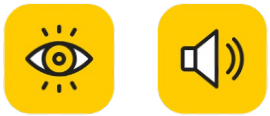
OTUS

NoSQL

otus.ru

Меня хорошо видно & слышно?



Защита проекта

Тема: Сравнение производительности двух NoSQL решений (Cassandra и ClickHouse)

Рутковский Дмитрий

Должность: старший инженер по автоматизации
Компания: VK(RuStore)

План защиты

- Цели проекта
- Что планировалось
- Используемые технологии
- Что получилось
- Схемы/архитектура
- Выводы

Цели проекта

1. Углубление знакомства с колоночными DB ClickHouse и Apache Cassandra
2. Сравнение производительности, затратности ресурсов, возможностей и ограничений связанных с использованием ClickHouse и Apache Cassandra
3. Получение опыта разработки и проведения нагрузочного тестирования noSql решений.

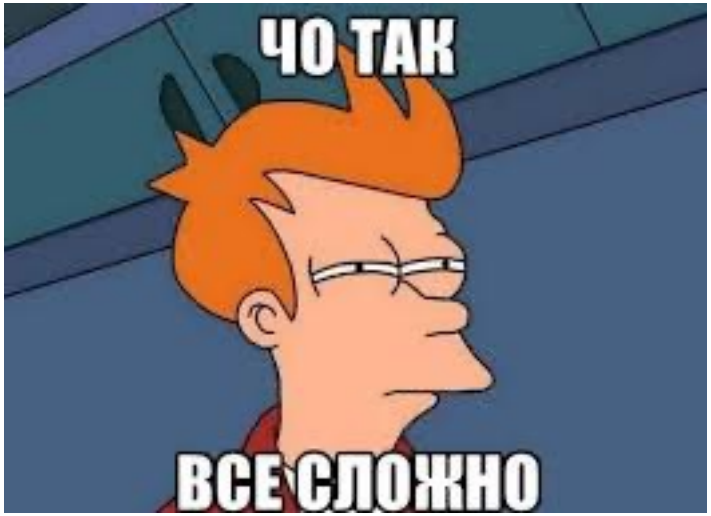
Что планировалось

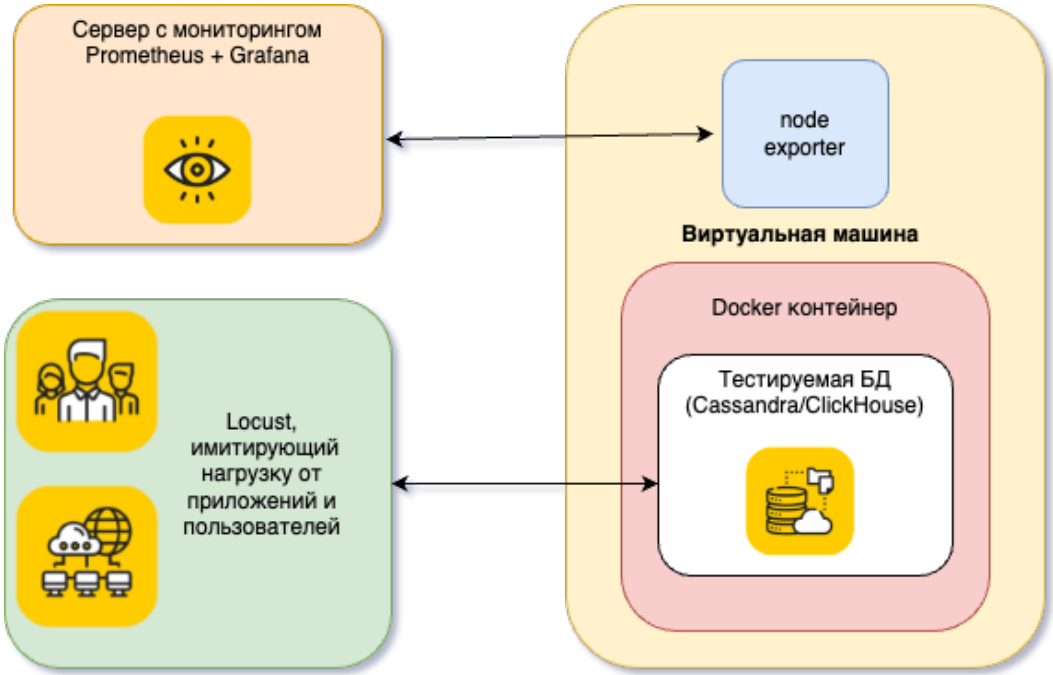
1. Развернуть Cassandra и ClickHouse в Stand Alone режиме (по 1 сущности) настроить базовую аутентификацию по логину и паролю
2. Найти набор данных большого объема (от 1млн. записей), адаптировать его для 2х БД и импортировать в каждую, одинаковую (или близкую) структуру данных
3. Определить инструмент нагрузки, разработать методику, скрипты нагрузочного тестирования
4. Развернуть мониторинг хоста с БД
5. Провести нагрузочные тесты, собрать результаты и статистику
6. Проанализировать результаты, подготовить презентацию.

Используемые технологии

1. Тестируемые БД: ClickHouse, Apache Cassandra развернутые в docker compose
2. Нормализация тестовых данных: Python (csv, pandas и др.)
3. Мониторинг: node_exporeter + prometheus + grafana (prometheus и графана в docker compose на сервере отличном от тестируемого)
4. Инструмент нагрузки: Locust (нагрузочные скрипты на Python)
5. Арендуемая ВМ для БД в яндекс облаке, с 4 CPU/ 8 Gb Memory, 40Gb SSD disk

Схемы (архитектура, БД)





Что НЕ получилось?

1. **Провести нагрузку с помощью jmeter/yandex tank** – не удалось кастомизировать и прикрутить драйвер Cassandra
2. Сравнить под нагрузкой работу **select с множественным условием** (в Cassandra нельзя использовать.)
3. Сравнить под нагрузкой **update** т.к. в clickHouse это, по сути, удаление и вставка
4. Сравнить **агрегатные функции** (sum, count и пр.) т.к. в Cassandra полностью не поддерживаются, варианты:

   а) использовать allow filtering в запросах (согласно документации не рекомендуется)

   б) использовать SASIIndex или представления (оба варианта помечены в документации как экспериментальные и не рекомендованы для production среды
5. 100% соответствия данных. В ходе тестов колонку с типом Float в ClickHouse пришлось преобразовать к типу Int, в связи с **особенностями обработки типа Float** (погрешности после запятой). На запрос select * from recipes2 WHERE total_price = 66.66 AND id = '98505a10-e868-4964-85cd-c1a45911a766’ не возвращается ничего, при этом если сделать запрос только с id в условии – вернется запись с total_price = 66.66. Но если выполнить SELECT * FROM recipes2 WHERE total_price BETWEEN 66.659 AND 66.661; то результат будет получен.

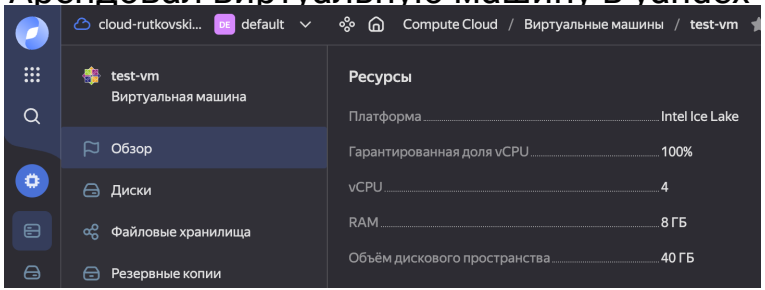
А что же получилось?

1. Выжить без сна
2. Выпить много чашек кофе
3. Потерять несколько выходных
4. Запомнить несколько hotkey VIM

Теперь поподробней...

Развертывание

- Арендовал виртуальную машину в yandex



- Развернул сервисы в docker
  ClickHouse файлы конфигураций и docker compose: ClickHouse github
  Cassandra файлы конфигураций и docker compose: Cassandra github
  Дополнительно использовал инструкции из просторов интернета: раз, два
- Настроил базовую аутентификацию по логину/паролю, остальные настройки оставил по дефолту, за исключением max_concurrent_queries в CH (упирался в ограничение в 100 одновременных запросов в ходе НТ)
- Установил клиенты для macOs для подключения: cqlsh, clickhouse-client, настроил dbeaver

Импорт данных

- Данные для основы брал здесь: ссылка
- Во время импорта, столкнулся с одной стороны с различной интерпретацией спецсимволов у инструментов импорта, удалил часть колонок и спецсимволов
- Добавил колонку с UUID и общей стоимостью, сгенерировал в них данные
- Написал для нормализации python-скрипт: github
- Для импорта в CQL использовал dsbulk-1.11.0 (более информативный лог чем у cqlsh

Итоговые структуры данных

Cassandra

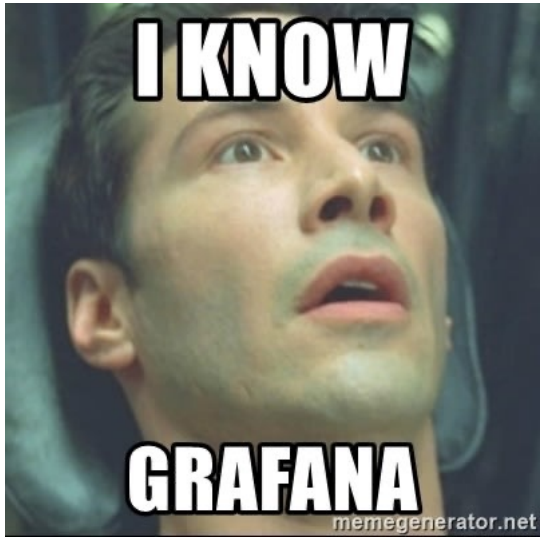
```
CREATE TABLE
my_keyspace.recipes5(
    id uuid,
    title text,
    link text,
    total_price float,
    PRIMARY KEY
(total_price, id));
```

ClickHouse

```
CREATE TABLE recipes2 (
id UUID,
title String,
ingredients String,
link String,
total_price UInt32
) ENGINE = MergeTree()
ORDER BY id;
ALTER TABLE `default`.recipes2
ADD INDEX idx_total_price total_price
TYPE minmax GRANULARITY 1;
OPTIMIZE TABLE `default`.recipes2 FINAL;
```

Мониторинг

- Установил node exporter для мониторинга нагрузочного сервера на хост ВМ
- Развернул на другом сервере связку grafana + prometheus в docker. Настройки и compose.yml: github
- Импортировал дашборд с сайта grafana
- Мониторинг бизнес-метрик (время отклика, пропускная способность, количество ошибок) реализовано штатными средствами locust. (поднимается сервер с веб-интерфейсом)

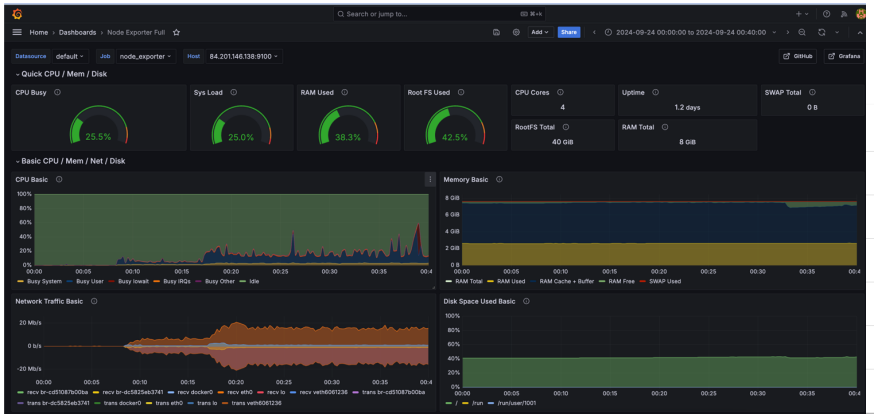


Нагрузочные скрипты

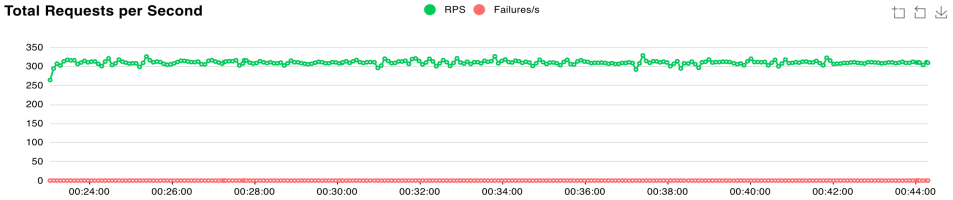
- Разработаны на python помощью фреймворка Locust.Скрипты: ClickHouse и Cassandra
- Выбрана следующая модель нагрузки для обоих инструментов:
  а) 60% запросов select с условием WHERE id = %s
  б)10% запросов insert одной сущности
  в) 10% запросов insert 100 сущностей (для Cassandra в виде батча 100 insert, для ClickHouse в виде одного большого insert)
  г) 10% запросов delete одной сущности с условием: id и total_price
  д) 10% запросов delete 100 сущностей (аналогично insert)
- В виду особенностей полученных результатов первых тестов, по ClickHouse, для дополнительной оценки был запущен ряд тестов:
  а) Тест1 - профиль указан выше.
  б) Тест2 - выключены delete (100 сущностей и 1 сущности)
  в) Тест3 – выключены insert, 100% запросов select
- Изначальная попытка выставить mutations_sync (выполнение delete синхронно) выставленный у ClickHouse для справедливой оценки времени выполнения обернулась провалом. На 6 rps процессор и память утилизировались в 100%, остальные тесты проводились с выключенным параметром.

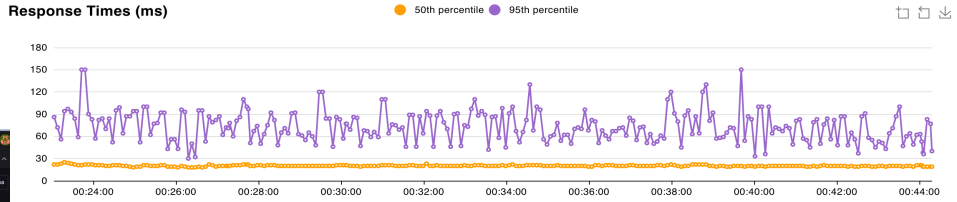
Тест Cassandra

- Тест прошел успешно достигнуто 300rps
- Утилизация CPU до 30% RAM 38%
- Ссылка на график, CSV и HTML-отчет: github



Charts



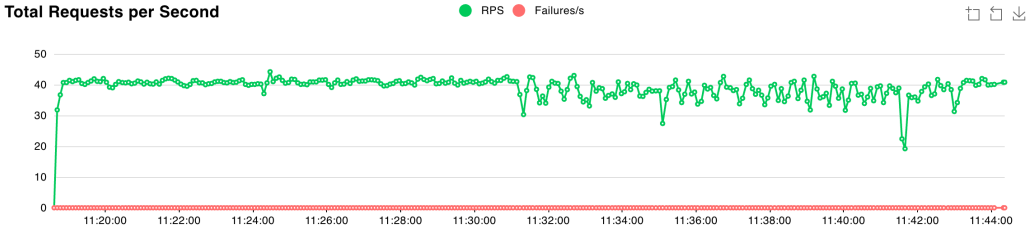


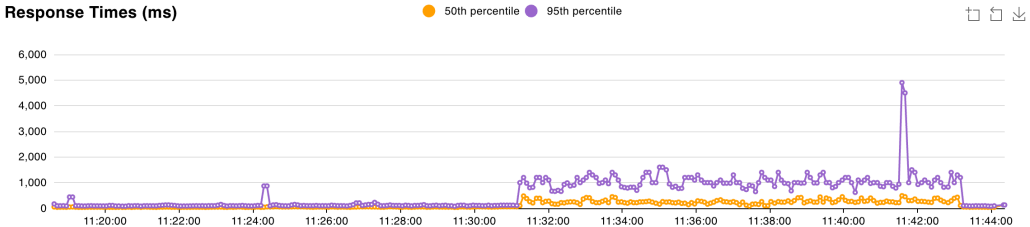
Type	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	RPS	Failures/s
delete	batch_delete	38460	0	26.27	11	696	30.04	0
insert	batch_insert	38459	0	36.89	15	813	30.04	0
delete	simple_delete	38588	0	23.93	8	673	30.14	0
insert	simple_insert	38588	0	28.49	8	792	30.14	0
select	simple_select	243033	0	29.21	9	5790	189.81	0
	Aggregated	397128	0	29.09	8	5790	310.16	0

Тест1 ClickHouse

- Тест прошел успешно достигнуто 40rps, при тесте на 60rps CPU и память утилизовались на 100% и контейнер упал.
- Утилизация CPU до 30% RAM 32%
- Ссылка на график, CSV и HTML-отчет: github

Charts



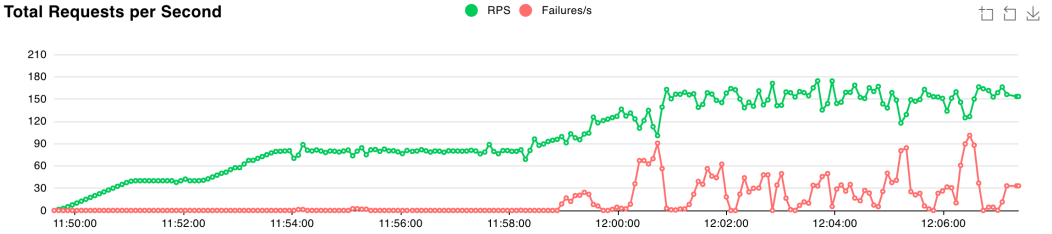


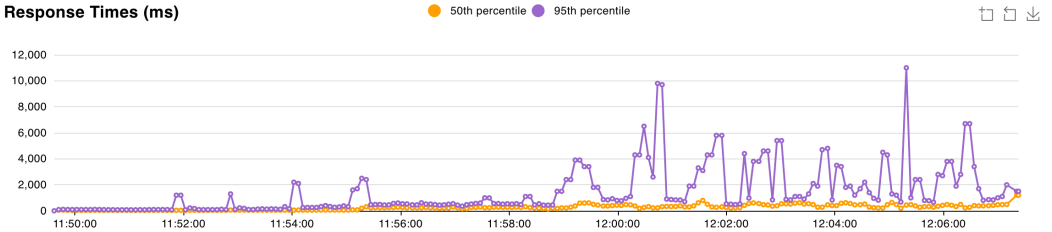
Type	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
delete	group_delete	5886	0	187.95	37	4500	0	3.84	0
insert	group_insert	5886	0	308.23	35	2197	0	3.84	0
delete	simple_delete	5656	0	129.79	20	4494	0	3.69	0
insert	simple_insert	5657	0	146	17	2007	0	3.69	0
select	simple_select	37016	0	194.9	24	5501	0	24.13	0
	Aggregated	60101	0	194.59	17	5501	0	39.17	0

Тест2 ClickHouse

- Тест прошел успешно достигнуто 50rps, при тесте выше 50rps CPU и память утилизовались на 100% и контейнер упал.
- Утилизация CPU до 30% RAM 32%
- Ссылка на график, CSV и HTML-отчет: github

Charts



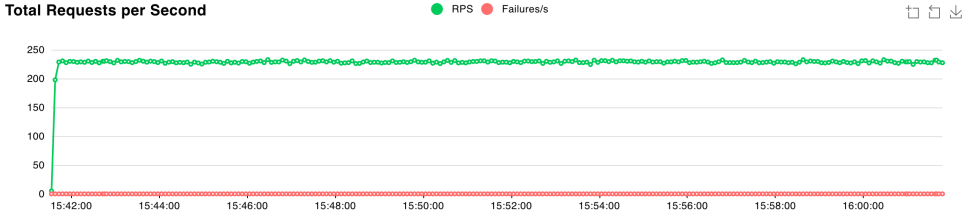


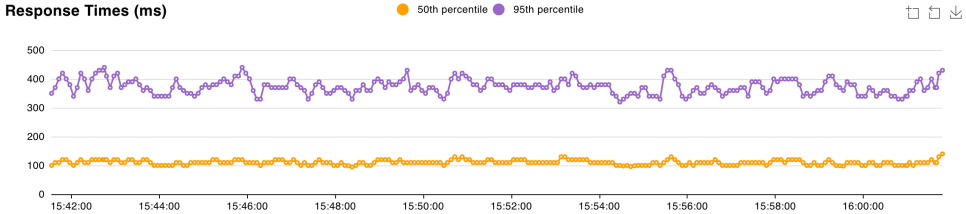
Type	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
insert	group_insert	26723	3632	497.63	35	28740	0	25.32	3.44
insert	simple_insert	26850	3313	452.9	16	32608	0	25.44	3.14
select	simple_select	53707	6715	464.68	16	44495	0	50.89	6.36
	Aggregated	107280	13660	469.94	16	44495	0	101.65	12.94

Тест3 ClickHouse

- Тест прошел успешно достигнуто 230rps, далее закончился ресурс CPU
- Утилизация CPU до 80% RAM 30%
- Ссылка на график, CSV и HTML-отчет: github

Charts





Request Statistics

Type	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
select	simple_select	279093	0	150.47	26	777	0	229.01	0
	Aggregated	279093	0	150.47	26	777	0	229.01	0

Анализ результатов

Type query	CH	Cassandra	CH	Cassandra	CH	Cassandra	CH	Cassandra
	Requests/s		Mediana		90%		Min Response Time	
group_delete	3.8	30.1	89	21	470	36	37	11
group_insert	3.8	30.1	94	27	850	63	35	15
simple_delete	3.7	30.1	61	19	290	34	20	8
simple_insert	3.7	30.1	36	19	470	54	17	8
simple_select	24.1	189.8	67	20	590	53	24	9
Total	39.2	310.2						

1. По всем тестируемым показателям, на дефолтных настройках при работе в 1 инстанс, Cassandra превосходит ClickHouse в использованных запросах для сравнения
2. Delete и групповая вставка является достаточно тяжелой операцией для CH
3. На используемых вычислительных мощностях, Cassandra способна обрабатывать в 9 раз больше запросов. В свою очередь CH требует больше ресурсов.
4. При этом влияние используемых драйверов не замечено (сравни разницу в min time и time по 90%)
5. Cassandra демонстрирует высокую производительность под нагрузкой на больших объемах данных, при этом обладает ограниченным функционалом по обработке данных в сравнении с ClickHouse
6. Возможно, в кластерном режиме или на ноде с большими ресурсами, результаты ,будут отличаться. Требуются дополнительные тесты.
7. Возможно, при увеличении batch с 100 до 1000, ситуация с распределением времени может поменяться. Требуются дополнительные тесты.

Выводы

1. Не смотря на то что обе рассматриваемые БД колоночные, они имеют разное целевое предназначение, и разный функционал.

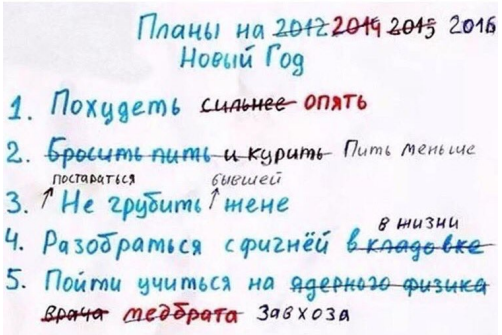
2. В случае Cassandra, запросы выполняются быстрее, ресурсы сервера потребляются меньше, в тоже время, большинство логики предполагается выполнять на стороне приложения. Cassandra более оптимизирована для выполнения delete и update данных.

3. В тоже время, ClickHouse может выполнять сложные статистические операции, группировки и агрегации на своей стороне, освобождая клиента от их выполнения.

4. Оба инструмента, требуют знания конфигурации, особенностей работы с ними и тонкой настройки, в зависимости от окружения и задач, которые с их помощью необходимо решать

Планы по развитию

1. Углубленное изучение реляционных БД (PostgreSQL)
2. Использование Locust как инструмента НТ, для решения рабочих задач
3. Выбор и прохождение DevOps курсов, переход в devOps или DBA
4. Преподавание в OTUS курсов по нагрузочному тестированию, чтобы делиться с другими качественными и полноценными знаниями.



Спасибо за внимание!