Task 4 - Налаштування реплікації та перевірка відмовостійкості MongoDB

I 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)

Для створення кластеру використовувався docker compose файл з неюбхідними параметрами. Після запуску кластру на одній з нод потрібно виконати наступне:

```
test> `({
...   _id: "rs0",
...   members: [
...     { _id: 0, host: "mongo1:27017" },
...     { _id: 1, host: "mongo2:27018" },
...     { _id: 2, host: "mongo3:27019" }
...   ]
... })
{ ok: 1 }
```

Перевіримо результат виконання:

```
rs0 [direct: other] test> rs.status()
{
  set: 'rs0',
  date: ISODate('2024-12-31T09:42:00.854Z'),
  myState: 2,
  term: Long('0'),
  syncSourceHost: '',
  syncSourceId: -1,
  heartbeatIntervalMillis: Long('2000'),
  majorityVoteCount: 2,
```

```
  writeMajorityCount: 2,
  votingMembersCount: 3,
  writableVotingMembersCount: 3,
...
members: [
    {
      _id: 0,
      name: 'mongo1:27017',
      health: 1,
      state: 1,
      stateStr: 'PRIMARY',
      uptime: 141,
      ...
    },
    {
      _id: 1,
      name: 'mongo2:27018',
      health: 1,
      state: 2,
      stateStr: 'SECONDARY',
      uptime: 109,
      ...
    },
    {
      _id: 2,
      name: 'mongo3:27019',
      health: 1,
      state: 2,
      stateStr: 'SECONDARY',
      uptime: 109,
      ...
    }
  ],
  ok: 1
```

Як можна побачити з виводу, перша нода була обрана як PRIMARY.

2. Спробувати зробити запис з однією відключеною нодою та write concern рівнім 3 та нескінченім таймаутом. Спробувати під час таймаута включити відключену ноду

Спочатку за допомогою `sudo docker stop mongo3` зупиняємо одну з нод та записуємо дані:

```
rs0 [direct: primary] test> db.testCollection.insertOne(
...   { key: "value1" },
...   { writeConcern: { w: 3, wtimeout: 0 } } // Таймаут = 0 означає, що очікування нескінченне
... );
{
  acknowledged: true,
  insertedId: ObjectId('6773bf01e307599282fe6911')
}
```

Після цього включимо ноду 3 та перевіримо присутність даних на ній:

```
sudo docker exec -it mongo3 mongosh

rs0 [direct: secondary] test> db.testCollection.find({ key: "value1" });
[ { _id: ObjectId('6773bf01e307599282fe6911'), key: 'value1' } ]
```

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”

```
rs0 [direct: primary] test> db.testCollection.insertOne(
...   { key: "value2" },
...   { writeConcern: { w: 3, wtimeout: 5000 } } // Таймаут = 5 секунд
```

```
... );
Uncaught:
MongoWriteConcernError[WriteConcernFailed]: waiting for replication timed
out
Additional information: {
  wtimeout: true,
  writeConcern: { w: 3, wtimeout: 5000, provenance: 'clientSupplied' }
}
Result: {
  n: 1,
  electionId: ObjectId('7fffffff0000000000000001'),
  opTime: { ts: Timestamp({ t: 1735639027, i: 1 }), t: Long('1') },
  writeConcernError: {
    code: 64,
    codeName: 'WriteConcernFailed',
    errmsg: 'waiting for replication timed out',
    errInfo: {
      wtimeout: true,
      writeConcern: { w: 3, wtimeout: 5000, provenance: 'clientSupplied' }
    }
  },
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1735639027, i: 1 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1735639027, i: 1 })
}
```

Як можна побачити, після закінчення таймауту, дані записались на ті ноди, які були присутні. Тепер повернем ноду 3 та перевіимо дані.

```
rs0 [direct: primary] test> db.getSiblingDB("test").runCommand({
...   find: "testCollection",
...   readConcern: { level: "majority" }
... });
{
  cursor: {
    firstBatch: [
      { _id: ObjectId('6773bf01e307599282fe6911'), key: 'value1' },
      { _id: ObjectId('6773bff317c3f331c1fe6911'), key: 'value2' }
    ],
    id: Long('0'),
    ns: 'test.testCollection'
  },
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1735639255, i: 1 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1735639255, i: 1 })
}
```

4. Продемонстрував 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

Зупиняємо ноду 1 за допомогою команди `sudo docker stop mongo1` та перевіряємо ноду 2:

```
rs0 [direct: secondary] test> rs.isMaster()
```

```
{
  topologyVersion: {
    processId: ObjectId('6773bc43436f1d3f860bacbb'),
    counter: Long('5')
  },
  hosts: [ 'mongo1:27017', 'mongo2:27018', 'mongo3:27019' ],
  setName: 'rs0',
  setVersion: 1,
  ismaster: false,
  secondary: true,
  primary: 'mongo3:27019',
  me: 'mongo2:27017',
  lastWrite: {
    opTime: { ts: Timestamp({ t: 1735639923, i: 1 }), t: Long('3') },
    lastWriteDate: ISODate('2024-12-31T10:12:03.000Z'),
    majorityOpTime: { ts: Timestamp({ t: 1735639923, i: 1 }), t:
Long('3') },
    majorityWriteDate: ISODate('2024-12-31T10:12:03.000Z')
  },
  maxBsonObjectSize: 16777216,
  maxMessageSizeBytes: 48000000,
  maxWriteBatchSize: 100000,
  localTime: ISODate('2024-12-31T10:12:04.865Z'),
  logicalSessionTimeoutMinutes: 30,
  connectionId: 41,
  minWireVersion: 0,
  maxWireVersion: 17,
  readOnly: false,
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1735639923, i: 1 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
```

```
  },
  operationTime: Timestamp({ t: 1735639923, i: 1 }),
  isWritablePrimary: false
}
```

Як можна побачити тут:

ismaster: false

Перевіримо статус кластеру:

```
rs0 [direct: secondary] test> rs.status()
{
  set: 'rs0',
  date: ISODate('2024-12-31T10:12:14.466Z'),
  myState: 2,
  term: Long('3'),
  syncSourceHost: 'mongo3:27019',
  syncSourceId: 2,
  heartbeatIntervalMillis: Long('2000'),
  ...
  },
  members: [
    {
      _id: 0,
      name: 'mongo1:27017',
      health: 0,
      state: 8,
      stateStr: '(not reachable/healthy)',
      ...
    },
    {
      _id: 1,
      name: 'mongo2:27018',
      health: 1,
```

```
      state: 2,
      stateStr: 'SECONDARY',
      uptime: 1851,
      optime: { ts: Timestamp({ t: 1735639933, i: 1 }), t: Long('3') },
      optimeDate: ISODate('2024-12-31T10:12:13.000Z'),
      lastAppliedWallTime: ISODate('2024-12-31T10:12:13.020Z'),
      lastDurableWallTime: ISODate('2024-12-31T10:12:13.020Z'),
      syncSourceHost: 'mongo3:27019',
      syncSourceId: 2,
      infoMessage: '',
      configVersion: 1,
      configTerm: 3,
      self: true,
      lastHeartbeatMessage: ''
    },
    {
      _id: 2,
      name: 'mongo3:27017',
      health: 1,
      state: 1,
      stateStr: 'PRIMARY',
      uptime: 19,
      ...
    }
  ],
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1735639933, i: 1 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1735639933, i: 1 })
}
```

Як можна побачити, головною стала нода 3.

Додамо дані на кластер та увімкнемо ноду 1:

```
rs0 [direct: primary] test> db.testCollection.insertOne({ key:
"newDataDuringDowntime" });
{
  acknowledged: true,
  insertedId: ObjectId('6773c53e6b9f878987fe6911')
}
rs0 [direct: primary] test> db.testCollection.find({ key:
"newDataDuringDowntime" });
[
  {
    _id: ObjectId('6773c53e6b9f878987fe6911'),
    key: 'newDataDuringDowntime'
  }
]
```

Після включення ноди 1 отримуємо дані:

```
rs0 [direct: secondary] test> db.testCollection.find({ key:
"newDataDuringDowntime" });
[
  {
    _id: ObjectId('6773c53e6b9f878987fe6911'),
    key: 'newDataDuringDowntime'
  }
]
```

II 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.

Для того, щоб не було lost updates, для оновлення каунтера необхідно використовувати функцію findOneAndUpdate()

Приклад використання:

```
db.grades.findOneAndUpdate(
   { "name" : "R. Stiles" },
   { $inc: { "points" : 5 } }
)
```

Для виконання цього завдання було написано код на мові програмування python та створені відповідні записи у базі даних:

```
rs0 [direct: secondary] test> db.createCollection("users")
{ ok: 1 }
rs0 [direct: primary] test> db.users.insertOne({
... user_name: "User1",
... likes_count: 0
... })
{
  acknowledged: true,
  insertedId: ObjectId('6773d81cbec2fb9c7cfe6911')
}
```

5. 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 - 100К

Частина кода для виконання завдання:

```python
# Функція для інкрементації каунтера з WriteConcern = 1
def increment_likes_with_writeConcern(user_name, increment_value, write_concern_level):
    write_concern = WriteConcern(write_concern_level)
    for _ in range(increment_value):
        collection.with_options(write_concern=write_concern).find_one_and_update(
            {"user_name": user_name},
            {"$inc": {"likes_count": 1}},
            return_document=True
        )

# Скидання каунтера перед запуском
collection.find_one_and_update(
    {"user_name": user_name},
    {"$set": {"likes_count": 0}},
    upsert=False
)

print("Starting...")
threads = []
start_time = time.time()

# Запуск потоків
for _ in range(num_threads):
    thread = threading.Thread(target=increment_likes_with_writeConcern, args=(user_name, increment_value, 1))
    threads.append(thread)
    thread.start()
```

Результат виконання:

```
Starting...
Time: 149.75
Expected: 100000
Real: 100000
```

6. 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 — 100К

Частина кода для виконання завдання:

```python
# Функція для інкрементації каунтера з WriteConcern = majority
def increment_likes_with_writeConcern(user_name, increment_value, write_concern_level):
    write_concern = WriteConcern(w=write_concern_level)
    for _ in range(increment_value):
        collection.with_options(write_concern=write_concern).find_one_and_update(
            {"user_name": user_name},
            {"$inc": {"likes_count": 1}},
            return_document=True
        )

# Скидання каунтера перед запуском
collection.find_one_and_update(
    {"user_name": user_name},
    {"$set": {"likes_count": 0}},
    upsert=False
)

print("Starting...")
threads = []
start_time = time.time()

# Запуск потоків
for _ in range(num_threads):
    thread = threading.Thread(target=increment_likes_with_writeConcern, args=(user_name, increment_value, "majority"))
    threads.append(thread)
    thread.start()
```

Результат виконання:

```
Starting...
Time: 209.57
Expected: 100000
Real: 100000
```

7. 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.

Додатково змінемо параметри клієнта:

```
client = MongoClient('mongodb://mongo1:27017,mongo2:27018,mongo3:27019/?replicaSet=rs0')
```

Після запуску роботи коду, виконується наступна команда:

```
sudo docker stop mongo1
[sudo] password for user:
mongo1
```

Записи у базу не зупинились. Результат виконання:

```
Time: 88.67
Expected: 100000
Real: 71955
```

8. 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.

Після запуску роботи коду, виконується наступна команда:

```
sudo docker stop mongo1
[sudo] password for user:
mongo1
```

Результат виконання:

```
Time: 134.54
Expected: 100000
Real: 100000
```

Як і очікувалось, втрати даних не було.

Лістинг

```
import threading
import time
```

```
from pymongo import MongoClient
from pymongo import WriteConcern

# Підключення до MongoDB
client = MongoClient('mongodb://localhost:27017')
db = client.test
collection = db.users

# Ініціалізація даних
increment_value = 10000
num_threads = 10
user_name = "User1"

# Ініціалізуємо колекцію (якщо запису немає, створюємо його)
if not collection.find_one({"user_name": user_name}):
collection.insert_one({"user_name": user_name, "likes_count": 0})

# Функція для інкрементації каунтера з WriteConcern = 1
def increment_likes_with_writeConcern(user_name, increment_value,
write_concern_level):
write_concern = WriteConcern(write_concern_level)
for _ in range(increment_value):
collection.with_options(write_concern=write_concern).find_one_and_update(
{"user_name": user_name},
{"$inc": {"likes_count": 1}},
return_document=True
)

# Скидання каунтера перед запуском
collection.find_one_and_update(
{"user_name": user_name},
{"$set": {"likes_count": 0}},
upsert=False
)
```

```
print("Starting...")
threads = []
start_time = time.time()

# Запуск потоків
for _ in range(num_threads):
thread = threading.Thread(target=increment_likes_with_writeConcern,
args=(user_name, increment_value, 1))
threads.append(thread)
thread.start()

for thread in threads:
thread.join()

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

# Перевірка результату
user = collection.find_one({"user_name": user_name})
expected_likes = num_threads * increment_value
actual_likes = user["likes_count"]
print(f"Time: {execution_time:.2f}")
print(f"Expected: {expected_likes}")
print(f"Real: {actual_likes}")

# Скидання каунтера після тесту
collection.find_one_and_update(
{"user_name": user_name},
{"$set": {"likes_count": 0}},
upsert=False
)
```