



Базы данных

Меня хорошо видно && слышно?



Защита проекта

Тема: Сравнение производительности SQL и NoSQL решений на большом количестве данных



Астраханцев Виктор

Должность: специалист QA
Компания: МТС Диджитал

astrvictor@gmail.com
<https://github.com/astrvictor>



Цели проекта

1. Проверка и закрепление полученных знаний и навыков
2. Получение дополнительных знаний
3. Портфолио для работодателя
4. Запись в сертификате

Задача

Есть большое количество клиентов с мобильными номерами (~ 100 млн) и различными характеристиками (пол, возраст, доход, интересы, ...)

Нужно составлять из клиентов сегменты для рекламы нужного размера, клиенты должны быть выбраны в сегмент исходя из нужных характеристик

Затем клиентам из сегмента может быть предложена различная реклама

Нужно реализовать backend для тестирования, протестировать разные базы данных (SQL, NoSQL, ...) и понять преимущества и недостатки конкретных баз данных для задачи по скорости и удобству работы

Дополнительно нужно реализовать равномерное использование клиентов, чтобы исключить ситуацию, когда одни клиенты используются постоянно а другие никогда не использовались

Какие базы данных используются


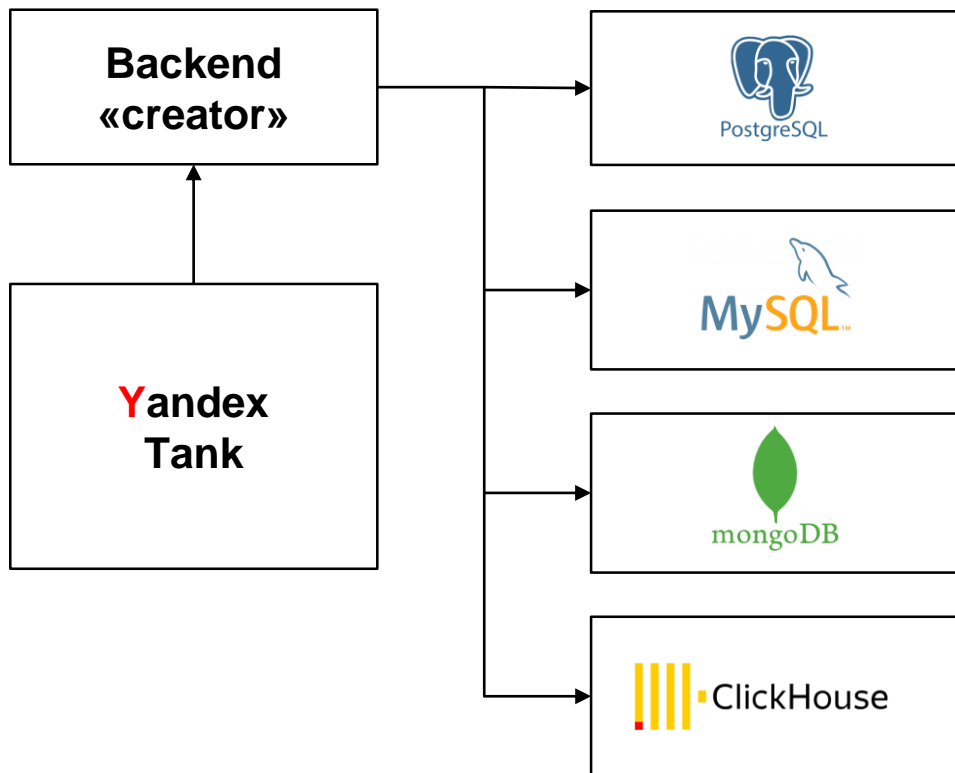
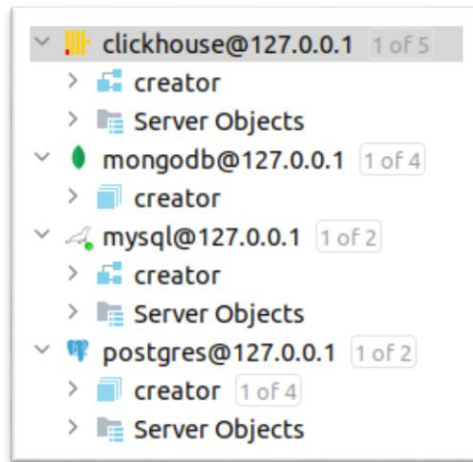
<p>PostgreSQL</p> <p>Тип: реляционная</p> <p>Первый выпуск: 1996</p> <p>Особенности: все возможности реляционных БД, язык запросов SQL</p>	 <p>PostgreSQL</p>	<p>MySQL</p> <p>Тип: реляционная</p> <p>Первый выпуск: 1995</p> <p>Особенности: все возможности реляционных БД, язык запросов SQL</p>	 <p>MySQL™</p>
<p>MongoDB</p> <p>Тип: документная</p> <p>Первый выпуск: 2009</p> <p>Особенности: нет хранимых процедур и триггеров, язык запросов JavaScript</p>	 <p>mongoDB</p>	<p>ClickHouse</p> <p>Тип: колоночная</p> <p>Первый выпуск: 2016</p> <p>Особенности: нет хранимых процедур и триггеров, изменение данных через мутации, язык запросов SQL</p>	 <p>ClickHouse</p>

Схема данных и приложения



clients	
gender	char
age	smallint
income	numeric(10,2)
nextuse	date
msisdn	bigint

segments	
id	uuid
msisdn	bigint



Равномерное использование клиентов

Первая версия с сортировкой по counter – плохо масштабировалась при увеличении количества

```
-- Создание сегмента
INSERT INTO creator.segments(id, msisdn)
SELECT uuid, msisdn
FROM creator.clients
ORDER BY counter
LIMIT size;

-- Обновление для равномерного использования
UPDATE creator.clients
SET counter = counter + 1
WHERE msisdn IN (
    SELECT msisdn
    FROM creator.segments
    WHERE id = uuid
);
```

Вторая версия с минимальной датой следующего использования – масштабируется хорошо

```
-- Создание сегмента
INSERT INTO creator.segments(id, msisdn)
SELECT uuid, msisdn
FROM creator.clients
WHERE nextuse < date now()
LIMIT size;

-- Обновление для равномерного использования
UPDATE creator.clients
SET nextuse = now() + 10
WHERE msisdn IN (
    SELECT msisdn
    FROM creator.segments
    WHERE id = uuid
);
```


Реализация на PostgreSQL

```
CREATE SCHEMA IF NOT EXISTS creator;  
  
CREATE TABLE IF NOT EXISTS creator.clients (  
    msisdn bigint primary key,  
    gender char(1),  
    age smallint,  
    income decimal(10,2),  
    nextuse date  
);  
  
CREATE INDEX clients_nextuse ON  
    creator.clients (nextuse);  
  
CREATE TABLE IF NOT EXISTS creator.segments (  
    id uuid,  
    msisdn bigint  
);  
  
CREATE INDEX segments_id_msisdn ON  
    creator.segments (id, msisdn);
```

```
-- Создание сегмента  
INSERT INTO creator.segments(id, msisdn)  
SELECT uuid, msisdn  
FROM creator.clients  
WHERE nextuse < date now()  
LIMIT size;  
  
-- Обновление для равномерного использования  
UPDATE creator.clients  
SET nextuse = now() + 10  
WHERE msisdn IN (  
    SELECT msisdn  
    FROM creator.segments  
    WHERE id = uuid  
);  
  
-- Получение сегмента  
SELECT msisdn  
FROM creator.segments  
WHERE id = uuid;
```

Реализация на MySQL

```
USE creator;

CREATE TABLE IF NOT EXISTS clients (
    msisdn bigint primary key,
    gender char(1),
    age tinyint,
    income decimal(10,2),
    nextuse date
) ENGINE = InnoDB;

CREATE INDEX clients_nextuse ON
    creator.clients (nextuse);

CREATE TABLE IF NOT EXISTS segments (
    id binary(16),
    msisdn bigint
) ENGINE = InnoDB;

CREATE INDEX segments_id_msisdn ON
    creator.segments (id, msisdn);
```

```
# Создание сегмента
INSERT INTO creator.segments(id, msisdn)
SELECT UUID_TO_BIN(uuid), msisdn
FROM creator.clients
WHERE nextuse < date now()
LIMIT size;

# Обновление для равномерного использования
UPDATE creator.clients
SET nextuse = now() + 10
WHERE msisdn IN (
    SELECT msisdn
    FROM creator.segments
    WHERE id = UUID_TO_BIN(uuid)
);

# Получение сегмента
SELECT msisdn
FROM creator.segments
WHERE id = UUID_TO_BIN(uuid);
```

Реализация на MongoDB

```
use creator;

// Создание клиентов
db.getSiblingDB("creator").getCollection("clients").insertMany([
  { _id: 790000000001, gender: "M", age: 25,
    income: 10000.00, nextuse: ISODate('2022-01-01T00:00:00.000Z') },
  { _id: 790000000002, gender: "F", age: 41,
    income: 20000.00, nextuse: ISODate('2022-01-01T00:00:00.000Z') }
]);

// Получение _id из clients
db.getSiblingDB("creator").getCollection("clients").find(
  { nextuse: { $lt: ISODate('2022-12-30T00:00:00.000Z') } }, { _id: 1 }
).limit(size)
```

```
// Создание сегментов
db.getSiblingDB("creator").getCollection("segments").insertMany([
  { id: "12345678-1234-5678-1234-567812345678", msisdn: 790000000001 },
  { id: "12345678-1234-5678-1234-567812345678", msisdn: 790000000002 }
]);

// Обновление nextuse
db.getSiblingDB("creator").getCollection("clients").updateMany(
  { _id: { $in: [790000000001, 790000000002] } },
  { "$set": { nextuse: ISODate('2022-12-30T00:00:00.000Z') } }
);

// Получение сегмента
db.getSiblingDB("creator").getCollection("segments").find(
  { id: "12345678-1234-5678-1234-567812345678" }, { _id: 0, msisdn: 1 }
);
```



Реализация на ClickHouse

```
CREATE DATABASE IF NOT EXISTS creator;  
  
CREATE TABLE IF NOT EXISTS creator.clients (  
    msisdn UInt64,  
    gender char(1),  
    age UInt8,  
    income Float32,  
    nextuse Date  
    ) ENGINE = MergeTree()  
    ORDER BY (msisdn);  
  
CREATE TABLE IF NOT EXISTS creator.segments (  
    id UUID,  
    msisdn UInt64  
    ) ENGINE = MergeTree()  
    PARTITION BY id  
    ORDER BY (id);  
  
-- Включение синхронных мутаций  
SET mutations_sync = 1;
```

```
-- Создание сегмента  
INSERT INTO creator.segments(id, msisdn)  
SELECT uuid, msisdn  
FROM creator.clients  
WHERE nextuse < date now()  
LIMIT size;  
  
-- Обновление для равномерного использования  
ALTER TABLE creator.clients  
UPDATE nextuse = now() + 10  
WHERE msisdn IN (  
    SELECT msisdn  
    FROM creator.segments  
    WHERE id = uuid  
);  
  
-- Получение сегмента  
SELECT msisdn  
FROM creator.segments  
WHERE id = uuid;
```

Методика тестирования

Hardware: Intel Core i7-3820 3,60 GHz, DDR3 SDRAM 1600 MHz, Samsung SSD 870 EVO SATA 2,5

Software: Host Ubuntu 20.04, VM VirtualBox 6.1.38 (Ubuntu 20.04, 4 core, 20 Gb), Docker 20.10.12, Docker-Compose 1.25.0,

Образы БД: clickhouse/clickhouse-server:22.8.11.15-alpine, postgres:15.1-bullseye, mysql:8.0.31-debian, mongo:5.0.14

Тест создания клиентов: Создание базы клиентов на 100 000 000 порциями по 10 000, измерение времени создания и размера занимаемых данных

Тест создания сегментов: Заполнение базы сегментов ~ 900 сегментов по 10 000 клиентов (1 сегмент в секунду, длительность 15 минут), после этого измерение времени создания 1 сегмента с помощью Yandex Tank (1 сегмент в секунду, длительность теста 2 минуты)

Тест получения сегментов: Измерение скорости получения сегмента из базы с помощью Yandex Tank (10 сегментов в секунду, длительность теста 1 минута)

Методы backend «creator»

```
curl --request POST 'http://127.0.0.1:8888/database/clickhouse'
```

// переключение между базами данных, варианты: clickhouse, mysql, postgres, mongodb

```
curl --request POST 'http://127.0.0.1:8888/clients/1000000'
```

```
{"database":"ClickHouse","size":1000000,"duration":4.113}
```

// создание базы данных клиентов нужного размера

```
curl --request POST 'http://127.0.0.1:8888/segment/10000'
```

```
{"database":"ClickHouse","uuid":"f876cf5f-e6a9-4a7b-881d-20b4059ea9e4","size":10000,"duration":0.025}
```

// создание сегмента нужного размера

```
curl --request GET 'http://127.0.0.1:8888/segment'
```

```
{"database":"ClickHouse","uuid":"f876cf5f-e6a9-4a7b-881d-20b4059ea9e4","size":10000,"duration":0.008}
```

// получение случайного сегмента из базы данных

```
curl --request DELETE 'http://127.0.0.1:8888/clients'
```

// очистка базы данных клиентов

Результаты тестов: создание 100М клиентов

```
astrvictor@minikube:~$ docker system df -v | grep "deployments_volume"
deployments_volume_mongodb          1          314.9MB
deployments_volume_mysql             1          209.5MB
deployments_volume_postgres          1          47.36MB
deployments_volume_clickhouse        1          372.3kB
astrvictor@minikube:~$
astrvictor@minikube:~$ curl --request POST 'http://127.0.0.1:8888/database/postgres'
astrvictor@minikube:~$ curl --request POST 'http://127.0.0.1:8888/clients/100000000'
{"database": "PostgreSQL", "size": 100000000, "duration": 1314.216}
astrvictor@minikube:~$ curl --request POST 'http://127.0.0.1:8888/database/mysql'
astrvictor@minikube:~$ curl --request POST 'http://127.0.0.1:8888/clients/100000000'
{"database": "MySQL", "size": 100000000, "duration": 4000.461}
astrvictor@minikube:~$ curl --request POST 'http://127.0.0.1:8888/database/mongodb'
astrvictor@minikube:~$ curl --request POST 'http://127.0.0.1:8888/clients/100000000'
{"database": "MongoDB", "size": 100000000, "duration": 1040.334}
astrvictor@minikube:~$ curl --request POST 'http://127.0.0.1:8888/database/clickhouse'
astrvictor@minikube:~$ curl --request POST 'http://127.0.0.1:8888/clients/100000000'
{"database": "ClickHouse", "size": 100000000, "duration": 429.451}
astrvictor@minikube:~$
astrvictor@minikube:~$ docker system df -v | grep "deployments_volume"
deployments_volume_mysql             1          8.181GB
deployments_volume_postgres          1          9.215GB
deployments_volume_clickhouse        1          6.173GB
deployments_volume_mongodb           1          3.978GB
astrvictor@minikube:~$
astrvictor@minikube:~$ docker system df -v | grep "deployments_volume"
deployments_volume_mysql             1          8.181GB
deployments_volume_postgres          1          9.215GB
deployments_volume_clickhouse        1          1.284GB
deployments_volume_mongodb           1          3.978GB
astrvictor@minikube:~$ █
```

Время создания базы 100М клиентов, сек

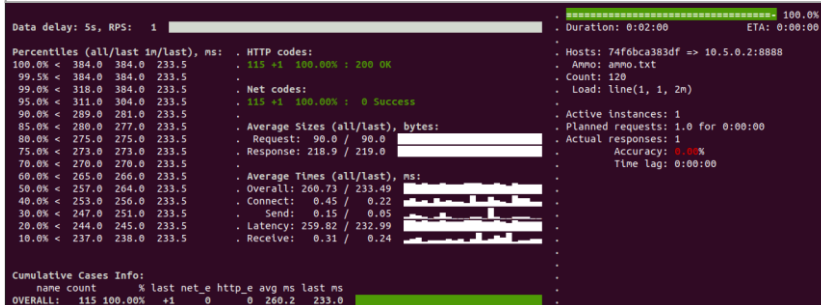
PostgreSQL	1314 (21,9 мин)
MySQL	4000 (66,6 мин)
MongoDB	1040 (17,3 мин)
ClickHouse	429 (7,2 мин)

Размер базы клиентов, GB

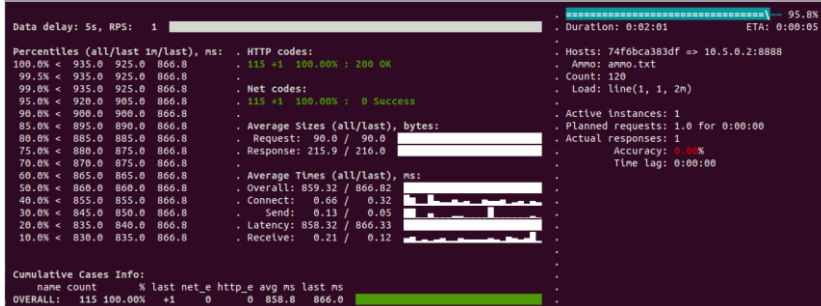
PostgreSQL	9,215
MySQL	8,181
MongoDB	3,978
ClickHouse	6,173 (1,284)

Результаты тестов: создание сегментов

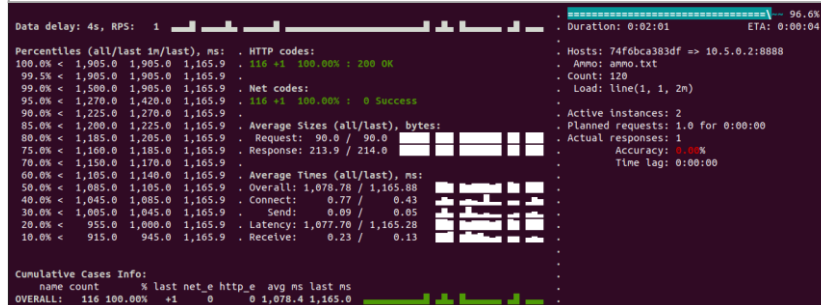
PostgreSQL: percentile 95.0% < 318ms



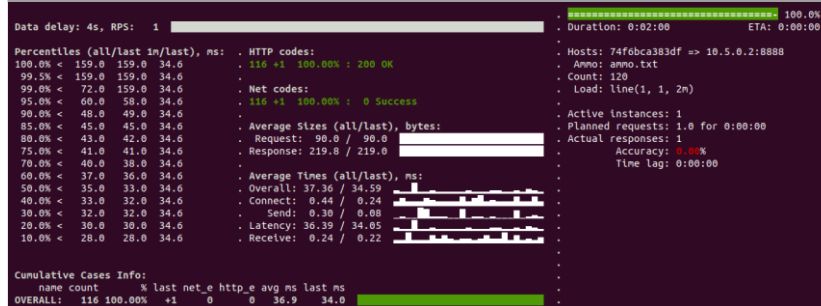
MongoDB: percentile 95.0% < 920ms



MySQL: percentile 95.0% < 1270ms



ClickHouse: percentile 95.0% < 60ms



PostgreSQL: percentile 95.0% < 17ms

The screenshot shows the output of the pgbench tool. At the top, it indicates 'Data delay: 3s, RPS: 10' and a progress bar. The 'Percentiles (all/last in/last), ms:' section shows a 95th percentile of 17.0 ms. The 'Average Times (all/last), ms:' section shows an overall average of 11.45 ms. The 'Cumulative Cases Info:' table shows 573 successful requests with an overall average of 11.4 ms.

name	count	% last	net_e	http_e	avg	ms	last	ms
OVERALL:	573	100.00%	+10	0	0	11.4	11.1	

MySQL: percentile 95.0% < 31ms

The screenshot shows the output of the pgbench tool for MySQL. The 'Percentiles (all/last in/last), ms:' section shows a 95th percentile of 31.0 ms. The 'Average Times (all/last), ms:' section shows an overall average of 20.78 ms. The 'Cumulative Cases Info:' table shows 574 successful requests with an overall average of 20.7 ms.

name	count	% last	net_e	http_e	avg	ms	last	ms
OVERALL:	574	100.00%	+10	0	0	20.7	21.5	

MongoDB: percentile 95.0% < 73ms

The screenshot shows the output of the pgbench tool for MongoDB. The 'Percentiles (all/last in/last), ms:' section shows a 95th percentile of 73.0 ms. The 'Average Times (all/last), ms:' section shows an overall average of 51.79 ms. The 'Cumulative Cases Info:' table shows 572 successful requests with an overall average of 51.7 ms.

name	count	% last	net_e	http_e	avg	ms	last	ms
OVERALL:	572	100.00%	+10	0	0	51.7	45.1	

ClickHouse: percentile 95.0% < 23ms

The screenshot shows the output of the pgbench tool for ClickHouse. The 'Percentiles (all/last in/last), ms:' section shows a 95th percentile of 23.0 ms. The 'Average Times (all/last), ms:' section shows an overall average of 14.66 ms. The 'Cumulative Cases Info:' table shows 570 successful requests with an overall average of 14.6 ms.

name	count	% last	net_e	http_e	avg	ms	last	ms
OVERALL:	570	100.00%	+10	0	0	14.6	15.1	

Выводы и планы по развитию

1. Задачи проекта выполнены

2. Результат: ClickHouse самый быстрый и экономный по месту (но используются мутации), Mongo интересный но сложноватый из-за JavaScript вместо SQL, PostgreSQL быстрее MySQL ~ x3 раза

3. Что не успел:

– Сравнить с key-value базами данных

– Проверить кластеры

– Проверить выборки с большим количеством условий

Спасибо за внимание!