

# Проектування високонавантажених систем.

## Лабораторна 4, звіт

Михайло Голуб

10 грудня 2025 р.

### 1 Завдання лабораторної роботи

#### 1.1 Налаштування реплікації

1. Налаштовувати 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 рівнем 3 та нескінченнім таймаутом. Спробувати під час таймаута включити відключену ноду
3. Аналогічно попередньому пункту, але задати скінченний таймаут та дочекатись його закінчення. Перевірити чи данні записалися 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 інкрементацію каунтеру лайків по 10\_000 на кожного клієнта з різними опціями взаємодії з MongoDB.

- Вказавши у параметрах `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 параметрах `findOneAndUpdate writeConcern = majority` (це буде означати, що Primary чекає поки значення запишеться 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 запустіть 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 запустіть 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 ралтового 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

```

1 version: "3.8"
2 services:
3   mongo1:
4     image: mongo:7.0
5     container_name: mongo1
6     command: ["--replSet", "rs0", "--bind_ip_all", "--port", "27017"]
7     ports:
8       - "27017:27017"
9     extra_hosts:
10      - "host.docker.internal:host-gateway"
11     volumes:
12       - mongo1_data:/data/db
13       - mongo1_config:/data/configdb

```

```

14 mongo2:
15   image: mongo:7.0
16   container_name: mongo2
17   command: ["--replSet", "rs0", "--bind_ip_all", "--port", "27018"]
18   ports:
19     - "27018:27018"
20   extra_hosts:
21     - "host.docker.internal:host-gateway"
22   volumes:
23     - mongo2_data:/data/db
24     - mongo2_config:/data/configdb
25 mongo3:
26   image: mongo:7.0
27   container_name: mongo3
28   command: ["--replSet", "rs0", "--bind_ip_all", "--port", "27019"]
29   ports:
30     - "27019:27019"
31   extra_hosts:
32     - "host.docker.internal:host-gateway"
33   volumes:
34     - mongo3_data:/data/db
35     - mongo3_config:/data/configdb
36 volumes:
37   mongo1_data:
38   mongo2_data:
39   mongo3_data:
40   mongo1_config:
41   mongo2_config:
42   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

```
1 from pymongo import MongoClient, WriteConcern
2 from pymongo.read_concern import ReadConcern
3 from pymongo.errors import PyMongoError
4 import datetime
5 import time
6
7 uri = "mongodb://localhost:27017,localhost:27018,localhost:27019/?"
8     replicaSet=rs0"
9 client = MongoClient(uri)
10 db = client.get_database("testdb")
11
12 collection = db.get_collection(
13     "test",
14     write_concern=WriteConcern(w=3, wtimeout=10000)
15 )
16
17 doc = {
18     "msg": "write concern 3 test",
19     "ts": datetime.datetime.now()
20 }
21
22 print("Attempting insert")
23
24 try:
25     result = collection.insert_one(doc)
26     print("Write acknowledged:", result.inserted_id)
27     inserted_id = result.inserted_id
28
29 except PyMongoError as e:
30     print("WRITE FAILED:")
31     print(type(e), str(e))
32
33 print("Attempting read")
34
35 collection_majority = db.get_collection(
36     "test",
37     read_concern=ReadConcern("majority")
38 )
39 time.sleep(1)
40
41 try:
42     read_doc = collection_majority.find_one({"_id": inserted_id})
43     print("Read result:", read_doc)
44
45 except PyMongoError as e:
46     print("READ FAILED:")
47     print(type(e), str(e))
```

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

Для повністю працюочого 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 працюочі 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

```
 1 import threading
 2 import time
 3 from tqdm import tqdm
 4 from pymongo import MongoClient, ReturnDocument, WriteConcern
 5
 6 MONGO_HOST = "127.0.0.1"
 7 MONGO_PORTS = [27017, 27018, 27019]
 8 REPLICA_SET = "rs0"
 9 DB_NAME = "HLS"
10 COLLECTION_NAME = "user_counter"
11
12 hosts = ",".join(f"{MONGO_HOST}:{port}" for port in MONGO_PORTS)
13 MONGO_URI = f"mongodb://{hosts}/?replicaSet={REPLICA_SET}&
    retryWrites=true&w=majority"
14
15 CALLS_PER_CLIENT = 10_000
16 NUM_CLIENTS = 10
17
18 print_lock = threading.Lock()
19
20 def get_client():
21     return MongoClient(MONGO_URI, serverSelectionTimeoutMS=10000)
22
23 def init_db():
24     client = get_client()
25     db = client[DB_NAME]
26     coll = db[COLLECTION_NAME]
27
28     coll.delete_many({})
29     coll.insert_one({
30         "user_id": 1,
31         "counter": 0,
32         "version": 0
33     })
34     print("MongoDB initialized: counter reset to 0")
35     client.close()
36
37 def worker(client_id):
38     client = get_client()
39     coll = client[DB_NAME][COLLECTION_NAME].with_options(
40         write_concern=WriteConcern(w="majority", wtimeout=5000)
41     )
42
43     bar = tqdm(total=CALLS_PER_CLIENT,
44                 position=client_id,
45                 leave=False)
46
47     for _ in range(CALLS_PER_CLIENT):
48         coll.find_one_and_update(
49             {"user_id": 1},
50             {"$inc": {"counter": 1}},
```

```

51         upsert=True,
52         return_document=ReturnDocument.AFTER,
53     )
54     bar.update(1)
55     with print_lock:
56         bar.close()
57     client.close()
58
59 def main():
60     print("MongoDB In-Place Update with Optimistic Concurrency
61           Control (version field)")
62     print("Initializing DB...")
63     init_db()
64     time.sleep(2)
65
66     client = get_client()
67     doc = client[DB_NAME][COLLECTION_NAME].find_one({"user_id": 1})
68     print(f"Initial counter: {doc['counter']}")"
69     client.close()
70
71     print(f"Starting {NUM_CLIENTS} clients with {CALLS_PER_CLIENT}
72           :,) increments each...")
73     start_time = time.time()
74
75     threads = []
76     for i in range(NUM_CLIENTS):
77         t = threading.Thread(target=worker, args=(i,))
78         t.start()
79         threads.append(t)
80
81     for t in threads:
82         t.join()
83
84     end_time = time.time()
85     elapsed = end_time - start_time
86
87     client = get_client()
88     final_doc = client[DB_NAME][COLLECTION_NAME].find_one({"user_id
89           ": 1})
90     final_counter = final_doc["counter"]
91     client.close()
92
93     total_calls = CALLS_PER_CLIENT * NUM_CLIENTS
94     throughput = total_calls / elapsed
95
96     print(f"Final count: {final_counter}")
97     print(f"Total time: {elapsed:.2f} s")
98     print(f"Throughput: {throughput:.2f} requests/sec")
99
100 if __name__ == "__main__":
101     main()
102     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

```

rs0 [direct: other] test> rs.status()
{
  set: "rs0",
  date: ISODate("2025-12-09T17:53:38.490Z"),
  myState: 1,
  state: 1,
  syncSourceHost: "",
  syncSourceId: -1,
  heartbeatIntervalMillis: Long(2000),
  majorityVoteCount: 3,
  votingReplicaCount: 2,
  votingMembersCount: 3,
  writableVotingMembersCount: 3,
  optimes: [
    lastCommittedOptime: { ts: Timestamp({ $: 1765302818, i: 1 }), t: Long(1) },
    lastCommittedWallTime: ISODate("2025-12-09T17:53:38.199Z"),
    syncDurableMajorityOptime: { ts: Timestamp({ t: 1765302818, i: 1 }), t: Long(1) },
    appliedOptime: { ts: Timestamp({ t: 1765302818, i: 1 }), t: Long(1) },
    durableOptime: { ts: Timestamp({ t: 1765302818, i: 1 }), t: Long(1) },
    lastAppliedWallTime: ISODate("2025-12-09T17:53:38.199Z"),
    lastDurableWallTime: ISODate("2025-12-09T17:53:38.199Z"),
    lastStableRecoveryTimestamp: Timestamp({ t: 1765302777, i: 1 }),
    electionCandidateMetrics: {
      lastElectionReason: "electionTimeout",
      lastElectionDate: ISODate("2025-12-09T17:53:08.145Z"),
      lastElectedAt: ISODate("2025-12-09T17:53:08.145Z"),
      lastCommittedOptimeAtElection: { ts: Timestamp({ t: 1765302777, i: 1 }), t: Long(-1) },
      lastSeenOptimeAtElection: { ts: Timestamp({ t: 1765302777, i: 1 }), t: Long(-1) },
      numVotesNeeded: 2,
      priorityAtElection: 1,
      staleAtElection: Long(10000),
      maxCatchUpOps: Long(0),
      newTermStartDate: ISODate("2025-12-09T17:53:08.185Z"),
      wMajorityWriteAvailabilityDate: ISODate("2025-12-09T17:53:08.706Z")
    },
    members: [
      {
        _id: 0,
        name: "host.docker.internal:27017",
        health: 1,
        state: 1,
        stateStr: "PRIMARY",
        uptime: 165,
        optime: { ts: Timestamp({ t: 1765302818, i: 1 }), t: Long(1) },
        optimeDate: ISODate("2025-12-09T17:53:38.000Z"),
        lastAppliedWallTime: ISODate("2025-12-09T17:53:38.199Z"),
        lastDurableWallTime: ISODate("2025-12-09T17:53:38.199Z"),
        syncSourceHost: "",
        syncSourceId: -1,
        infoMessage: "Could not find member to sync from",
        electionTime: Timestamp({ t: 1765302788, i: 1 }),
        electionDate: ISODate("2025-12-09T17:53:08.000Z"),
        configVersion: 1,
        configTerm: 1,
        self: true,
        lastHeartbeatMessage: ""
      },
      {
        _id: 1,
        name: "host.docker.internal:27018",
        health: 1,
        state: 2,
        stateStr: "SECONDARY",
        uptime: 40,
        optime: { ts: Timestamp({ t: 1765302808, i: 1 }), t: Long(1) },
        optimeDate: ISODate("2025-12-09T17:53:28.000Z"),
        optimeLastApplied: ISODate("2025-12-09T17:53:28.000Z"),
        lastAppliedWallTime: ISODate("2025-12-09T17:53:38.199Z"),
        lastDurableWallTime: ISODate("2025-12-09T17:53:38.199Z"),
        lastHeartbeat: ISODate("2025-12-09T17:53:38.178Z"),
        lastHeartbeatRecv: ISODate("2025-12-09T17:53:37.191Z"),
        pingMs: Long(0),
        lastHeartbeatMessage: "",
        syncSourceHost: "host.docker.internal:27017",
        syncSourceId: 0,
        infoMessage: "",
        configVersion: 1,
        configTerm: 1
      },
      {
        _id: 2,
        name: "host.docker.internal:27019",
        health: 1,
        state: 2,
        stateStr: "SECONDARY",
        uptime: 40,
        optime: { ts: Timestamp({ t: 1765302809, i: 1 }), t: Long(1) },
        optimeDate: ISODate("2025-12-09T17:53:28.000Z"),
        optimeLastApplied: ISODate("2025-12-09T17:53:28.000Z"),
        lastAppliedWallTime: ISODate("2025-12-09T17:53:38.000Z"),
        lastDurableWallTime: ISODate("2025-12-09T17:53:38.199Z"),
        lastHeartbeat: ISODate("2025-12-09T17:53:38.179Z"),
        lastHeartbeatRecv: ISODate("2025-12-09T17:53:37.191Z"),
        pingMs: Long(0),
        lastHeartbeatMessage: "",
        syncSourceHost: "host.docker.internal:27017",
        syncSourceId: 0,
        infoMessage: "",
        configVersion: 1,
        configTerm: 1
      }
    ],
    ok: 1,
    $clusterTime: {
      clusterTime: Timestamp({ t: 1765302818, i: 1 }),
      signature: {
        hash: Binary.createFromBase64("AAAAAAAAAAAAAAA=AAA=AAA=AAA=AAA="),
        keyId: Long(0)
      }
    },
    operationTime: Timestamp({ t: 1765302818, i: 1 })
}

```

Рис. 2: Лог існування Primary