

Проектування високонавантажених систем

Лабораторна робота 5

Торгало Ігнатій ФБ-51мп

1. Розгортання Cassandra у Docker та підключення через cqlsh

Запущено контейнер з Cassandra та виконано підключення до СУБД через cqlsh.

```
(kali㉿Kali-labs)-[~/Desktop]
└─$ sudo docker ps
CONTAINER ID   IMAGE      COMMAND
STATUS          PORTS     NAMES
3d89e8dd3992   cassandra:latest "docker-entrypoint.s..."   About a minute ago
Up About a minute  7000-7001/tcp, 7199/tcp, 9160/tcp, 0.0.0.0:9042→9042/
tcp, [::]:9042→9042/tcp   cassandra-lab

(kali㉿Kali-labs)-[~/Desktop]
└─$ sudo docker exec -it cassandra-lab cqlsh
Connected to Test Cluster at 127.0.0.1:9042
[cqlsh 6.2.0 | Cassandra 5.0.6 | CQL spec 3.4.7 | Native protocol v5]
Use HELP for help.
cqlsh> █
```

Рисунок 1 — Запущений контейнер Cassandra та підключення через cqlsh

2. Створення keyspace shop та таблиці items

Створено keyspace з SimpleStrategy та replication_factor = 1. Таблиця items спроектована так, щоб швидко виконувати запити в межах категорій: category використано як partition key, а price (та id) — як clustering key для сортування за ціною. Для додаткових властивостей товару використано тип map (props), а для пошуку по вмісту map створено індекс.

```
(kali㉿Kali-labs)-[~/Desktop]
└─$ sudo docker exec -it cassandra-lab cqlsh
Connected to Test Cluster at 127.0.0.1:9042
[cqlsh 6.2.0 | Cassandra 5.0.6 | CQL spec 3.4.7 | Native protocol v5]
Use HELP for help.
cqlsh> CREATE KEYSPACE shop WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 1};
cqlsh> USE shop;
cqlsh:shop> CREATE TABLE items (
    ...     category text,
    ...     price decimal,
    ...     id uuid,
    ...     name text,
    ...     producer text,
    ...     props map<text, text>,
    ...     PRIMARY KEY ((category), price, id)
    ... ) WITH CLUSTERING ORDER BY (price ASC);
cqlsh:shop> CREATE INDEX ON items(ENTRIES(props));
```

Рисунок 2 — Створення keyspace *shop* та таблиці *items* з індексом на map

3. Наповнення таблиці *items* тестовими даними

Додано тестові товари різних категорій (Electronics, Books) з різними наборами властивостей у *props*.

```
cqlsh:shop> INSERT INTO items (category, price, id, name, producer, props) VALUES ('Electronics', 1000, uuid(), 'Laptop X', 'Dell', {'color': 'black', 'ram': '16GB'});
cqlsh:shop> INSERT INTO items (category, price, id, name, producer, props) VALUES ('Electronics', 500, uuid(), 'Phone Y', 'Apple', {'color': 'white', 'storage': '128GB'});
cqlsh:shop> INSERT INTO items (category, price, id, name, producer, props) VALUES ('Electronics', 1200, uuid(), 'Laptop Pro', 'Apple', {'color': 'grey'});
cqlsh:shop> INSERT INTO items (category, price, id, name, producer, props) VALUES ('Books', 20, uuid(), 'Cassandra Guide', 'OReilly', {'pages': '300'});
```

Рисунок 3 — Добавлення тестових даних у *items*

4. Перевірка структури *items* та вибірка товарів у категорії

Використано DESCRIBE для перегляду структури таблиці та SELECT для вибірки всіх товарів у категорії. У запитах ALLOW FILTERING не використовувалось.

```
cqlsh:shop> DESCRIBE TABLE items;

CREATE TABLE shop.items (
    category text,
    price decimal,
    id uuid,
    name text,
    producer text,
    props map<text, text>,
    PRIMARY KEY (category, price, id)
) WITH CLUSTERING ORDER BY (price ASC, id ASC)
    AND additional_write_policy = '99p'
    AND allow_auto_snapshot = true
    AND bloom_filter_fp_chance = 0.01
    AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
    AND cdc = false
    AND comment = ''
    AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy', 'max_threshold': '32', 'min_threshold': '4'}
    AND compression = {'chunk_length_in_kb': '16', 'class': 'org.apache.cassandra.io.compress.LZ4Compressor'}
    AND memtable = 'default'
    AND crc_check_chance = 1.0
    AND default_time_to_live = 0
    AND extensions = {}
    AND gc_grace_seconds = 864000
    AND incremental_backups = true
    AND max_index_interval = 2048
    AND memtable_flush_period_in_ms = 0
    AND min_index_interval = 128
    AND read_repair = 'BLOCKING'
    AND speculative_retry = '99p';

CREATE INDEX items_props_idx ON shop.items (entries(props));
```

Рисунок 4 — Результатам DESCRIBE TABLE items

```
cqlsh:shop> SELECT * FROM items WHERE category = 'Electronics';
+-----+-----+-----+-----+-----+-----+
| category | price | id               | name      | producer | props          |
+-----+-----+-----+-----+-----+-----+
| Electronics | 500  | 8624dadcd-fb13-4b78-9b79-a8d163a58cda | Phone Y   | Apple    | {"color": "white", "storage": "128GB"} |
| Electronics | 1000 | 7e2cd97c-5cc9-4358-9882-c5e42c433d59 | Laptop X  | Dell     | {"color": "black", "ram": "16GB"}        |
| Electronics | 1200 | 1df79fb2-b13d-405d-a657-8a67e0326416 | Laptop Pro | Apple    | {"color": "grey"}           |
+-----+-----+-----+-----+-----+-----+
(3 rows)
```

Рисунок 5 — Вибірка товарів у категорії Electronics (відсортовано за price)

5. Створення таблиці orders та запити до замовлень

Створено таблицю orders для зберігання замовлень користувачів. Для швидкого доступу до замовлень конкретного замовника customer_name використано як partition key, а order_time — як clustering key з сортуванням за часом (DESC). Виконано вибірку замовлень клієнта, підрахунок суми та отримання часу запису ціни через WRITETIME.

```

cqlsh:shop> CREATE TABLE orders (
...     customer_name text,
...     order_time timestamp,
...     order_id uuid,
...     items_ids list<uuid>,
...     total_price decimal,
...     PRIMARY KEY ((customer_name), order_time)
... ) WITH CLUSTERING ORDER BY (order_time DESC);
cqlsh:shop> INSERT INTO orders (customer_name, order_time, order_id, total_price) VALUES ('Ivan', toTimestamp(now()), uuid(), 1
500);
cqlsh:shop> INSERT INTO orders (customer_name, order_time, order_id, total_price) VALUES ('Ivan', '2023-01-01 10:00:00', uuid())
, 200);
cqlsh:shop>
cqlsh:shop>
cqlsh:shop> SELECT * FROM orders WHERE customer_name = 'Ivan';
cqlsh:shop>
+-----+-----+-----+-----+-----+
| customer_name | order_time | items_ids | order_id | total_price |
+-----+-----+-----+-----+-----+
| Ivan | 2026-01-18 19:05:11.844000+0000 | null | cda1581a-dafc-4d5c-a586-1f0953f3ecc8 | 1500
| Ivan | 2023-01-01 10:00:00.000000+0000 | null | 165bb356-97d4-49a3-85fa-e0fb03a4238b | 200
+-----+-----+-----+-----+-----+
(2 rows)
cqlsh:shop> SELECT sum(total_price) FROM orders WHERE customer_name = 'Ivan';
cqlsh:shop>
+-----+
| system.sum(total_price) |
+-----+
| 1700 |
+-----+
(1 rows)
cqlsh:shop> SELECT writetime(total_price) FROM orders WHERE customer_name = 'Ivan';
cqlsh:shop>
+-----+
| writetime(total_price) |
+-----+
| 1768763111838463 |
| 1768763115691547 |
+-----+
(2 rows)

```

Рисунок 6 — Таблиця *orders* та запити (вибірка, sum, WRITETIME)

6. Розгортання кластеру Cassandra з 3 нод та перевірка конфігурації

Розгорнуто кластер з трьох нод. Перевірено стан та склад кластеру командою nodetool status.

Для розгортання кластеру використано docker-compose.yml з трьома сервісами (cassandra-1/2/3), seed-нodoю cassandra-1 та параметрами обмеження heap (MAX_HEAP_SIZE, HEAP_NEWSIZE). Додано healthcheck через cqlsh, щоб наступні ноди стартували лише після готовності seed-ноди.

```

1 services:
2   cassandra-1:
3     image: cassandra:latest
4     container_name: cassandra-1
5     environment:
6       - CASSANDRA_SEEDS=cassandra-1
7       - CASSANDRA_CLUSTER_NAME=MyCluster
8       - CASSANDRA_ENDPOINT_SNITCH=GossipingPropertyFileSnitch
9       - CASSANDRA_DC=DC1
10      # Ограничеваем память!
11      - MAX_HEAP_SIZE=512M
12      - HEAP_NEWSIZE=100M
13     ports:
14       - 9042:9042
15     healthcheck:
16       test: ["CMD", "cqlsh", "-e", "describe keyspaces"]
17       interval: 30s
18       timeout: 10s
19       retries: 5
20
21   cassandra-2:
22     image: cassandra:latest
23     container_name: cassandra-2
24     environment:
25       - CASSANDRA_SEEDS=cassandra-1
26       - CASSANDRA_CLUSTER_NAME=MyCluster
27       - CASSANDRA_ENDPOINT_SNITCH=GossipingPropertyFileSnitch
28       - CASSANDRA_DC=DC1
29       - MAX_HEAP_SIZE=512M
30       - HEAP_NEWSIZE=100M
31     depends_on:
32       cassandra-1:
33         condition: service_healthy
34
35   cassandra-3:
36     image: cassandra:latest
37     container_name: cassandra-3
38     environment:
39       - CASSANDRA_SEEDS=cassandra-1
40       - CASSANDRA_CLUSTER_NAME=MyCluster
41       - CASSANDRA_ENDPOINT_SNITCH=GossipingPropertyFileSnitch
42       - CASSANDRA_DC=DC1
43       - MAX_HEAP_SIZE=512M
44       - HEAP_NEWSIZE=100M
45     depends_on:
46       cassandra-1:
47         condition: service_healthy
48

```

Рисунок 7 — Фрагмент docker-compose.yml для 3 нод Cassandra

```

(kali㉿Kali-labs)-[~/Desktop/cassandra]
$ sudo docker exec -it cassandra-1 nodetool status
Datacenter: DC1
=====
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address      Load      Tokens  Owns (effective)  Host ID            Rack
UN  172.19.0.4  80.03 KiB  16        59.3%          0e0c2d4f-47c5-4b15-b0fa-2128351ce9e8  rack1
UN  172.19.0.2  119.83 KiB 16        64.7%          765176ec-2644-4c69-8ee1-97a678b0e898  rack1
UN  172.19.0.3  147.5 KiB  16        76.0%          a529d3c1-6def-4b95-9e3b-183cae4afbf5  rack1

```

Рисунок 8 — nodetool status (3 ноди у статуси UN)

7. Keyspace з replication factor 1/2/3 та тестові таблиці

Створено три keyspace (rf_1, rf_2, rf_3) з SimpleStrategy та replication_factor 1, 2, 3. У кожному keyspace створено просту таблицю test та вставлено тестовий рядок.

```
$ sudo docker exec -it cassandra-1 cqlsh
Connected to MyCluster at 127.0.0.1:9042
[cqlsh 6.2.0 | Cassandra 5.0.6 | CQL spec 3.4.7 | Native protocol v5]
Use HELP for help.
cqlsh> CREATE KEYSPACE rf_1 WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 1};
cqlsh> CREATE KEYSPACE rf_2 WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 2};
cqlsh> CREATE KEYSPACE rf_3 WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 3};
cqlsh>
cqlsh> CREATE TABLE rf_1.test (id int PRIMARY KEY, val text);
cqlsh> CREATE TABLE rf_2.test (id int PRIMARY KEY, val text);
cqlsh> CREATE TABLE rf_3.test (id int PRIMARY KEY, val text);
cqlsh>
cqlsh> INSERT INTO rf_1.test (id, val) VALUES (1, 'data_rf1');
cqlsh> INSERT INTO rf_2.test (id, val) VALUES (1, 'data_rf2');
cqlsh> INSERT INTO rf_3.test (id, val) VALUES (1, 'data_rf3');
```

Рисунок 9 — Створення rf_1/rf_2/rf_3, таблиць test та вставка даних

8. Відключення ноди та перевірка читання/запису з різними рівнями Consistency

Відключено одну ноду (cassandra-3) та повторно перевірено стан кластеру. Далі виконано спроби читання з різних keyspace при різних рівнях Consistency.

Спостереження:

- Для rf_1 (RF=1) при недоступності ноди, яка зберігала єдину репліку, читання може завершуватися помилкою навіть на CONSISTENCY ONE.
- Для rf_3 (RF=3) читання на CONSISTENCY ONE залишається доступним (потрібна відповідь хоча б від однієї з живих реплік).
- Для rf_3 при CONSISTENCY ALL потрібні відповіді від усіх реплік; при недоступності однієї ноди отримуємо помилку Unavailable.

```
[kali㉿Kali-labs] -[~/Desktop/cassandra]
$ sudo docker stop cassandra-3
cassandra-3

[kali㉿Kali-labs] -[~/Desktop/cassandra]
$ sudo docker exec -it cassandra-1 nodetool status
Datacenter: DC1
=====
Status=Up/Down
I/ State=Normal/Leaving/Joining/Moving
-- Address      Load    Tokens  Owns    Host ID                               Rack
UN 172.19.0.4   80.75 KiB  16        ?    0e0c2d4f-47c5-4b15-b0fa-2128351ce9e8  rack1
UN 172.19.0.2   120.62 KiB  16        ?    765176ec-2644-4c69-8ee1-97a678b0e898  rack1
DN 172.19.0.3   145.73 KiB  16        ?    a529d3c1-6def-4b95-9e3b-183cae4afbf5  rack1

Note: Non-system keyspaces don't have the same replication settings, effective ownership information is meaningless
```

Рисунок 10 — Відключення ноди cassandra-3 та стан кластера (DN)

```

(kali㉿Kali-labs)-[~/Desktop/cassandra]
$ sudo docker exec -it cassandra-1 cqlsh
Connected to MyCluster at 127.0.0.1:9042
[cqlsh 6.2.0 | Cassandra 5.0.6 | CQL spec 3.4.7 | Native protocol v5]
Use HELP for help.
cqlsh> SELECT * FROM rf_1.test WHERE id = 1;
NoHostAvailable: ('Unable to complete the operation against any hosts', {<Host: 127.0.0.1:9042 DC1>: Unavailable('Error from se
rver: code=1000 [Unavailable exception] message="Cannot achieve consistency level ONE" info={\'consistency\': \'ONE\', \'requir
ed_replicas\': 1, \'alive_replicas\': 0}')})
cqlsh> CONSISTENCY ONE;
Consistency level set to ONE.
cqlsh> SELECT * FROM rf_3.test WHERE id = 1;
id | val
---+---
 1 | data_rf3

(1 rows)
cqlsh> CONSISTENCY ALL;
Consistency level set to ALL.
cqlsh> SELECT * FROM rf_3.test WHERE id = 1;
NoHostAvailable: ('Unable to complete the operation against any hosts', {<Host: 127.0.0.1:9042 DC1>: Unavailable('Error from se
rver: code=1000 [Unavailable exception] message="Cannot achieve consistency level ALL" info={\'consistency\': \'ALL\', \'requir
ed_replicas\': 3, \'alive_replicas\': 2}')})
cqlsh> █

```

Рисунок 11 — Перевірка CONSISTENCY (ONE працює, ALL повертає Unavailable)

9. Аналіз продуктивності counter: ONE vs QUORUM

Таблиця з counter має бути створена у keyspace з RF=3. Далі 10 клієнтів паралельно виконують інкремент по 10_000 разів. Необхідно порівняти час виконання та кінцеве значення лічильника для Consistency ONE та QUORUM (очікувано 100000).

Результати:

- Consistency ONE: час виконання \approx 15.86 сек, кінцеве значення = 100000 (очікуване)
- Consistency QUORUM: час виконання \approx 11.42 сек, кінцеве значення = 100000 (очікуване)

```

(venv)-(kali㉿Kali-labs)-[~/Desktop/cassandra]
$ python3 main.py
— Тест: Consistency ONE —
Время выполнения: 15.86 сек

(venv)-(kali㉿Kali-labs)-[~/Desktop/cassandra]
$ open main.py

(venv)-(kali㉿Kali-labs)-[~/Desktop/cassandra]
$ python3 main.py
— Тест: Consistency QUORUM —
Время выполнения: 11.42 сек

```

Рисунок 12 — Порівняння часу виконання тесту для Consistency ONE та QUORUM

10. Висновок

У ході лабораторної роботи було розгорнуто Cassandra у Docker та виконано моделювання даних у форматі column family. Було створено таблиці items та orders з правильно підібраними ключами для швидких запитів у межах partition key (category для items та customer_name для orders) без використання ALLOW FILTERING. Також було розгорнуто кластер з 3 нод та досліджено вплив replication factor і Consistency Level на доступність читання. Експеримент показав, що RF=1 робить систему вразливою до відмови ноди з даними, тоді як RF=3 дозволяє читати з CONSISTENCY ONE при відмові однієї ноди, але CONSISTENCY ALL стає недоступним. Додатково виконано тест продуктивності для counter (RF=3): Consistency ONE — 15.86 сек, Consistency QUORUM — 11.42 сек (значення можуть відрізнятися через прогрів кешу та навантаження системи).