Отчет по домашнему заданию №8 курса Highload Architect

Запуск:

git clone https://github.com/filatkinen/socialnet

cd socialnet/labs/lab08

make up

Остановка:

make down

Проверка:

После того, как все контейнеры запустятся, подождать примерно еще пару минут.

Затем выполняем файл:

./check.sh

Что происходит при запуске данного скрипта:

1. Вызывается сервис диалогов, реализованный на cassandra — добавление записи диалога
2. Вызывается сервис диалогов, реализованный на postgres — добавление записи диалога

В обоих случаях видим 200 OK

Описание решения

За основу была взята cassandra, а не tarantool, так как понятно, что in-memory DB будет быстрее, чем postgres. Но с in-memory решениями уже были активности, тот же redis. Понятно, что redis это key-value по сравнению с tarantul, которая имеет болле сложную схему, но обе in-memory.

Хотелось сравнить в рамках данной работы скорость с которой cassandra опережает postgres при сохранении данных. Наиболее это было бы заметно при больших объемах БД, но даже при 0-ом наполнении как далее покажут результататы, cassandra выигрывает.

Схема данных cassandra

```
CREATE KEYSPACE IF NOT EXISTS dialog
   WITH replication = {
      'class': 'SimpleStrategy',
      'replication_factor': '1'
      };

CREATE TABLE IF NOT EXISTS dialog.dialogs (
                        user_id text,
                        friend_id text,
                        dialog_id uuid,
                        message text,
                        PRIMARY KEY (user_id, friend_id, dialog_id)
);
```

Схема данных postgres

```
CREATE TABLE dialogs
(
   dialog_id CHAR(36) NOT NULL,
   user_id   CHAR(36),
```

```
    friend_id CHAR(36),
    message   TEXT
);
```

Замечу, что индексы для таблицы в postgres не создавались намеренно, так как мы проверяли только добавление данных и влияние индексов хотелось бы избежать при добавлении данных. Для cassandra без индексов создать таблицу в принципе невозможно.

Рузультаты тестирования

DB	Количество запросов 10000, 100 одновременно	Количество запросов 100000, 1000 одновременно	Количество запросов 1000000, 10000 одновременно
cassandra	Average = 0,01 сек 95% = 0,02 сек 200 OK = 10000	Average = 0,08 сек 95% = 0,29 сек 200 OK = 100000	Average = 0,75 сек 95% = 2,9 сек 200 OK = 995943 400 - 4057
postgres	Average = 0,015 сек 95% = 0,06 сек 200 OK = 10000	Average = 0,12 сек 95% = 0,4 сек 200 OK = 10000	Average = 3,95 сек 95% = 14 сек 200 OK = 918602 400 - 81398

Выводы

Cassandra показывает хорошие результаты по сравнению с postgres при добавлении данных.