BIGDATA TEAM

MegaFon Course: Big Data

Spark + Cassandra

Andrey Titov, andrey.titov@bigdatateam.org
Big Data Instructor @ BigData Team, http://bigdatateam.org/
Senior Spark Engineer @ NVIDIA

26.08.2019, Moscow, Russia

BIGDATA TEAM

На этом занятии

- Зачем нужен NoSQL, когда есть SQL
- CAP теорема и где там Cassandra
- Архитектура БД
- CQL
- Утилиты cqlsh и nodetool
- Spark Cassandra connector
- Workshop

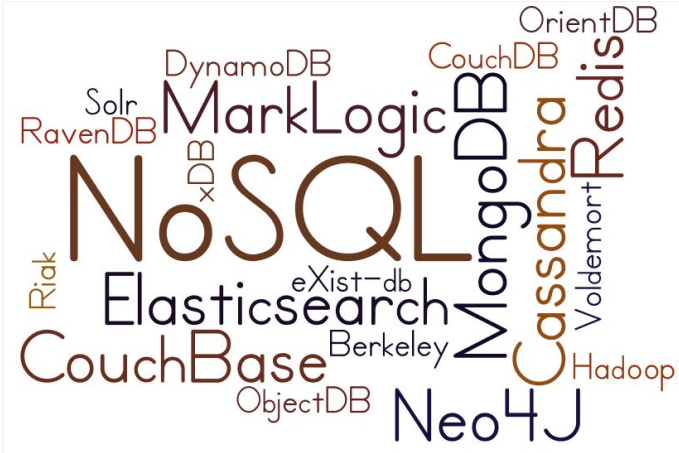
Зачем нужен NoSQL

BIGDATA TEAM

Зачем нужен NoSQL

NoSQL - термин, описывающий класс БД, имеющих архитектурные отличия от классических реляционных БД

Основными факторами развития NoSQL БД считаются:



- скорость запросов
- количество данных
- ACID не нужен не только лишь всем

Пример использования

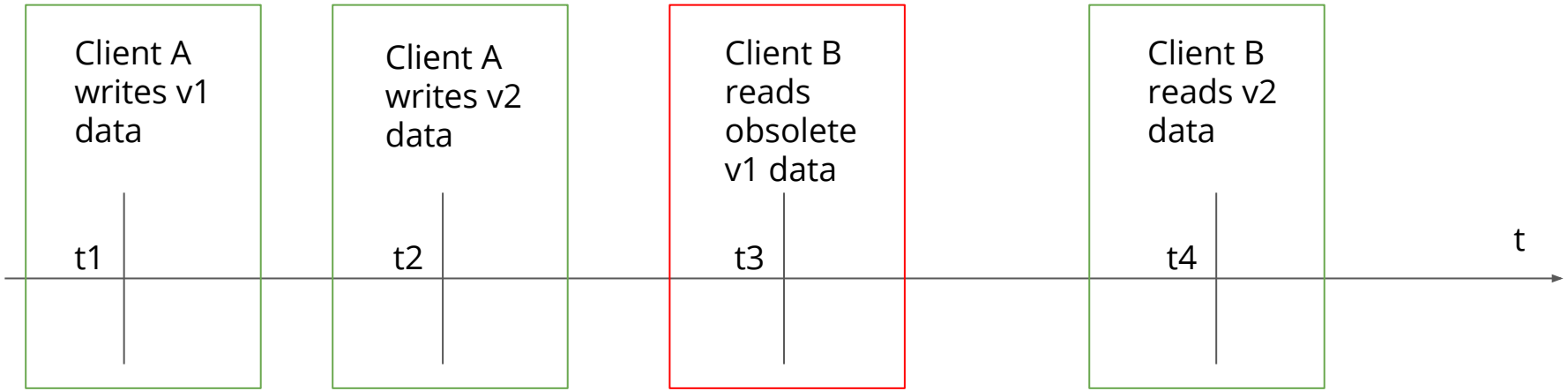
Вы проектируете платформу сбора данных с различных датчиков автомобилей всего мира:

- Критична ли потеря одного события?
- Что нам важнее - консистентность данных или доступность системы?
- Будут ли проблемы с сетью?
- Сколько событий в секунду мы будем обрабатывать?

BIGDATA TEAM

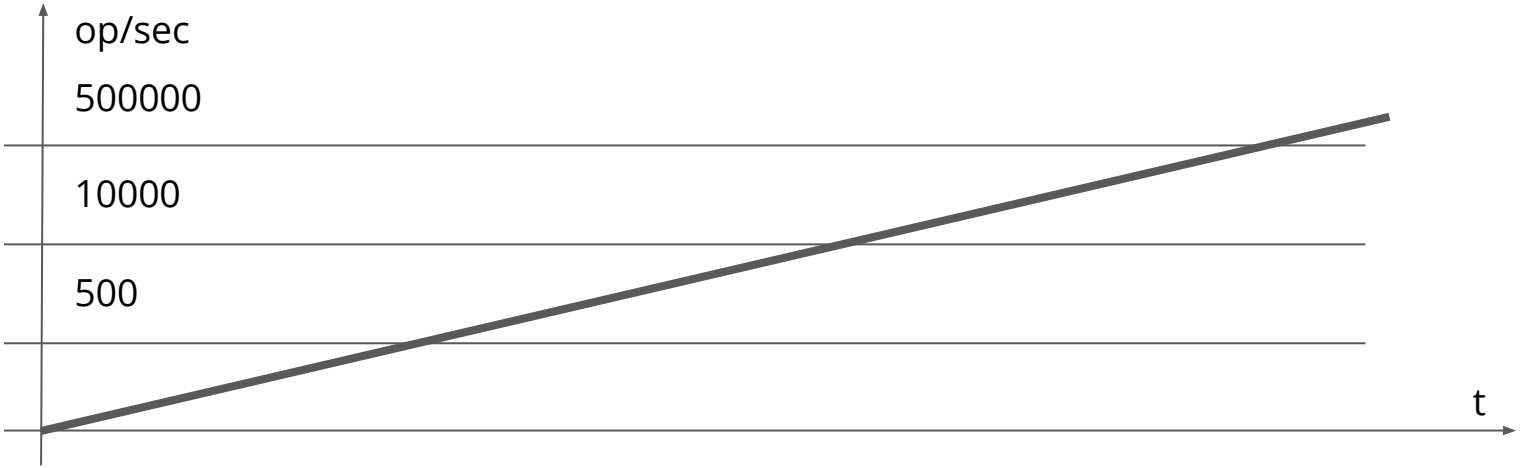
Eventual consistency

Eventual consistency - режим работы БД, при котором допускается некоторое время **с** момента получения подтверждения записи новых данных **до** доступности этих данных при чтении



Web scale

Web scale - класс проектируемых систем, в отношении которых невозможно определить даже порядок количества пользователей и нагрузки на нее

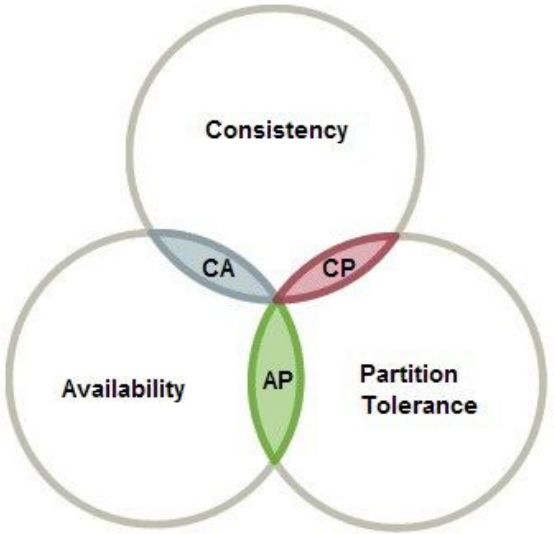


Теорема CAP Брюера

CAP теорема

В любой распределенной БД возможно гарантировать выполнение двух из трех свойств:

- Consistency
- Availability
- Partition tolerance



CAP теорема

Consistency

Результат любого запроса проявляется везде и сразу после того, как мы получили ответ на запрос

Availability

Любая доступная нода должна ответить на запрос

Partition tolerance

Система продолжает работать в условиях нарушения сетевой связности

CAP теорема

CA системы

При возникновении проблем все ноды перестают обрабатывать запросы, но зато все консистентно :3

CP системы

В случае проблем никто не гарантирует доступность данных

AP системы

Система будет доступна после ядерного апокалипсиса, но некоторое время может возвращать не то, что вы ожидаете

CAP теорема

Cassandra - AP система в теореме CAP. На практике это означает:

- высокая доступность данных
- нет транзакций (не совсем)
- можно строить гео-кластера
- слабая согласованность (eventual)
- линейная масштабируемость
- высокая пропускная способность (особенно на запись)

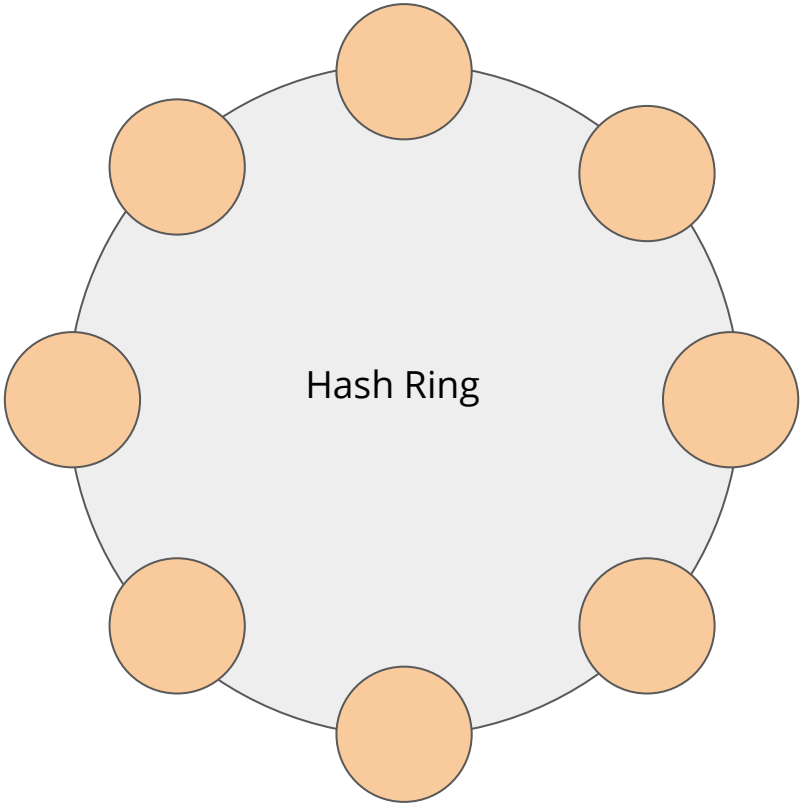
Архитектура БД

Симметричная система

Cassandra имеет симметричную архитектуру

Каждый узел отвечает за хранение данных, обработку запросов и состояние кластера

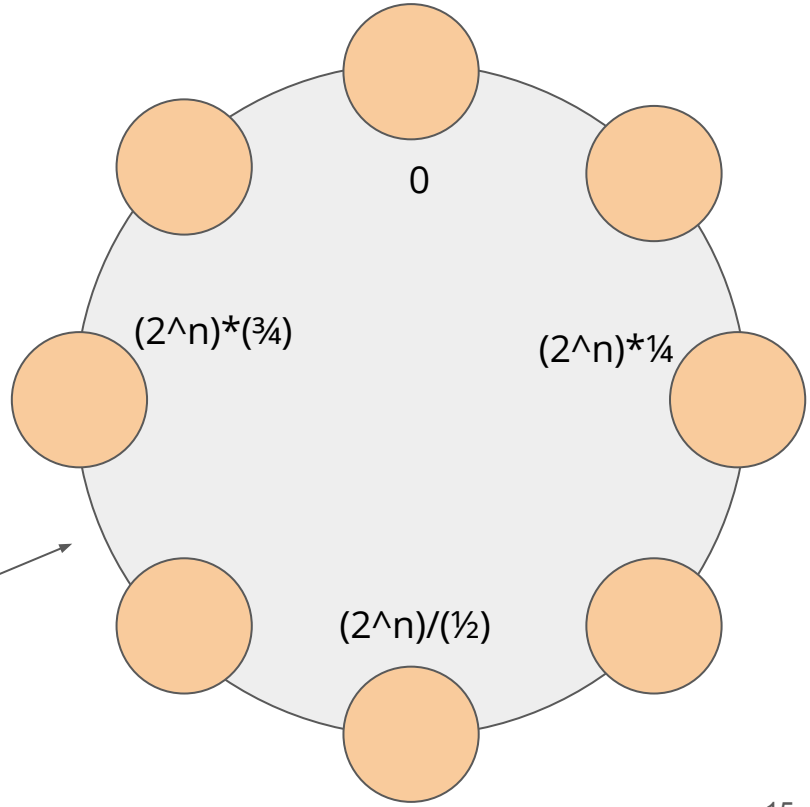
Расположение данных определяется значением хеш функции от **Partition key**



BIGDATA TEAM

Hash Ring

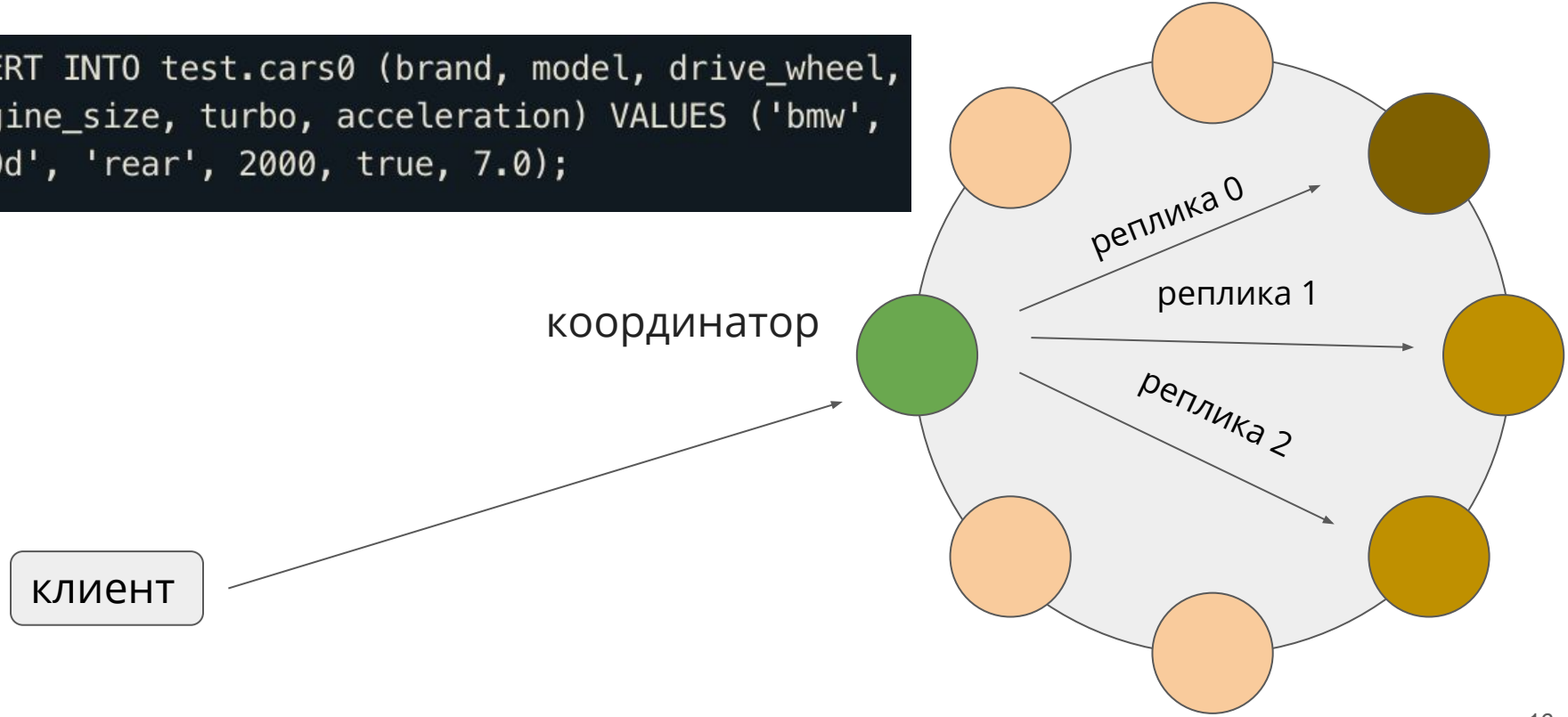
```
CREATE TABLE IF NOT EXISTS test.cars0 (
    brand text PRIMARY KEY,
    model text,
    engine_size int,
    drive_wheel text,
    turbo boolean,
    acceleration float);
```



BIGDATA TEAM

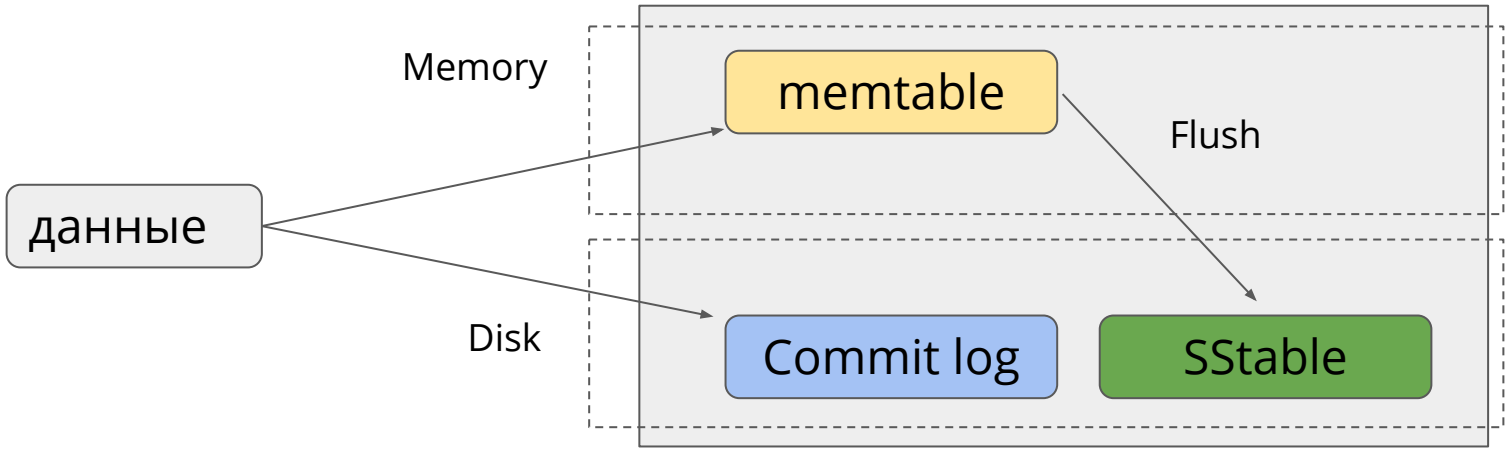
Запись данных

```
INSERT INTO test.cars0 (brand, model, drive_wheel,
 engine_size, turbo, acceleration) VALUES ('bmw',
'220d', 'rear', 2000, true, 7.0);
```



BIGDATA TEAM

Архитектура узла



Создание таблиц

Чтение и запись данных

BIGDATA TEAM

Создание таблицы

```
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.4 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh> CREATE TABLE IF NOT EXISTS test.cars0 (
   ...     brand text PRIMARY KEY,
   ...     model text,
   ...     engine_size int,
   ...     drive_wheel text,
   ...     turbo boolean,
   ...     acceleration float);
cqlsh> SELECT * FROM test.cars0;

 brand | acceleration | drive_wheel | engine_size | model | turbo
-------+--------------+-------------+-------------+-------+-------

(0 rows)
cqlsh>
```

BIGDATA TEAM

Запись данных в таблицу

```
cqlsh> INSERT INTO test.cars0 (brand, model, drive_wheel, engine_size, turbo, acceleration)
 VALUES ('bmw', '220d', 'rear', 2000, true, 7.0);
cqlsh> SELECT * FROM test.cars0;

 brand | acceleration | drive_wheel | engine_size | model | turbo
-------+--------------+-------------+-------------+-------+-------
   bmw |            7 |        rear |        2000 |  220d |  True

(1 rows)
cqlsh>
```

BIGDATA TEAM

Фильтрация по Partition key

```
cqlsh> SELECT * FROM test.cars0 WHERE brand = 'bmw';

 brand | acceleration | drive_wheel | engine_size | model | turbo
-------+--------------+-------------+-------------+-------+-------
   bmw |            7 |        rear |        2000 |  220d |  True

(1 rows)
cqlsh> SELECT * FROM test.cars0 WHERE model = '220d';
InvalidRequest: Error from server: code=2200 [Invalid query] message="Cannot execute this q
uery as it might involve data filtering and thus may have unpredictable performance. If you
 want to execute this query despite the performance unpredictability, use ALLOW FILTERING"
```

BIGDATA TEAM

Композитный ключ

```
cqlsh> CREATE TABLE IF NOT EXISTS test.cars1 (
   ...      brand text,
   ...      model text,
   ...      engine_size int,
   ...      drive_wheel text,
   ...      turbo boolean,
   ...      acceleration float, PRIMARY KEY (brand, model));
cqlsh>
cqlsh> SELECT * FROM test.cars1;

 brand | model | acceleration | drive_wheel | engine_size | turbo
-------+-------+--------------+-------------+-------------+-------

(0 rows)
```

BIGDATA TEAM

Фильтрация по композитному ключу

```
cqlsh> SELECT * FROM test.cars1 WHERE brand = 'bmw';

 brand | model | acceleration | drive_wheel | engine_size | turbo
-------+-------+--------------+-------------+-------------+-------
   bmw |  220d |            7 |        rear |        2000 |  True

(1 rows)
cqlsh> SELECT * FROM test.cars1 WHERE model = '220d';
InvalidRequest: Error from server: code=2200 [Invalid query] message="Cannot execute this q
uery as it might involve data filtering and thus may have unpredictable performance. If you
 want to execute this query despite the performance unpredictability, use ALLOW FILTERING"
cqlsh> SELECT * FROM test.cars1 WHERE brand = 'bmw' and model = '220d';

 brand | model | acceleration | drive_wheel | engine_size | turbo
-------+-------+--------------+-------------+-------------+-------
   bmw |  220d |            7 |        rear |        2000 |  True

(1 rows)
```

Расширение композитного ключа

```
cqlsh> CREATE TABLE IF NOT EXISTS test.cars2 (
   ...     brand text,
   ...     model text,
   ...     engine_size int,
   ...     drive_wheel text,
   ...     turbo boolean,
   ...     acceleration float, PRIMARY KEY (brand, model, engine_size));
cqlsh>
cqlsh>
cqlsh>
cqlsh> SELECT * FROM test.cars2;

 brand | model | engine_size | acceleration | drive_wheel | turbo
-------+-------+-------------+--------------+-------------+-------

(0 rows)
```

BIGDATA TEAM

Пропуск Cluster key при фильтрации

```
cqlsh> SELECT * FROM test.cars2 WHERE brand = 'bmw' and model = '220d' and engine_size = 20
00;

 brand | model | engine_size | acceleration | drive_wheel | turbo
-------+-------+-------------+--------------+-------------+-------
   bmw |  220d |        2000 |            7 |        rear |  True

(1 rows)
cqlsh> SELECT * FROM test.cars2 WHERE brand = 'bmw' and engine_size = 2000;
InvalidRequest: Error from server: code=2200 [Invalid query] message="PRIMARY KEY column "e
ngine_size" cannot be restricted as preceding column "model" is not restricted"
cqlsh>
```

non-EQ relation

```
cqlsh> SELECT * FROM test.cars2 WHERE brand = 'bmw' and model = '220d' and engine_size <= 2
000;

 brand | model | engine_size | acceleration | drive_wheel | turbo
-------+-------+-------------+--------------+-------------+-------
   bmw |  220d |        2000 |            7 |        rear |  True

(1 rows)
```

```
cqlsh> SELECT * FROM test.cars2 WHERE brand = 'bmw' and model in ('220d', '320ix') and engi
ne_size <= 2000;

 brand | model | engine_size | acceleration | drive_wheel | turbo
-------+-------+-------------+--------------+-------------+-------
   bmw |  220d |        2000 |            7 |        rear |  True

(1 rows)
```

BIGDATA TEAM

Расширение Partition key

```
cqlsh> CREATE TABLE IF NOT EXISTS test.cars3 (
   ...     brand text,
   ...     model text,
   ...     engine_size int,
   ...     drive_wheel text,
   ...     turbo boolean,
   ...     acceleration float, PRIMARY KEY ((brand, model), drive_wheel, engine_size));
cqlsh>
cqlsh> SELECT * FROM test.cars3;

 brand | model | drive_wheel | engine_size | acceleration | turbo
-------+-------+-------------+-------------+--------------+-------

(0 rows)
```

BIGDATA TEAM

Пропуск Partition key при фильтрации

```
cqlsh> SELECT * FROM test.cars3 WHERE brand = 'bmw' and model in ('220d');

 brand | model | drive_wheel | engine_size | acceleration | turbo
-------+-------+-------------+-------------+--------------+-------
   bmw |  220d |        rear |        2000 |            7 |  True

(1 rows)
cqlsh> SELECT * FROM test.cars3 WHERE model = '220d';
InvalidRequest: Error from server: code=2200 [Invalid query] message="Cannot execute this q
uery as it might involve data filtering and thus may have unpredictable performance. If you
 want to execute this query despite the performance unpredictability, use ALLOW FILTERING"
cqlsh> SELECT * FROM test.cars3 WHERE brand = 'bmw';
InvalidRequest: Error from server: code=2200 [Invalid query] message="Cannot execute this q
uery as it might involve data filtering and thus may have unpredictable performance. If you
 want to execute this query despite the performance unpredictability, use ALLOW FILTERING"
```

BIGDATA TEAM

Фильтрация по композитному ключу

```
cqlsh> SELECT * FROM test.cars3 WHERE brand in ('bmw') and model = '220d';

 brand | model | drive_wheel | engine_size | acceleration | turbo
-------+-------+-------------+-------------+--------------+-------
   bmw |  220d |        rear |        2000 |            7 |  True

(1 rows)
```

```
cqlsh> SELECT * FROM test.cars3 WHERE brand = 'bmw' and model = '220d' and drive_wheel = 'r
ear' and engine_size >= 2000;

 brand | model | drive_wheel | engine_size | acceleration | turbo
-------+-------+-------------+-------------+--------------+-------
   bmw |  220d |        rear |        2000 |            7 |  True

(1 rows)
```

BIGDATA TEAM

non-EQ relation по двум Cluster key

```
cqlsh> SELECT * FROM test.cars3 WHERE brand = 'bmw' and model = '220d' and drive_wheel >= '
rear' and engine_size >= 2000;
InvalidRequest: Error from server: code=2200 [Invalid query] message="Clustering column "en
gine_size" cannot be restricted (preceding column "drive_wheel" is restricted by a non-EQ r
elation)"
```

BIGDATA TEAM

Перезапись данных

```
cqlsh> INSERT INTO test.cars3 (brand, model, drive_wheel, engine_size, turbo, acceleration)
 VALUES ('bmw', '220d', 'rear', 2000, true, 6.9);
cqlsh> SELECT * FROM test.cars3;

 brand | model | drive_wheel | engine_size | acceleration | turbo
-------+-------+-------------+-------------+--------------+-------
   bmw |  220d |        rear |        2000 |          6.9 |  True

(1 rows)
```

BIGDATA TEAM

Удаление данных

```
cqlsh> DELETE FROM test.cars3 WHERE brand = 'bmw';
InvalidRequest: Error from server: code=2200 [Invalid query] message="Some partition key pa
rts are missing: model"
cqlsh> DELETE FROM test.cars3 WHERE brand = 'bmw' and model = '220d';
cqlsh> DELETE FROM test.cars3 WHERE brand = 'bmw' and model = '220d' and drive_wheel = 'rea
r';
cqlsh> DELETE FROM test.cars3 WHERE brand = 'bmw' and model = '220d' and drive_wheel = 'rea
r' and engine_size = 2000;
```

Cassandra Query Language

Выводы

Partition keys (Pk) определяют расположение данных в кластере

Cluster keys (Ck) определяют расположение данных внутри партиции

Композитный ключ строки - это набор (Pk, Ck)

non-EQ restriction применим только для последнего Ck

Все ключи, кроме последнего Ck, можно ограничивать только с помощью EQ restriction (=, in)

Нельзя пропускать ключи при фильтрации (слева направо)

Легковесные транзакции (LWT)

Атомарность

Атомарна ли данная операция?

```
ACC = SELECT acceleration FROM test.cars3 WHERE brand = 'bmw' and model = '220d';
INSERT INTO test.cars0 (brand, model, drive_wheel, engine_size, turbo, acceleration)
VALUES ('bmw', '220d', 'rear', 2000, true, ACC - 0.1);
```

Атомарность

Атомарна ли данная операция? (псевдокод)

```
ACC = SELECT acceleration FROM test.cars3 WHERE brand = 'bmw' and model = '220d';
INSERT INTO test.cars0 (brand, model, drive_wheel, engine_size, turbo, acceleration)
VALUES ('bmw', '220d', 'rear', 2000, true, ACC - 0.1);
```

Нет. Поскольку в Cassandra нет ACID блокировок, возможна ситуация, когда между нашим чтением и записью кто-то другой модифицирует данные в этой строке. В этом случае мы потеряем данные и никогда не узнаем об этом.

BIGDATA TEAM

Пример UPDATE

```
cqlsh> UPDATE test.cars3 SET acceleration = 6.9 WHERE brand = 'bmw' and model = '220d' and driv
e_wheel = 'rear' and engine_size = 2000;
cqlsh> SELECT * FROM test.cars3;

 brand | model | drive_wheel | engine_size | acceleration | turbo
-------+-------+-------------+-------------+--------------+-------
   bmw |  220d |        rear |        2000 |          6.9 |  True
```

По аналогии с предыдущим примером, здесь мы также можем потерять данные

LWT UPDATE

```
cqlsh> INSERT INTO test.cars3 (brand, model, drive_wheel, engine_size, turbo, acceleration) VAL
UES ('bmw', '220d', 'rear', 2000, true, 7.0) IF NOT EXISTS;

 [applied] | brand | model | drive_wheel | engine_size | acceleration | turbo
-----------+-------+-------+-------------+-------------+--------------+-------
     False |   bmw |  220d |        rear |        2000 |          6.9 |  True

cqlsh> SELECT * FROM test.cars3;

 brand | model | drive_wheel | engine_size | acceleration | turbo
-------+-------+-------------+-------------+--------------+-------
   bmw |  220d |        rear |        2000 |          6.9 |  True
```

LWT UPDATE

```
cqlsh> UPDATE test.cars3 SET acceleration = 6.9 WHERE brand = 'bmw' and model = '220d' and driv
e_wheel = 'rear' and engine_size = 2000 IF EXISTS;

 [applied]
-----------
      True

cqlsh> SELECT * FROM test.cars3;

 brand | model | drive_wheel | engine_size | acceleration | turbo
-------+-------+-------------+-------------+--------------+-------
   bmw |  220d |        rear |        2000 |          6.9 |  True
```

LWT UPDATE

```
cqlsh> UPDATE test.cars3 SET acceleration = 6.9 WHERE brand = 'bmw' and model = '220d' and driv
e_wheel = 'rear' and engine_size = 2000 IF acceleration = 7.0;

 [applied] | acceleration
-----------+--------------
     False |          6.9
```

```
cqlsh> UPDATE test.cars3 SET acceleration = 6.8 WHERE brand = 'bmw' and model = '220d' and driv
e_wheel = 'rear' and engine_size = 2000 IF acceleration = 6.9;

 [applied]
-----------
      True
```

Counter tables

Counter tables

```
cqlsh> UPDATE test.cars3 SET acceleration = acceleration - 0.1 WHERE brand = 'bmw' and model =
'220d' and drive_wheel = 'rear' and engine_size = 2000;
InvalidRequest: Error from server: code=2200 [Invalid query] message="Invalid operation (accele
ration = acceleration - 0.1) for non counter column acceleration"
```

```
cqlsh> CREATE TABLE IF NOT EXISTS test.cars4 (
   ...     brand text,    model text,
   ...     engine_size int,
   ...     drive_wheel text,
   ...     acceleration counter, PRIMARY KEY ((brand, model), drive_wheel, engine_size));
cqlsh>
```

Counter tables

```
cqlsh> UPDATE test.cars4 SET acceleration = acceleration + 70 WHERE brand = 'bmw' and model = '
220d' and drive_wheel = 'rear' and engine_size = 2000;
cqlsh> SELECT * FROM test.cars4;

 brand | model | drive_wheel | engine_size | acceleration
-------+-------+-------------+-------------+--------------
   bmw |  220d |        rear |        2000 |           70

(1 rows)
cqlsh> UPDATE test.cars4 SET acceleration = acceleration - 1 WHERE brand = 'bmw' and model = '2
20d' and drive_wheel = 'rear' and engine_size = 2000;
cqlsh> SELECT * FROM test.cars4;

 brand | model | drive_wheel | engine_size | acceleration
-------+-------+-------------+-------------+--------------
   bmw |  220d |        rear |        2000 |           69
```

LWT and Counter tables

Выводы

В классических РБД при проектировании таблиц и схем данных мы исходим из нормальных форм и стараемся избегать дублирования данных

В Cassandra в основе всего лежат запросы к данным - мы оптимизируем схему каждой таблицы для оптимальной работы запросов и часто прибегаем к денормализации данных

LWT и **Counter tables** позволяют обеспечить атомарность некоторых операций, но имеют дополнительные накладные расходы

Уровни согласованности

Уровни согласованности

Высокая доступность данных в Cassandra обеспечивается за счет **репликации**

Количество реплик таблицы задается на уровне **keyspace** (базы данных)

Чтение и запись данных возможна с разными уровнями согласованности

Это позволяет работать с данными разной критичности, начиная с **“эти данные мне не особо то и нужны, но пусть запишуться хоть как нибудь”** до **“это business critical данные, cassandra должна мне дать 100% гарантию, что они записаны во все реплики”**

BIGDATA TEAM

Read consistency levels

```
cqlsh> CONSISTENCY
               ALL           LOCAL_ONE     ONE           THREE
;              ANY           LOCAL_QUORUM  QUORUM        TWO
<enter>        EACH_QUORUM   LOCAL_SERIAL  SERIAL
cqlsh> CONSISTENCY ALL;
Consistency level set to ALL.
```

ALL - все реплики должны вернуть одинаковые данные

QUORUM - *(rep_factor/2) + 1* реплик должно вернуть данные последней версии

ONE - берется ответ ближайшей реплики

BIGDATA TEAM

Write consistency levels

```
cqlsh> CONSISTENCY
               ALL            LOCAL_ONE     ONE            THREE
;              ANY            LOCAL_QUORUM  QUORUM         TWO
<enter>        EACH_QUORUM    LOCAL_SERIAL  SERIAL
cqlsh> CONSISTENCY ALL;
Consistency level set to ALL.
```

ALL - все реплики должны подтвердить запись

QUORUM - *(rep_factor/2) + 1* реплик должны подтвердить запись

ONE - любая реплика должна подтвердить запись

ANY (мой любимый) - любая нода должна подтвердить запись

Spark Cassandra connector

Spark Cassandra Connector

https://github.com/datastax/spark-cassandra-connector/blob/master/doc/15_python.md

```
spark.read\
    .format("org.apache.spark.sql.cassandra")\
    .options(table="kv", keyspace="test")\
    .load().show()
```

```
df.write\
    .format("org.apache.spark.sql.cassandra")\
    .mode('append')\
    .options(table="kv", keyspace="test")\
    .save()
```

Настройка

Зависимости берем из https://mvnrepository.com/artifact/com.datastax.spark/spark-cassandra-connector

И добавляем в скрипт spark-submit:

```
./bin/pyspark \
  --packages com.datastax.spark:spark-cassandra-connector_2.11:2.3.2
```

Настройка

Параметры конфигурации доступны здесь:

https://github.com/datastax/spark-cassandra-connector/blob/master/doc/reference.md

Наиболее важные:

```
spark.cassandra.connection.host
spark.cassandra.auth.username
spark.cassandra.auth.password
spark.cassandra.output.consistency.level
spark.cassandra.input.consistency.level
```

Плюсы и минусы

Плюсы

- поддерживает predicate-pushdown для фильтров и соединений
- позволяет быстро писать и читать

Минусы

- плохо переживает недоступность узлов и сетевые сбои
- не поддерживает LWT и Counter tables
- может уходить в состояние неопределенности
- легко ошибиться и прочитать всю таблицу

Workshop

Thank you! Questions?

Feedback:
http://rebrand.ly/mf2019q2_feedback_11_cassandraspark

Andrey Titov, andrey.titov@bigdatateam.org
Big Data Instructor @ BigData Team, http://bigdatateam.org/
Senior Spark Engineer @ NVIDIA