з неблокуючим заповненням, але він створює можливість гонитви даних, тому не є нормальним рішенням для цієї проблеми. Далі по швидкості був оптимістичний метод блокування, який працює швидше за песимістичний, бо в цьому випадку просто біжить в циклі і пробує змінити значення. І найповільнішим був метод з песимістичним блокуванням.

8. *Робота з Bounded queue*

Конфіг черги:

```yaml
hazelcast:
  queue:
    queue:
      max-size: 10
```

Також, оскільки розгортання було через докер, потрібно було скопіювати цей файл замість того, що був по замовчуванню:

```
docker run \
    -d\
    -v "$(pwd)"/hazelcast.yaml:/opt/hazelcast/hazelcast.yaml \
    --network hazelcast-network \
    -e HZ_NETWORK_PUBLICADDRESS=$HOST_IP:$MAP_PORT \
    -e HZ_CLUSTERNAME=hello-world \
    -e HAZELCAST_CONFIG=hazelcast.yaml \
    -p $MAP_PORT:$PORT hazelcast/hazelcast:5.3.8
```

Створення записувача:

```python
import hazelcast
import hazelcast.config
import time

if __name__ == '__main__':

    client = hazelcast.HazelcastClient(
        cluster_name="hello-world",
    )

    queue = client.get_queue("queue")
    for i in range(1, 101):
        while True:
            if queue.offer(i, timeout=2).result():
                print(f"added {i}")
                break
            else:
                print("queue is full")

client.shutdown()
```

Створення читача:

```python
import hazelcast

if __name__ == '__main__':
    client = hazelcast.HazelcastClient(
        cluster_name="hello-world",
    )

    queue = client.get_queue("queue")

    while True:
        item = queue.take().result()
        if item is None:
            break
        print(f"taken {item}")


    client.shutdown()
```

Якщо записувач дойшов до значення 10, але ніхто з читачів не взяв значення з черги, то він зупиняється

```
added 1
added 2
added 3
added 4
added 5
added 6
added 7
added 8
added 9
added 10
queue is full
```

Також, якщо запустити більше ніж одного читача, то дані при читанні розподіляються між ними по черзі