Київський національний університет імені Тараса
Шевченка
Факультет комп’ютерних наук та кібернетики

ЛАБОРАТОРНА РОБОТА №2

З курсу “Сучасні технології баз даних”

Виконала:
Студентка 3 курсу, групи МІ-3
Спеціальності “Комп’ютерні науки”
Кондратюк Вікторія Василівна

Київ - 2023

Предметна область та обрані технології

Для порівняння SQL та NoSQL баз даних оберемо PostgreSQL та MongoDB на системі криптовалютної біржі.

Доречність і мотивація використання MongoDB обумовлені наступним:

1. Гнучкість схеми:

   Дозволяє зберігати дані без фіксованої схеми, що особливо корисно у світі криптовалют, де формати даних можуть змінюватися та розширюватися.

2. Швидкість та масштабованість:

   Володіє добре оптимізованим механізмом для роботи з великою кількістю даних та швидкістю їх опрацювання, що може бути важливо для системи криптовалютної біржі з великим обсягом торгів.

3. Підтримка запитів:

   Дозволяє виконувати різноманітні запити, включаючи складні запити до вкладених структур даних, що може бути корисним для аналізу та візуалізації фінансових даних.

4. Географічна реплікація:

   Надає можливість реплікації даних на різних вузлах у різних географічних областях, що може підвищити доступність та надійність системи.

5. Гнучкість розширення:

   Дозволяє легко розширювати базу даних за допомогою горизонтального масштабування (шардування), що може бути важливо для вирішення проблем зростання обсягу даних.

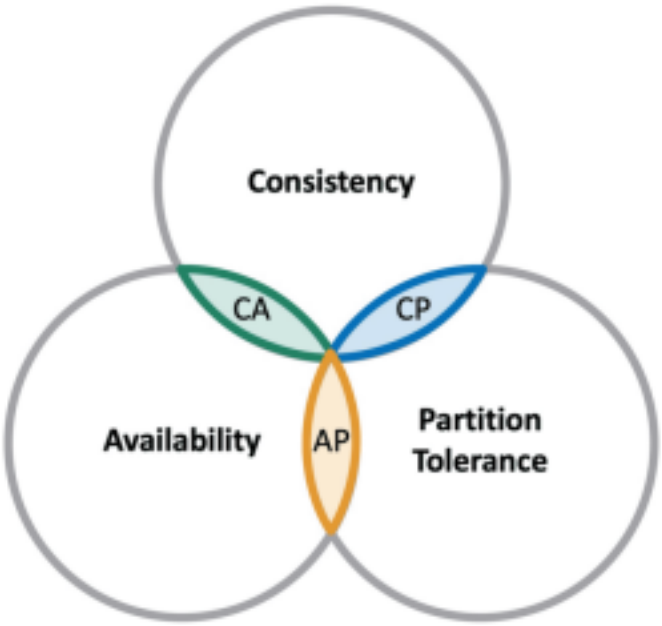
Оркестрація інфраструктури

Використаємо контейнеризовані версії баз даних за допомогою технології docker.

Конфігурація баз даних

Важливо взяти до уваги той факт, що обидві бази даних мають конфіг, що дозволяє робити екстенсивні зміни в їх взаємодії. Серед цих змін присутні також такі, що можуть впливати на швидкодію запитів до цих баз: кешування, кількість потоків, WAL та інші.

CAP теорема PostgreSQL vs MongoDB



PostgreSQL зазвичай класифікують як CA, в той час як MongoDB як AP. Очевидно, що в них присутні механізми для реалізації третьої частини в певній мірі, але фокус на дві інші.

Відповідно до їх класифікації, можна зрозуміти, що доступність, стійкість до розділення та eventual consistency MongoDB гарно

накладаються на систему транзакцій криптовалютної біржі.

PostgreSQL бенчмарк

Задамо наступним чином схему:

```
CREATE TABLE users (
    user_id SERIAL PRIMARY KEY,
    username VARCHAR(255),
    email VARCHAR(255),
    password_hash VARCHAR(255),
    wallet_balance DECIMAL(18, 2)
);

CREATE TABLE assets (
    asset_id SERIAL PRIMARY KEY,
    asset_name VARCHAR(255),
    symbol VARCHAR(10)
);

CREATE TABLE orders (
    order_id SERIAL PRIMARY KEY,
    user_id INT REFERENCES users(user_id),
    asset_id INT REFERENCES assets(asset_id),
    order_type VARCHAR(4),
    quantity DECIMAL(18, 8),
    price DECIMAL(18, 2),
    status VARCHAR(20),
    timestamp TIMESTAMP
);
```

Згенеруємо по 1000 записів та отримаємо метрики за допомогою EXPLAIN ANALYZE:

```sql
-- Insert 1000 users
EXPLAIN ANALYZE
INSERT INTO users (username, email, password_hash, wallet_balance)
SELECT 'user'||i||'example.com', 'user'||i||'@example.com', 'hashed_password', random()*1000
FROM generate_series(1,1000) AS i;

-- Insert 1000 assets
EXPLAIN ANALYZE
INSERT INTO assets (asset_name, symbol)
SELECT CONCAT('asset', i), CONCAT('SYM', i)
FROM generate_series(1,1000) AS i;

-- Insert 1000 orders
EXPLAIN ANALYZE
INSERT INTO orders (user_id, asset_id, order_type, quantity, price, status, timestamp)
SELECT
    floor(random()*1000)+1,
    floor(random()*1000)+1,
    CASE
        WHEN random() < 0.33 THEN 'BUY'
        WHEN random() >= 0.33 AND random() < 0.66 THEN 'SWAP'
        ELSE 'SELL'
    END,
    random()*10,
    random()*100,
    'OPEN',
    NOW()
FROM generate_series(1,1000);
```

Маємо відповідні результати:

```
                                                          QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------
 Insert on users  (cost=0.00..60.00 rows=0 width=0) (actual time=4.593..4.594 rows=0 loops=1)
   ->  Subquery Scan on "*SELECT*"  (cost=0.00..60.00 rows=1000 width=1572) (actual time=0.188..2.238 rows=1000 loops=1)
         ->  Function Scan on generate_series i  (cost=0.00..35.00 rows=1000 width=104) (actual time=0.078..0.792 rows=1000 loops=1)
 Planning Time: 0.213 ms
 Execution Time: 4.665 ms
(5 rows)

                                                          QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------
 Insert on assets  (cost=0.00..25.00 rows=0 width=0) (actual time=2.779..2.779 rows=0 loops=1)
   ->  Function Scan on generate_series i  (cost=0.00..25.00 rows=1000 width=558) (actual time=0.128..0.966 rows=1000 loops=1)
 Planning Time: 0.034 ms
 Execution Time: 2.811 ms
(4 rows)

                                                          QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------
 Insert on orders  (cost=0.00..92.50 rows=0 width=0) (actual time=4.730..4.732 rows=0 loops=1)
   ->  Subquery Scan on "*SELECT*"  (cost=0.00..92.50 rows=1000 width=138) (actual time=0.161..2.535 rows=1000 loops=1)
         ->  Function Scan on generate_series  (cost=0.00..57.50 rows=1000 width=104) (actual time=0.072..0.275 rows=1000 loops=1)
 Planning Time: 0.063 ms
 Trigger for constraint orders_user_id_fkey: time=15.188 calls=1000
 Trigger for constraint orders_asset_id_fkey: time=14.627 calls=1000
 Execution Time: 34.623 ms
(7 rows)
```

MongoDB бенчмарк:

Аналогічна схема:

```
const db = db.getSiblingDB('lab2');

db.createCollection("users", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["username", "email", "passwordHash", "walletBalance"],
      properties: {
        username: { bsonType: "string" },
        email: { bsonType: "string" },
        passwordHash: { bsonType: "string" },
        walletBalance: { bsonType: "decimal" }
      }
    }
  }
});

db.createCollection("assets", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["assetName", "symbol"],
      properties: {
        assetName: { bsonType: "string" },
        symbol: { bsonType: "string" }
      }
    }
  }
});

db.createCollection("orders", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["userId", "assetId", "orderType", "quantity", "price", "status", "timestamp"],
      properties: {
        userId: { bsonType: "int" },
        assetId: { bsonType: "int" },
        orderType: { bsonType: "string" },
        quantity: { bsonType: "decimal" },
        price: { bsonType: "decimal" },
        status: { bsonType: "string" },
        timestamp: { bsonType: "date" }
      }
    }
  }
});
```

Аналогічно створюємо записи і вимірюємо проміжки часу, витраченому на виконання:

```
const db = db.getSiblingDB('lab2');

// Insert 1000 users

const usersData = [];
for (let i = 1; i <= 1000; i++) {
    const user = {
        insertOne: {
            document: {
                username: 'user' + i + 'example.com',
                email: 'user' + i + '@example.com',
                passwordHash: 'hashed_password',
                walletBalance: Decimal128.fromString((Math.random() * 1000).toFixed(2)),
            }
        }
    };
    usersData.push(user);
}

const startUsersInsert = new Date();
db.users.bulkWrite(usersData);
const endUsersInsert = new Date();
const usersExecutionTime = endUsersInsert - startUsersInsert;
print(`Users insertion time: ${usersExecutionTime} ms`);
```

```
// Insert 1000 assets

const assetsData = [];
for (let i = 1; i <= 1000; i++) {
    const asset = {
        insertOne: {
            document: {
                assetName: 'asset' + i,
                symbol: 'SYM' + i,
            }
        }
    };
    assetsData.push(asset);
}

const startAssetsInsert = new Date();
db.assets.bulkWrite(assetsData);
const endAssetsInsert = new Date();
const assetsExecutionTime = endAssetsInsert - startAssetsInsert;
print(`Assets insertion time: ${assetsExecutionTime} ms`);
```

```
// Insert 1000 orders

const ordersData = [];
for (let i = 1; i <= 1000; i++) {
    const order = {
        userId: Math.floor(Math.random() * 1000) + 1,
        assetId: Math.floor(Math.random() * 1000) + 1,
        orderType: Math.random() < 0.33 ? 'BUY' : (Math.random() >= 0.33 && Math.random() < 0.66 ? 'SWAP' : 'SELL'),
        quantity: Decimal128.fromString((Math.random() * 10).toFixed(8)),
        price: Decimal128.fromString((Math.random() * 100).toFixed(2)),
        status: 'OPEN',
        timestamp: new Date()
    };

    ordersData.push(order);
}

const startOrdersInsert = new Date();
db.orders.insertMany(ordersData);
const endOrdersInsert = new Date();
const ordersExecutionTime = endOrdersInsert - startOrdersInsert;
print(`Orders insertion time: ${ordersExecutionTime} ms`);
```

Результати:

```
Users insertion time: 40 ms
Assets insertion time: 17 ms
Orders insertion time: 35 ms
```

Висновок

Було отримано знання із використання MongoDB та порівняно імплементація однакової логіки у різних технологіях. Виявлено зручний спосіб проводити базові бенчмарки та аналіз операцій в обох базах. Вибір між PostgreSQL та MongoDB залежить від конкретних вимог проекту. Якщо необхідна гнучкість у схемі та швидкість масової вставки, MongoDB може бути кращим варіантом. З іншого боку, якщо важлива структура даних та складні операції зв'язку, PostgreSQL може виявитися більш підходящим.

Посилання на код

https://github.com/viktoriina/db-lab2