Проєктування високонавантаженних систем. Лабораторна 4, звіт

Михайло Голуб

10 грудня 2025 р.

1 Завдання лабораторної роботи

1.1 Налаштування реплікації

1. Налаштувати реплікацію в конфігурації: Primary with Two Secondary Members (P-S-S) (всі ноди можуть бути запущені як окремі процеси або у Docker контейнерах)
2. Спробувати зробити запис з однією відключеною нодою та write concern рівнім 3 та нескінченім таймаутом. Спробувати під час таймаута включити відключену ноду
3. Аналогічно попередньому пункту, але задати скінченний таймаут та дочекатись його закінчення. Перевірити чи данні записались і чи доступні на читання з рівнем readConcern: “majority”
4. Продемонструвати перевибори primary node відключивши поточний primary (Replica Set Elections) і що після відновлення роботи старої primary на неї реплікуються нові дані, які з’явилися під час її простою

1.2 Аналіз продуктивності та перевірка цілісності

Аналогічно попереднім завданням, необхідно буде створити колекцію (таблицю) з каунтером лайків. Далі з 10 окремих клієнтів одночасно запустити інкерементацію каунтеру лайків по 10_000 на кожного клієнта з різними опціями взаємодії з MongoDB.

- Вказавши у парметрах `findOneAndUpdate writeConcern = 1` (це буде означати, що запис іде тільки на Primary ноду і не чекає відповіді від Secondary), запустіть 10 клієнтів з інкрементом по 10_000 на кожному з них. Виміряйте час виконання та перевірте чи кінцеве значення буде дорівнювати очікуваному - 100K

- Вказавши у парметрах `findOneAndUpdate writeConcern = majority` (це буде означати, що Primary чекає поки значення запишется на більшість нод), запустіть 10 клієнтів з інкрементом по 10_000 на кожному з них. Виміряйте час виконання та перевірте чи кінцеве значення буде дорівнювати очікуваному - 100K
- Повторно запустить код при `writeConcern = 1`, але тепер під час роботи відключіть Primary ноду і подивитись що буде обрана інша Primary нода, яка продовжить обробку запитів, і чи кінцевий результат буде коректним.
- Повторно запустить код при `writeConcern = majority`, але тепер під час роботи відключіть Primary ноду і подивитись що буде обрана інша Primary нода, яка продовжить обробку запитів, і чи кінцевий результат буде коректним.

При `writeConcern = 1` деякі записи можуть губитись під час раптового відключення. При `writeConcern = majority` має виходити очікуваний результат.

1.3 Вимоги до звіту та реалізації

- команди та результати їх виконання у вигляді скріншотів
- аналіз отриманих результатів

2 Хід роботи

2.1 Реалізація MongoDB

MongoDB запущена через Docker відповідним docker-compose:

Лістинг 1: docker-compose.yml

```
version: "3.8"
services:
  mongo1:
    image: mongo:7.0
    container_name: mongo1
    command: ["--replSet", "rs0", "--bind_ip_all", "--port", "27017"]
    ports:
      - "27017:27017"
    extra_hosts:
      - "host.docker.internal:host-gateway"
    volumes:
      - mongo1_data:/data/db
      - mongo1_config:/data/configdb
```

```
  mongo2:
    image: mongo:7.0
    container_name: mongo2
    command: ["--replSet", "rs0", "--bind_ip_all", "--port", "27018"]
    ports:
      - "27018:27018"
    extra_hosts:
      - "host.docker.internal:host-gateway"
    volumes:
      - mongo2_data:/data/db
      - mongo2_config:/data/configdb
  mongo3:
    image: mongo:7.0
    container_name: mongo3
    command: ["--replSet", "rs0", "--bind_ip_all", "--port", "27019"]
    ports:
      - "27019:27019"
    extra_hosts:
      - "host.docker.internal:host-gateway"
    volumes:
      - mongo3_data:/data/db
      - mongo3_config:/data/configdb
volumes:
  mongo1_data:
  mongo2_data:
  mongo3_data:
  mongo1_config:
  mongo2_config:
  mongo3_config:
```

Після запуску контейнерів, виконано команду для ініціалізації:

```
test> rs.initiate({
...   _id: "rs0",
...   members: [
...     { _id: 0, host: "host.docker.internal:27017" },
...     { _id: 1, host: "host.docker.internal:27018" },
...     { _id: 2, host: "host.docker.internal:27019" }
...   ]
... })
{ ok: 1 }
```

Рис. 1: Ініціалізація кластеру

Як видно з логу на рисунку 2, вибори проведені і Primary існує.

2.2 Прості записи

Виконання простих записів з різними умовами роботи нодів MongoDB виконує наступний код:

Лістинг 2: simple_write.py

```
from pymongo import MongoClient, WriteConcern
from pymongo.read_concern import ReadConcern
from pymongo.errors import PyMongoError
import datetime
import time

uri = "mongodb://localhost:27017,localhost:27018,localhost:27019/?
    replicaSet=rs0"
client = MongoClient(uri)

db = client.get_database("testdb")

collection = db.get_collection(
    "test",
    write_concern=WriteConcern(w=3, wtimeout=10000)
)

doc = {
    "msg": "write concern 3 test",
    "ts": datetime.datetime.now()
}

print("Attempting insert")

try:
    result = collection.insert_one(doc)
    print("Write acknowledged:", result.inserted_id)
    inserted_id = result.inserted_id

except PyMongoError as e:
    print("WRITE FAILED:")
    print(type(e), str(e))

print("Attempting read")

collection_majority = db.get_collection(
    "test",
    read_concern=ReadConcern("majority")
)
time.sleep(1)

try:
    read_doc = collection_majority.find_one({"_id": inserted_id})
    print("Read result:", read_doc)

except PyMongoError as e:
    print("READ FAILED:")
    print(type(e), str(e))
```

```
48
49 print("Done.")
50 input("Press Enter to exit...")
```

Для повністю працюючого кластера код виводить:

```
Attempting insert
Write acknowledged: 693864cfa44a957b71adf265
Attempting read
Read result: {'_id': ObjectId('693864cfa44a957b71adf265'),
 'msg': 'write concern 3 test',
 'ts': datetime.datetime(2025, 12, 9, 20, 5, 3, 473000)}
Done.
```

Для одного вимкненого Secondary код вічно очікував. Після увімкнення ноди:

```
Attempting insert
Write acknowledged: 693864ebf413918fc4237cd8
Attempting read
Read result: {'_id': ObjectId('693864ebf413918fc4237cd8'),
 'msg': 'write concern 3 test',
 'ts': datetime.datetime(2025, 12, 9, 20, 5, 31, 380000)}
Done.
```

Скінченний таймаут для вимкненого Secondary:

```
Attempting insert
WRITE FAILED:
<class 'pymongo.errors.WTimeoutError'> waiting for replication timed out,
full error: {'code': 64, 'codeName': 'WriteConcernFailed',
'errmsg': 'waiting for replication timed out',
'errInfo': {'wtimeout': True, 'writeConcern': {'w': 3, 'wtimeout': 10000, 'provenance': 'cli
```

Скінченний таймаут та увімкнення Secondary під час нього:

```
Attempting insert
Write acknowledged: 693865b51c6e7fa3b7764217
Attempting read
Read result: {'_id': ObjectId('693865b51c6e7fa3b7764217'),
 'msg': 'write concern 3 test',
 'ts': datetime.datetime(2025, 12, 9, 20, 8, 53, 70000)}
```

Продемонструвати вибори не вийшло, оскільки працюючі ноди не дозволили підключення через термінал до mongosh після вимкнення Primary ноди. Довгий процес дебагу не виявив помилок, які могли б викликати таку поведінку :/

2.3 Реалізація клієнта

Клієнт запускається пайтон-скриптом, в якому в залежності від завдання виставляється відповідний `writeConcern`.

Лістинг 3: worker.py

```
import threading
import time
from tqdm import tqdm
from pymongo import MongoClient, ReturnDocument, WriteConcern

MONGO_HOST = "127.0.0.1"
MONGO_PORTS = [27017, 27018, 27019]
REPLICA_SET = "rs0"
DB_NAME = "HLS"
COLLECTION_NAME = "user_counter"

hosts = ",".join(f"{MONGO_HOST}:{port}" for port in MONGO_PORTS)
MONGO_URI = f"mongodb://{hosts}/?replicaSet={REPLICA_SET}&
    retryWrites=true&w=majority"

CALLS_PER_CLIENT = 10_000
NUM_CLIENTS = 10

print_lock = threading.Lock()

def get_client():
    return MongoClient(MONGO_URI, serverSelectionTimeoutMS=10000)

def init_db():
    client = get_client()
    db = client[DB_NAME]
    coll = db[COLLECTION_NAME]

    coll.delete_many({})
    coll.insert_one({
        "user_id": 1,
        "counter": 0,
        "version": 0
    })
    print("MongoDB initialized: counter reset to 0")
    client.close()

def worker(client_id):
    client = get_client()
    coll = client[DB_NAME][COLLECTION_NAME].with_options(
        write_concern=WriteConcern(w="majority", wtimeout=5000)
    )

    bar = tqdm(total=CALLS_PER_CLIENT,
               position=client_id,
               leave=False)

    for _ in range(CALLS_PER_CLIENT):
        coll.find_one_and_update(
            {"user_id": 1},
            {"$inc": {"counter": 1}},
```

```
            upsert=True,
            return_document=ReturnDocument.AFTER,
        )
        bar.update(1)
    with print_lock:
        bar.close()
    client.close()

def main():
    print("MongoDB In-Place Update with Optimistic Concurrency Control (version field)")
    print("Initializing DB...")
    init_db()
    time.sleep(2)

    client = get_client()
    doc = client[DB_NAME][COLLECTION_NAME].find_one({"user_id": 1})
    print(f"Initial counter: {doc['counter']}")
    client.close()

    print(f"Starting {NUM_CLIENTS} clients with {CALLS_PER_CLIENT:,} increments each...")
    start_time = time.time()

    threads = []
    for i in range(NUM_CLIENTS):
        t = threading.Thread(target=worker, args=(i,))
        t.start()
        threads.append(t)

    for t in threads:
        t.join()

    end_time = time.time()
    elapsed = end_time - start_time

    client = get_client()
    final_doc = client[DB_NAME][COLLECTION_NAME].find_one({"user_id": 1})
    final_counter = final_doc["counter"]
    client.close()

    total_calls = CALLS_PER_CLIENT * NUM_CLIENTS
    throughput = total_calls / elapsed

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

if __name__ == "__main__":
    main()
    input("\nPress Enter to exit...")
```

2.4 WC = 1

Результати тестування:
Час роботи: 26.37 секунд
Пропускна здатність: 3792.65 запитів/с
Кінцеве значення лічильника: 100к

2.5 WC = 'majority'

Результати тестування:
Час роботи: 76.2 секунди
Пропускна здатність: 1312.37 запитів/с
Кінцеве значення лічильника: 100к

2.6 WC = 1 та падіння Primary

Результати тестування:
Час роботи: 23.05 секунд
Пропускна здатність: 4338.28 запитів/с
Кінцеве значення лічильника: 100к

2.7 WC = 'majority' та падіння Primary

Результати тестування:
Час роботи: 59.51 секунд
Пропускна здатність: 1680.42 запитів/с
Кінцеве значення лічильника: 100к

3 Результати

Варіант	Час роботи	Пропускна здатність	Значення лічильника
1	26.37	3792.65	100к
majority	76.2	1312.37	100к
1 та падіння	23.05	4338.28	100к
majority та падіння	59.51	1680.42	100k

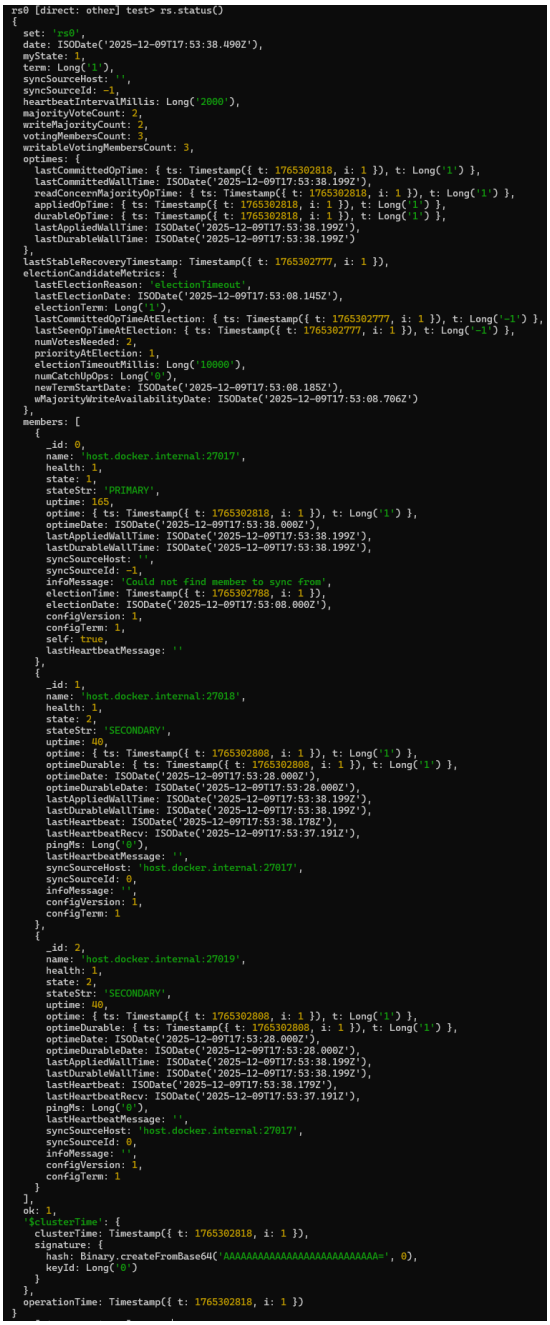

Рис. 2: Лог існування Primary