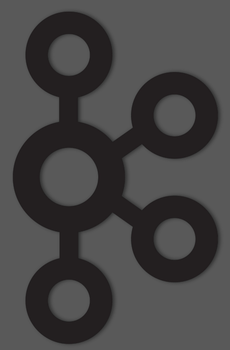
Вставить в ClickHouse и не умереть...

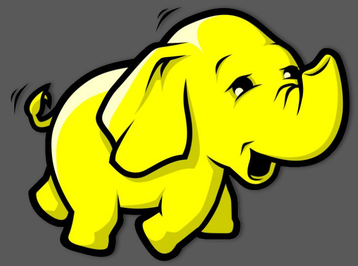
Артём Шутак
Ведущий разработчик, OK.RU

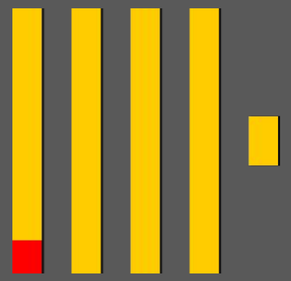
Обо мне

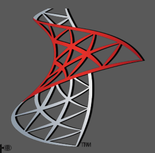
- Артём Шутак
- 10 лет в IT: инженер и архитектор
- 7 лет в распределенных системах и BigData
- 4 года в Grid Dynamics: инженер => архитектор
- Контрибьютил в Apache Ignite и Apache Griffin
- ~1 год в OK.RU, команда Data Platform

Data Platform









samza

And more...

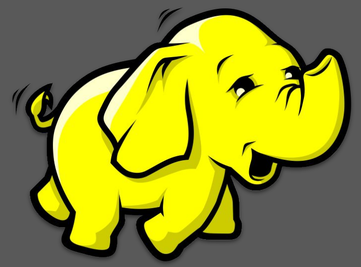
Data Platform

17 Kafka кластеров

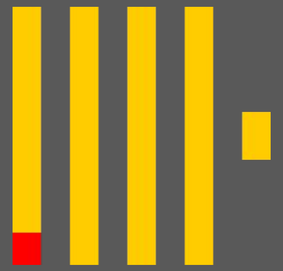
Самый большой:
180 серверов, 540 TB

Compute
2k+ batch jobs / день

(не только Spark)



150PB HDD
+ 35 PB / год
+ 10TB «сырых» / день



Вставляем в 72 сервера

- 1,5 TB / день
- **50 млрд** строк / день

HDD : 150/ **600** TB
NVME: 60 / 210 TB
RAM : 5.5 TB

О чем поговорим

- ClickHouse и вставка в него
- Рассмотрим 3 неожиданных сценария
- Немного заглянем «под капот»
- Еще будет
  - Kafka
  - Spark, но чуть-чуть

Ситуация 1

Непредсказуемые результаты

Вставляем и читаем

```
CREATE TABLE test_table (
  `a` Int8,
  `b` String) …
```

```
INSERT INTO test_table (a,b)
VALUES (239, 'my value')

INSERT INTO test_table (a,b)
VALUES (239, 'my value')
```

```
SELECT count() FROM test_table
```

Какой результат?

- 0
- 1
- 2
- 4

1

Вставляем и читаем

```
CREATE TABLE test_table (
  `a` Int8,
  `b` String)
ENGINE = ReplicatedMergeTree()

INSERT INTO test_table (a,b)
VALUES (239, 'my value')

INSERT INTO test_table (a,b)
VALUES (239, 'my value')


SELECT count() FROM test_table
```

Вставляем и читаем

```
CREATE TABLE test_table (
  `a` Int8,
  `b` String)
ENGINE = MergeTree()

INSERT INTO test_table (a,b)
VALUES (239, 'my value')

INSERT INTO test_table (a,b)
VALUES (239, 'my value')


SELECT count() FROM test_table
```

Какой результат?

- 0
- 1
- 2
- 4

2

*MergeTree «на пальцах»

- Базовый движок в ClickHouse
- Для «Аналитических» запросов
- Основан на LSM-Tree
  - Но не полностью его реализует
- Позволяет много писать и читать
- Колоночный формат хранения

Документация ClickHouse

MergeTree Engine Family

Table engines from the MergeTree family are the core of ClickHouse data storage capabilities. They provide most features for resilience and high-performance data retrieval: columnar storage, custom partitioning, sparse primary index, secondary data-skipping indexes, etc.

Base MergeTree table engine can be considered the default table engine for single-node ClickHouse instances because it is versatile and practical for a wide range of use cases.

For production usage ReplicatedMergeTree is the way to go, because it adds high-availability to all features of regular MergeTree engine. A bonus is automatic data deduplication on data ingestion, so the software can safely retry if there was some network issue during insert.

Документация ClickHouse

MergeTree Engine Family

Table engines from the MergeTree family are the core of ClickHouse data storage capabilities. They provide most features for resilience and high-performance data retrieval: columnar storage, custom partitioning, sparse primary index, secondary data-skipping indexes, etc.

Base MergeTree table engine can be considered the default table engine for single-node ClickHouse instances because it is versatile and practical for a wide range of use cases.

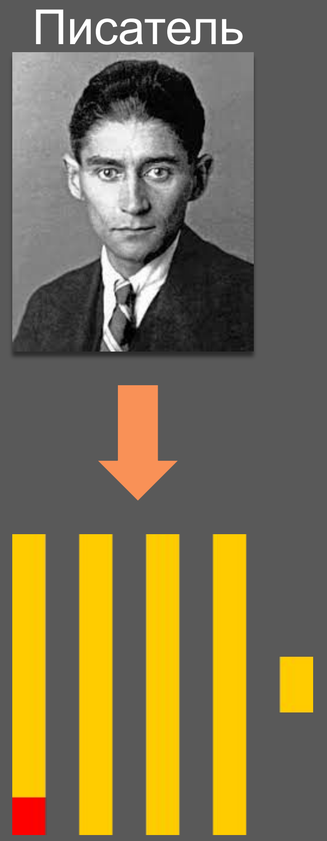
For production usage ReplicatedMergeTree is the way to go, because it adds high-availability to all features of regular MergeTree engine. A bonus is automatic data deduplication on data ingestion, so the software can safely retry if there was some network issue during insert.

Мотивация к дедупликации

Мотивация: Fail-over и Exactly-Once семантика

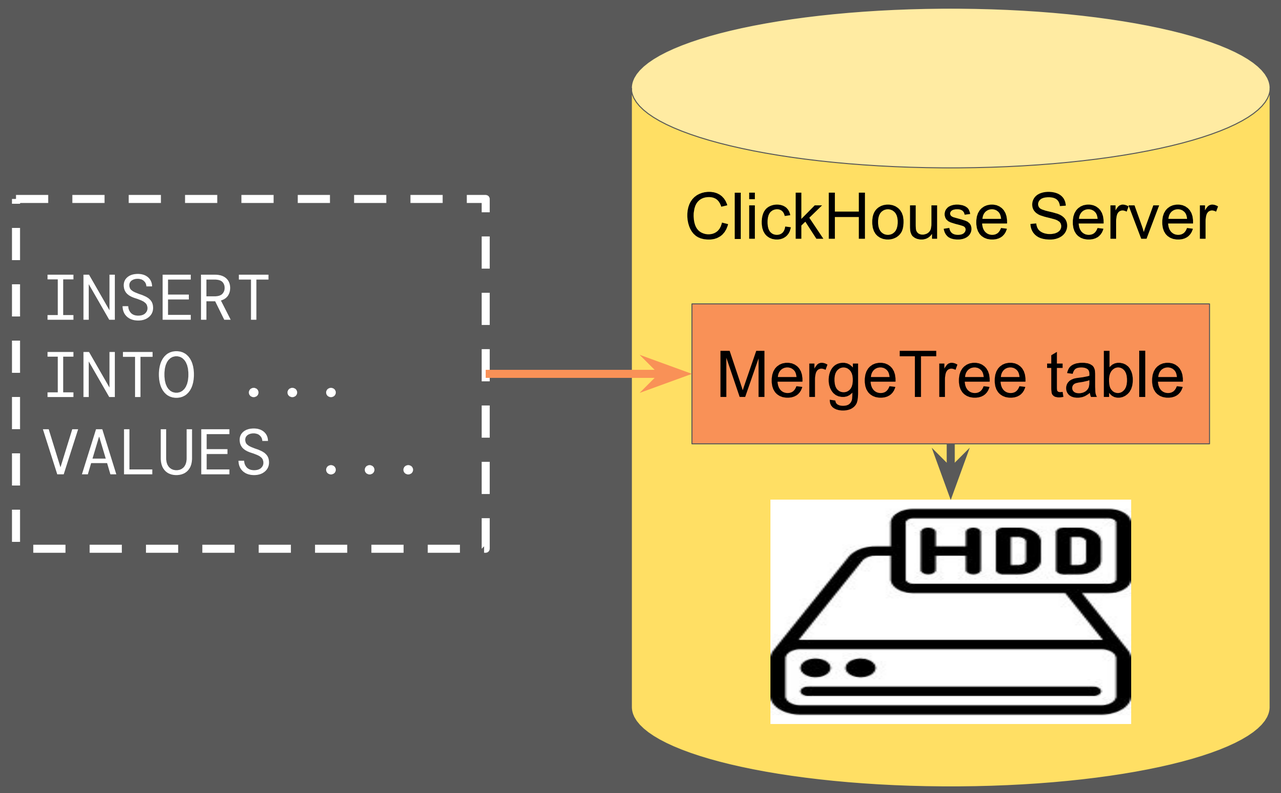
Пример: процессу надо вставить данные в ClickHouse

1. Успешно записывает данные
2. Падает, не успев отметить данные, как вставленные
3. В следующий раз записывает данные снова
4. => Дубликаты!!!



Дедупликация в ReplicatedMergeTree

Вставка в *MergeTree: batch-и и блоки



- Каждая вставка сразу идет на диск
  - => Запись большими **batch**-ами
- Большие batch-и разделяются на **блоки**

Вспомним изначальный пример

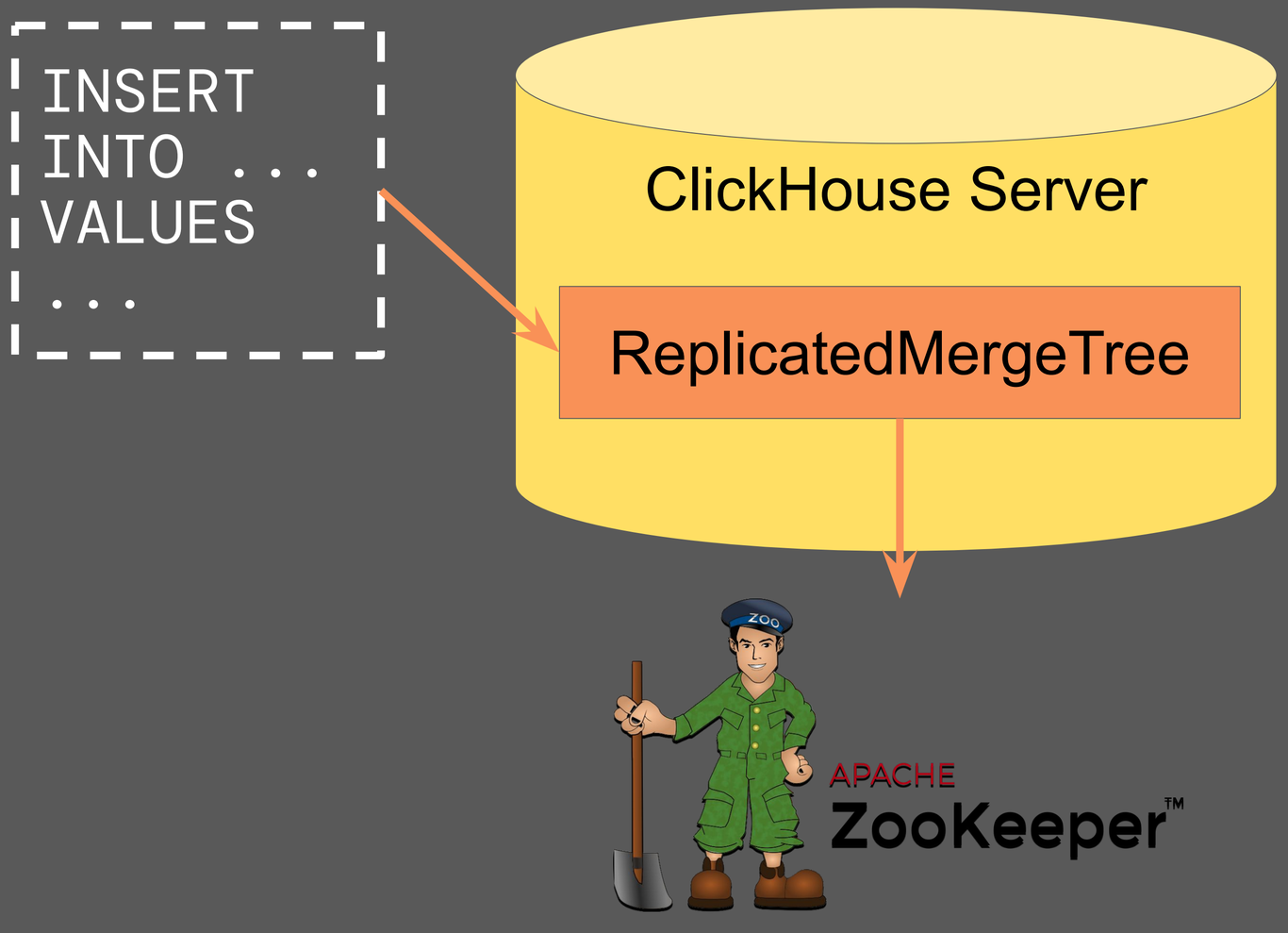
```
CREATE TABLE test_table (
  `a` Int8,
  `b` String)
ENGINE = ReplicatedMergeTree()

INSERT INTO test_table (a,b)
VALUES (239, 'my value')

INSERT INTO test_table (a,b)
VALUES (239, 'my value')

SELECT count() FROM test_table
```

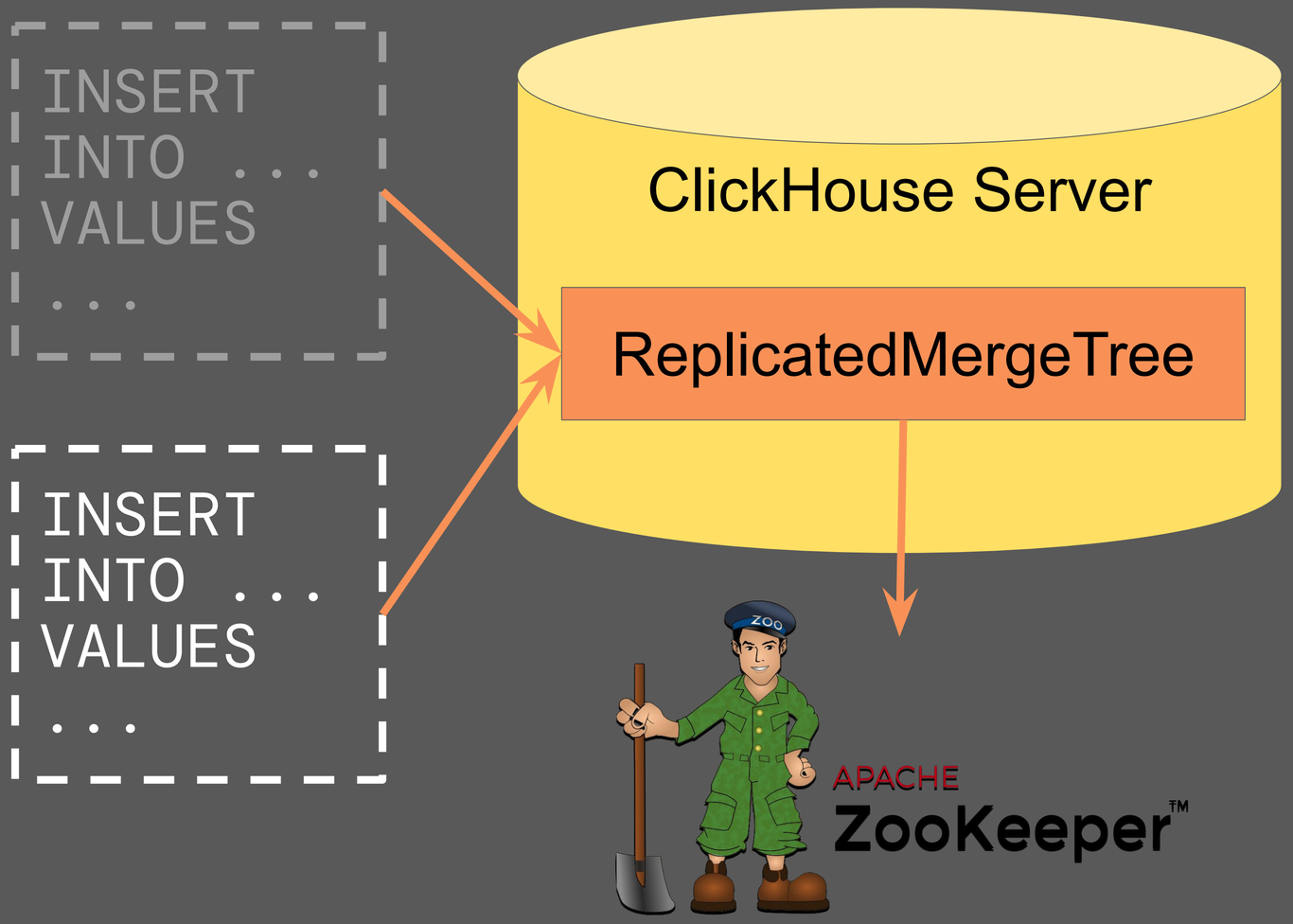
ReplicatedMergeTree: первая вставка



ClickHouse

1. Считает **«хеш»** для блока
2. Сохраняет в Zookeeper

ReplicatedMergeTree: вторая вставка



ClickHouse

1. Считает «**хеш**» для блока
2. Находит «**хеш**» в Zookeeper
3. => Пропускает вторую вставку

Дедупликация: конфигурация

*ReplicatedMergeTree

- insert_deduplicate: 0 or 1
  - **Включена** по умолчанию!
- replicated_deduplication_window
  - Количественный лимит на хранение хеш-сумм
  - Хеш-суммы хранятся в Zookeeper
  - 100 по умолчанию
- replicated_deduplication_window_seconds
  - Лимит на хранение хеш-сумм, но в секундах
  - Неделя по умолчанию

Можно включить для MergeTree (без репликации)

- **Выключена** по умолчанию
- **Свои опции**

Дедупликация в примерах

Что вернет ‘select count()’?

```
INSERT INTO test_table (a,b)
VALUES (239, 'my value');

INSERT INTO test_table (a,b)
VALUES (1, 'another value');

INSERT INTO test_table (a,b)
VALUES (239, 'my value');
```

2

Что вернет ‘select count()’?

```
INSERT INTO test_table (a,b)
VALUES  (239,'my value'),
        (1,  'another value');

INSERT INTO test_table (a,b)
VALUES  (1,  'another value'),
        (239,'my value');
```

2

What does ‘select count()’ return?

```
INSERT INTO test_table (a,b)
VALUES (239, 'my value');
```

2 недели

2

```
INSERT INTO test_table (a,b)
VALUES (239, 'my value');
```

Выводы

insert_deduplicate

- Exactly-Once семантика на вставке
  - (Если знаешь как)
  - Классная и нужная фича!
- Поведение по умолчанию шокирует
  - + Разное поведение для разных движков

Ситуация 2

Когда вставка завершится?

Утренний диалог

Готовы ли данные за сегодня?

Да.

Аналитик данных

Инженер
Data Platform

Утренний диалог

Странно. 5 минут назад данных не было совсем. Теперь что-то есть, но результаты каждый раз разные.

Аналитик данных

ClickHouse “eventually consistent”. Это ожидаемо.

Инженер
Data Platform

Утренний диалог

...

Когда процесс вставки закончится?

...

Мне надо кое-что проверить.

Аналитик данных

Инженер Data Platform

3 дня спустя...

Наши раскопки

ClickHouse страдает

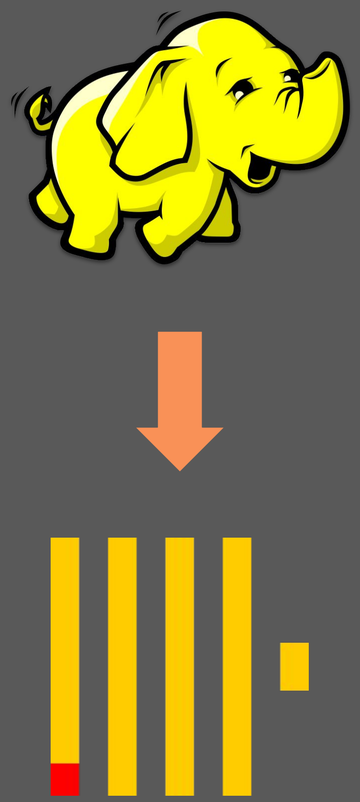
- Каждый день в 5-8 утра
- Высокое потребление CPU
- Высокий Disk IO
- DDL операции подвисают
- “Session expired” в ClickHouse логах

Zookeeper

- Высокое количество ZK ошибок
- Большое количество операций
- GC паузы

Давайте добавим немного контекста

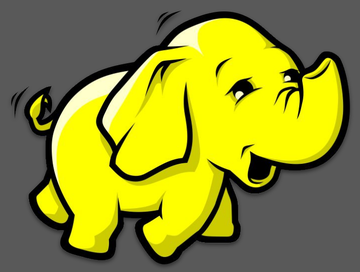
Batching: вставляем данные за день



```
INSERT INTO important_kpi
       (Date,  Timestamp,Key,Value)
VALUES (<today>, ...),
       (<today>, ...),
       …
```

Вставили 50 миллионов строк.
ClickHouse сообщил, что все успешно.

Batching: вставляем данные за день



```
SELECT count()
FROM important_kpi
WHERE Date = <today>
```

10 min

```
SELECT count() ...
```

30 min

```
SELECT count() ...
```

10 k

20 mln

40 mln

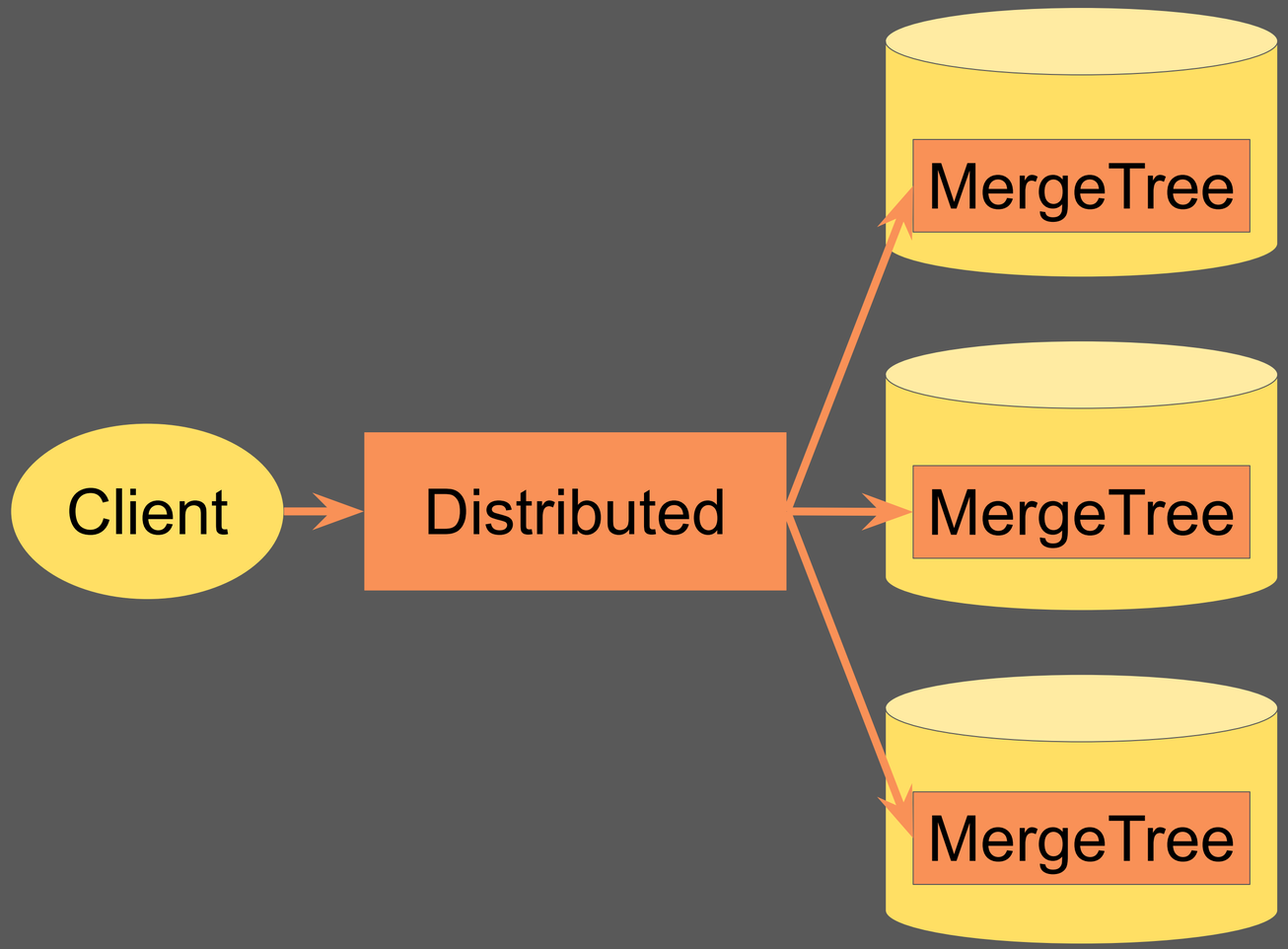
Расследование

```
INSERT INTO important_kpi
       (Date,  Timestamp,Key,Value)
VALUES (<today>, ...),
       (<today>, ...),
       …


SHOW CREATE TABLE important_kpi;

CREATE TABLE important_kpi ...
ENGINE = Distributed(...)
```

Distributed «на пальцах»



Проблема

- *MergeTree - на одном сервере
- Хотим горизонтально масштабироваться
- Заводим несколько машин
- Распределяем по ним данные
- Как будем теперь из них читать (ну и писать в них)?

Решение: Distributed

Distributed

Движок Distributed не хранит данные самостоятельно, а позволяет обрабатывать запросы распределённо, на нескольких серверах. Чтение автоматически распараллеливается. При чтении будут использованы индексы таблиц на удалённых серверах, если есть.

Движок Distributed принимает параметры:

- имя кластера в конфигурационном файле сервера
- имя удалённой базы данных
- имя удалённой таблицы
- (не обязательно) ключ шардирования.
- (не обязательно) имя политики, оно будет использоваться для хранения временных файлов для асинхронной отправки

Наши раскопки

ClickHouse страдает

- Каждый день в 5-8 утра
- Высокое потребление ЦПУ
- **Высокий Disk IO**
- DDL операции подвисают
- “Session expired” в ClickHouse логах

Zookeeper

- Высокое количество ZK ошибок
- Большое количество операций
- GC паузы

Давайте поизучаем, что у нас там на диске...

Высокий Disk IO

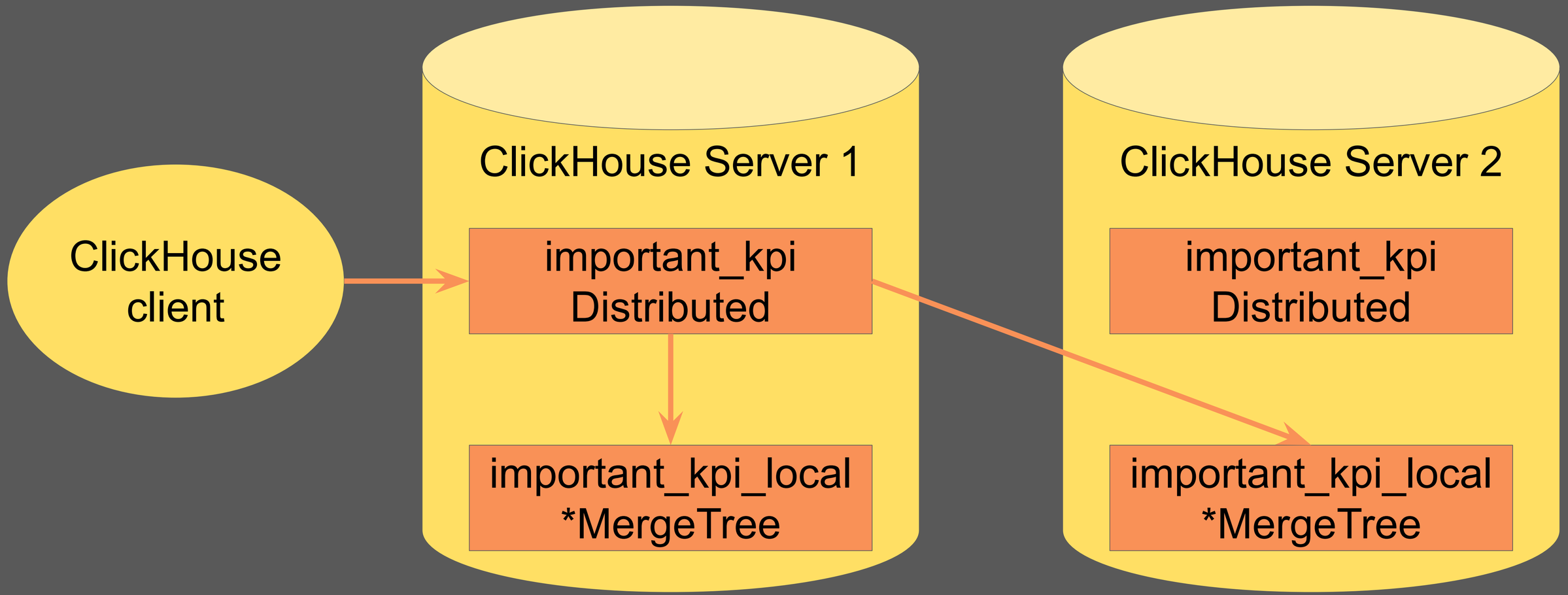
- Структура папок в ClickHouse

  `<data_dir>/<db>/<table>/<data_files>`
- Создается и для Distributed
- Там всегда пусто
  - (Ожидание из документации)
- Там не пусто!!!

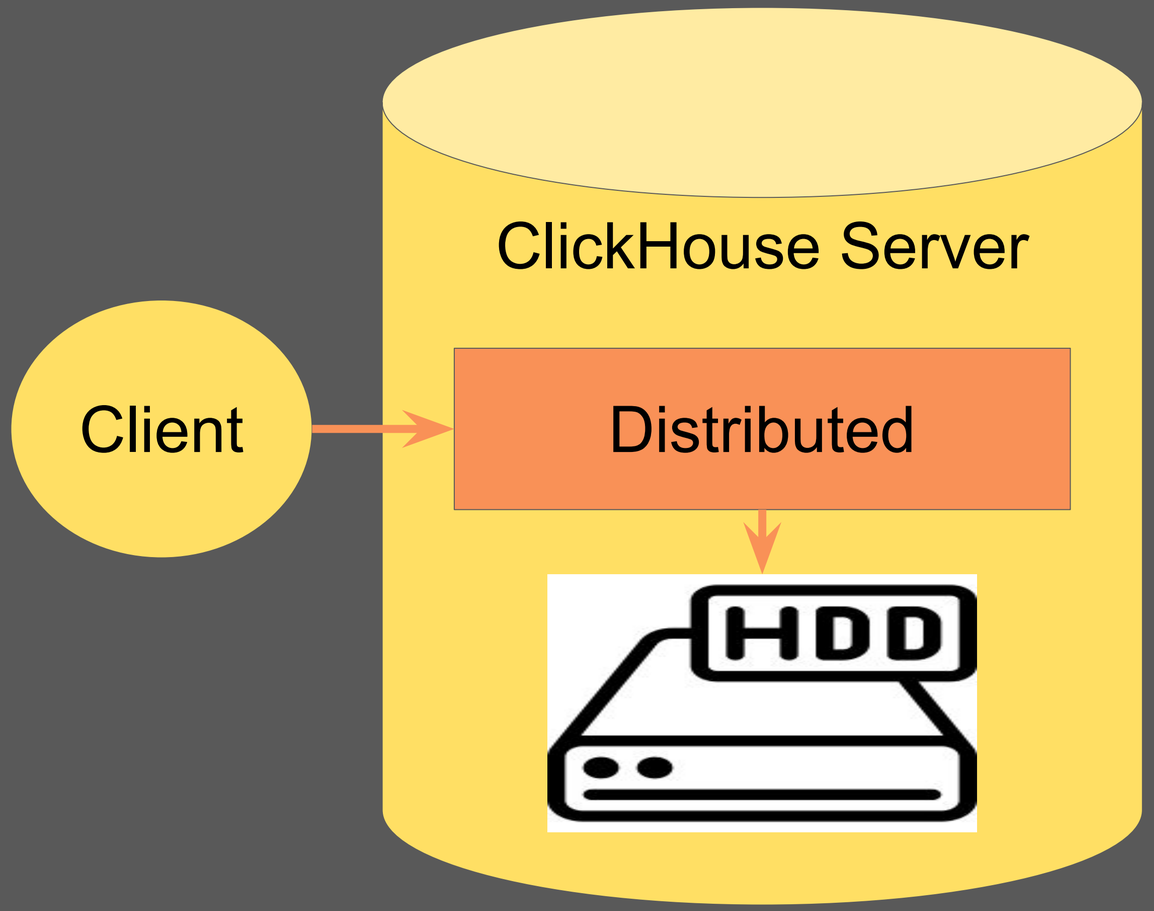
  `/var/lib/clickhouse/data/default/important_kpi/<some_files>`

Вставка в Distributed таблицу

ClickHouse Distributed Engine: ожидания

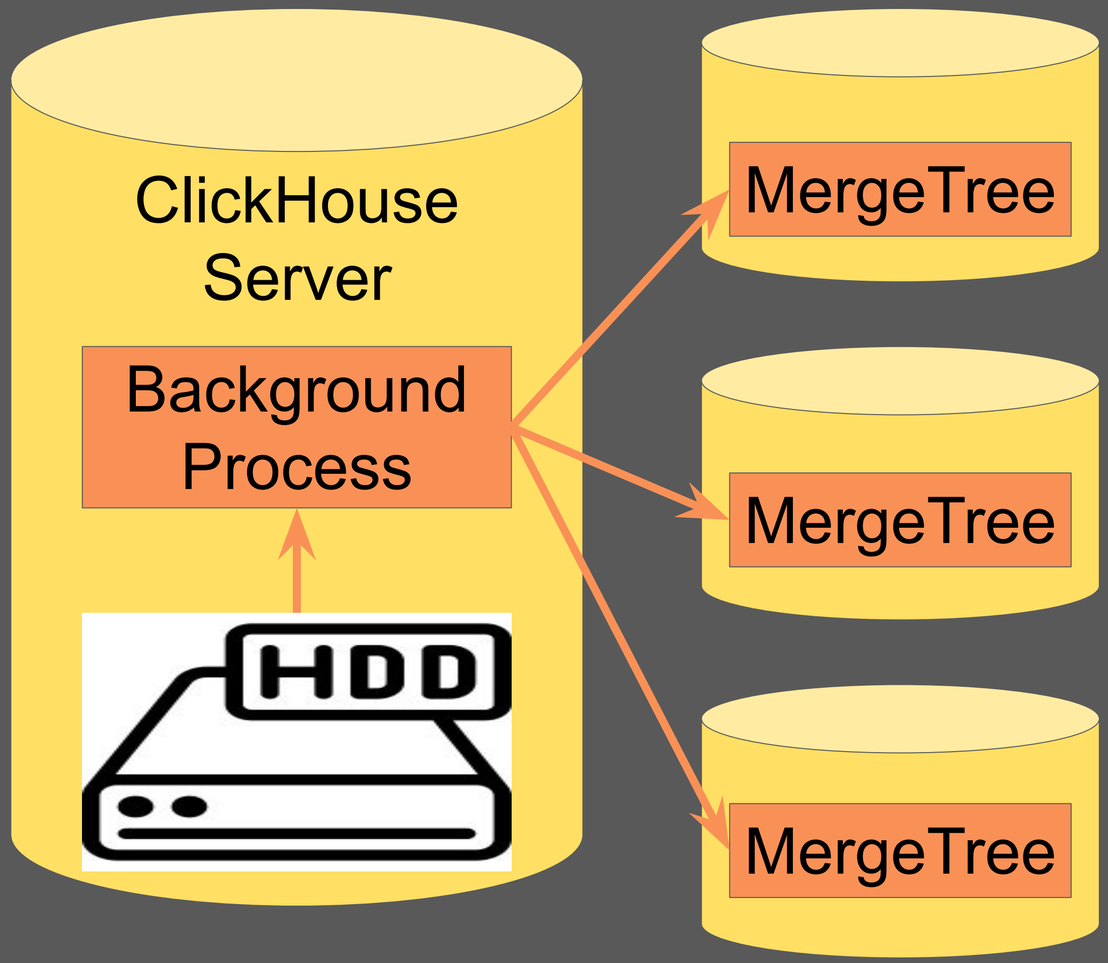


ClickHouse Distributed Engine: реальность 1



1. Клиент выбирает случайный сервер
2. Сервер сохраняет данные локально (на диск)
3. Клиент получает ответ “ok”

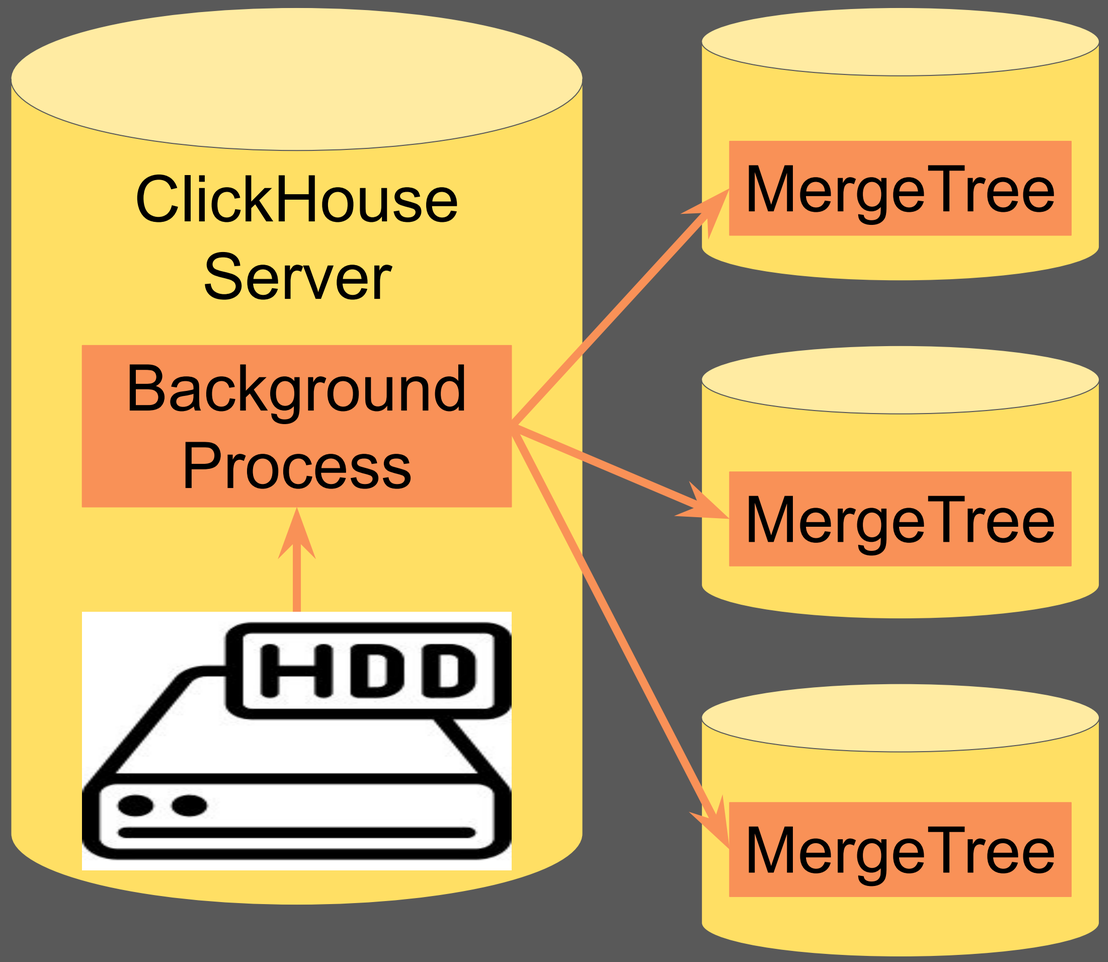
ClickHouse Distributed Engine: реальность 2



Background процесс

1. Читает данные с диска
2. «Парсит» Sharding Key
3. Вставляет с учетом Sharding Key

ClickHouse Distributed Engine: подводные камни



1. Eventual Consistency
2. Не является fault-tolerant
   a. Возможный data-loss
   b. (в доке есть)
3. Дополнительный Disk IO
4. «Парсинг» данных
   a. Извелкаем Sharding Key
5. Маленькие batch-и:
   200k / 20 = 10k

Что же делать?

Решение 1:
Конфигурация Distributed таблицы

- `insert_distributed_sync`
  - Синхронное добавление данных
  - Может привести к дубликатам
- insert_shard_id
  - Номер shard-а, в который вставлять
  - Можно указать для запроса
  - По сути «ручное» управление
- `insert_distributed_one_random_shard`
  - Вставка данных в случайный shard
  - Не идемпотентная операция

Решение 2:
Вставка в «локальную» таблицу «руками»

- Date - ключ шардирования
  - Вставляем целый день в один shard
  - shard_num = Date % number_of_shards
  - Только «маленькие» таблицы и большая история
- Сложный ключ шардирования и свои инструменты
  - Об этом позже

Выводы

Distributed Engine

Чтение

- Хорошо
- Можно было бы лучше
  - Если использовать информацию о ключе шардирования

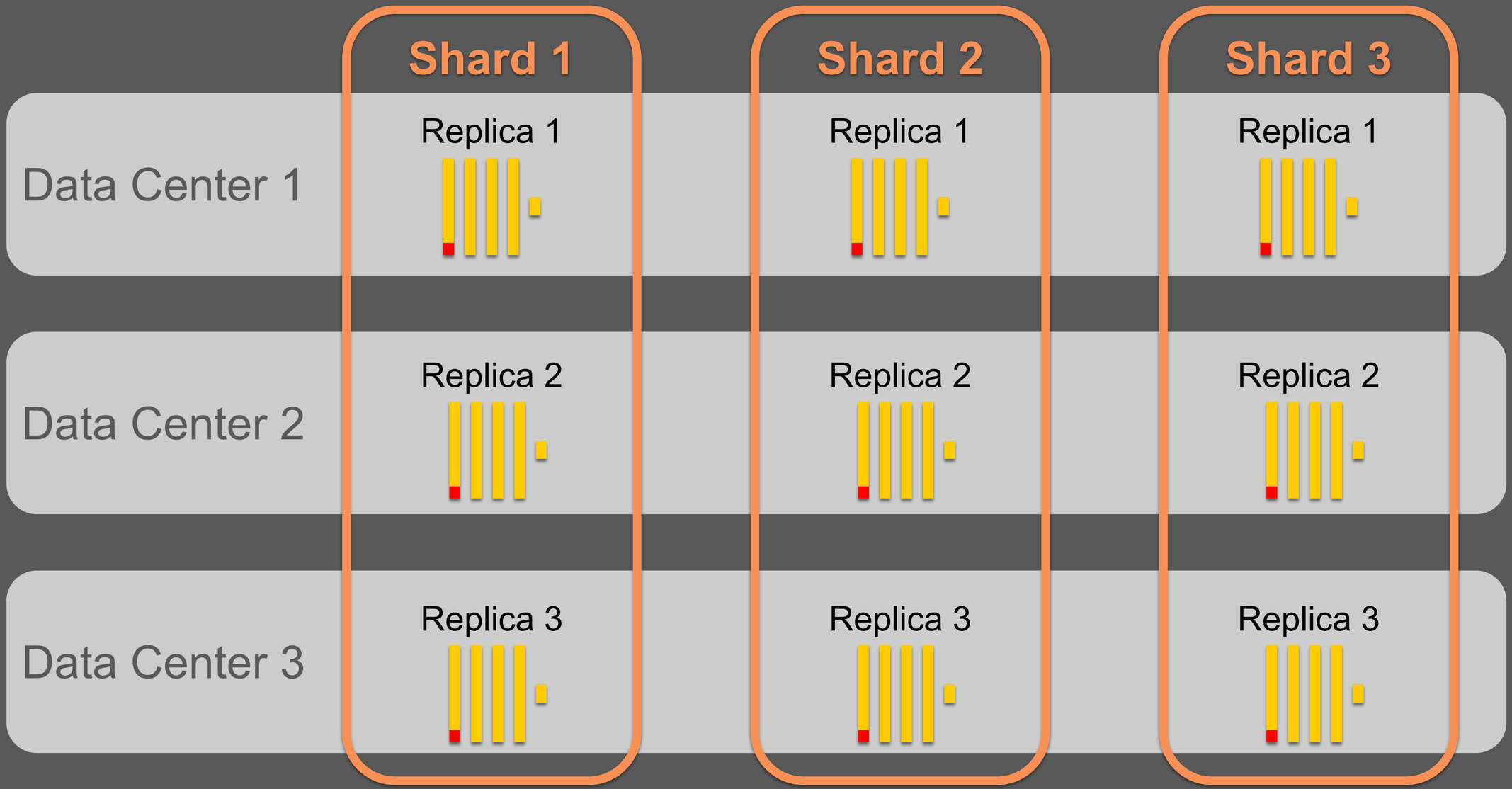
Запись

- Сложно и непрозрачно
- Сомнительное поведение по умолчанию
- Можно настроить, но не идеально
- В сложных случаях - все «руками»

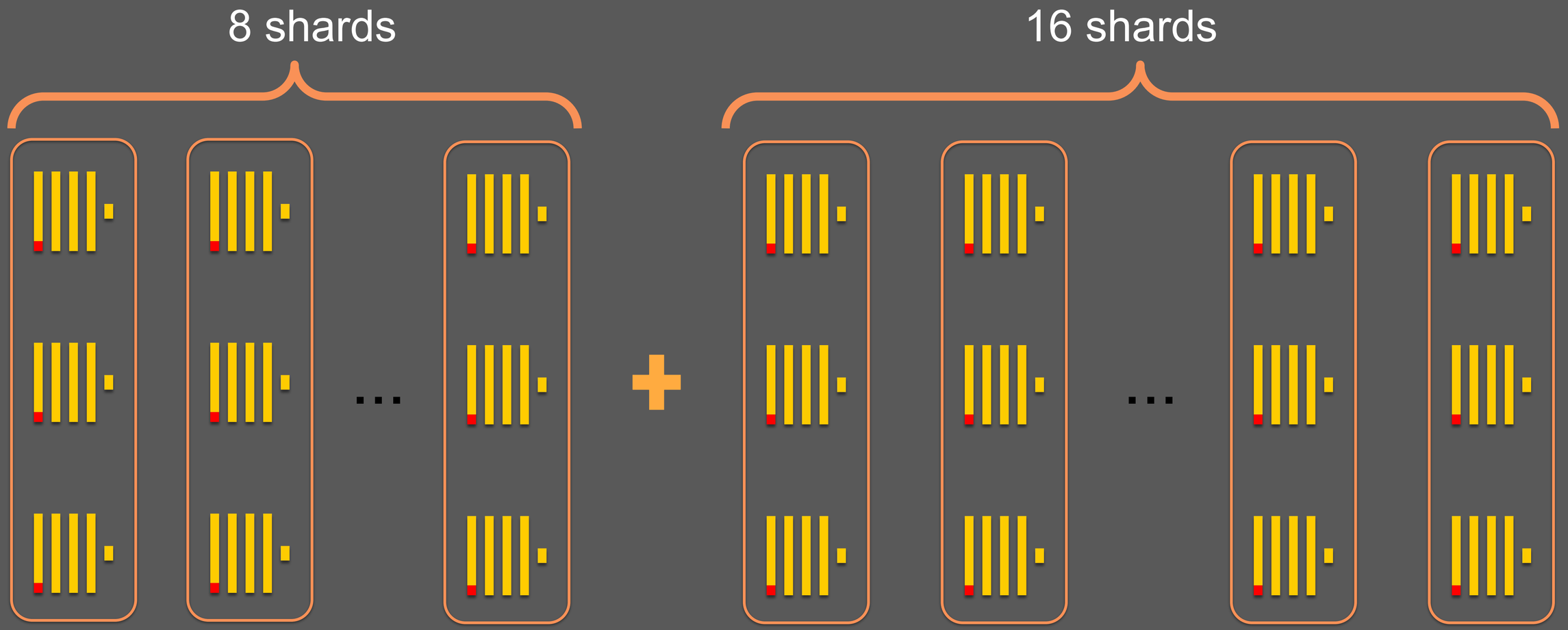
Ситуация 3

Увеличиваем размер кластера

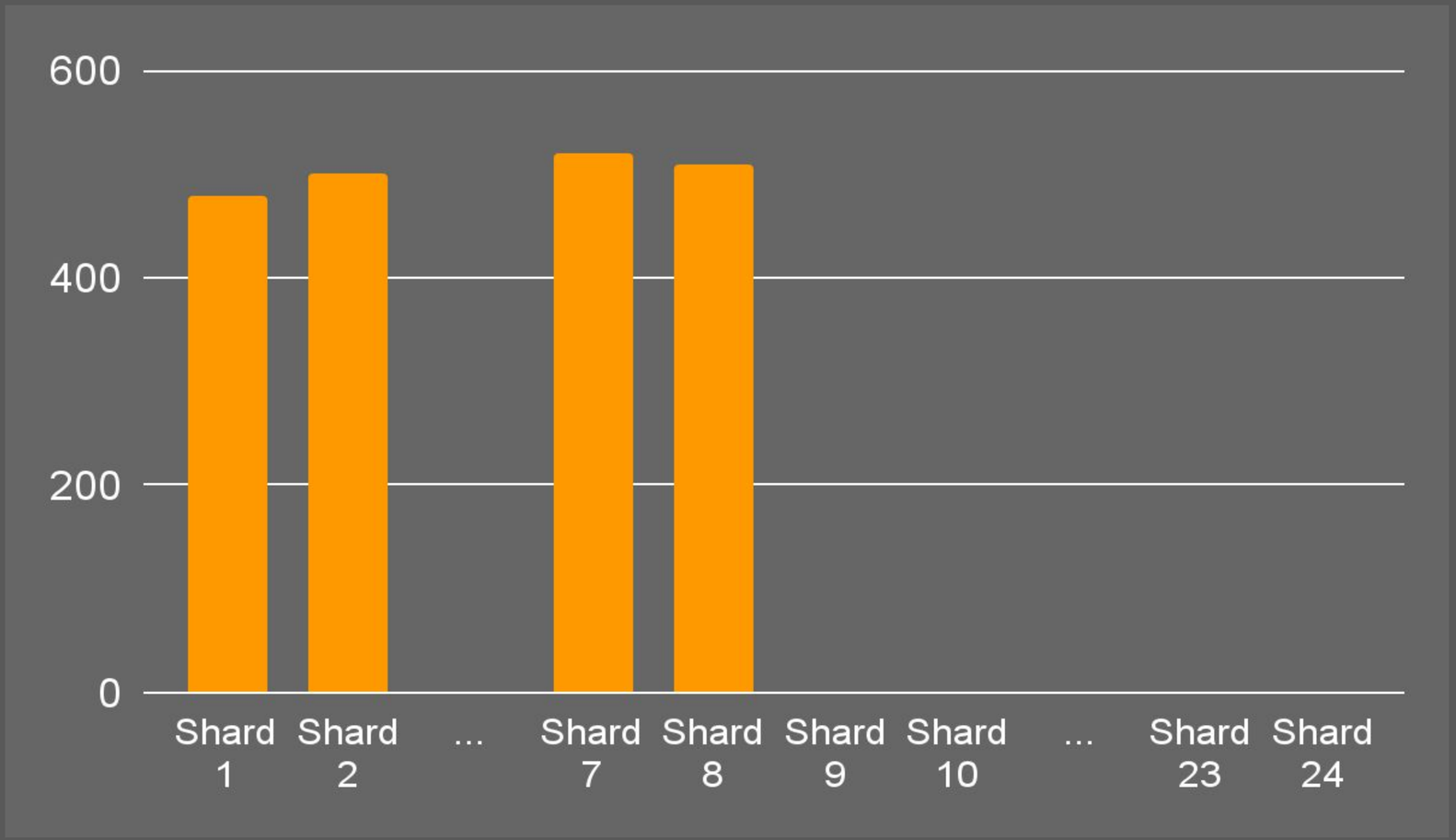
Наш ClickHouse: Топология и Гомогенность



Данные не вмещаются в кластер



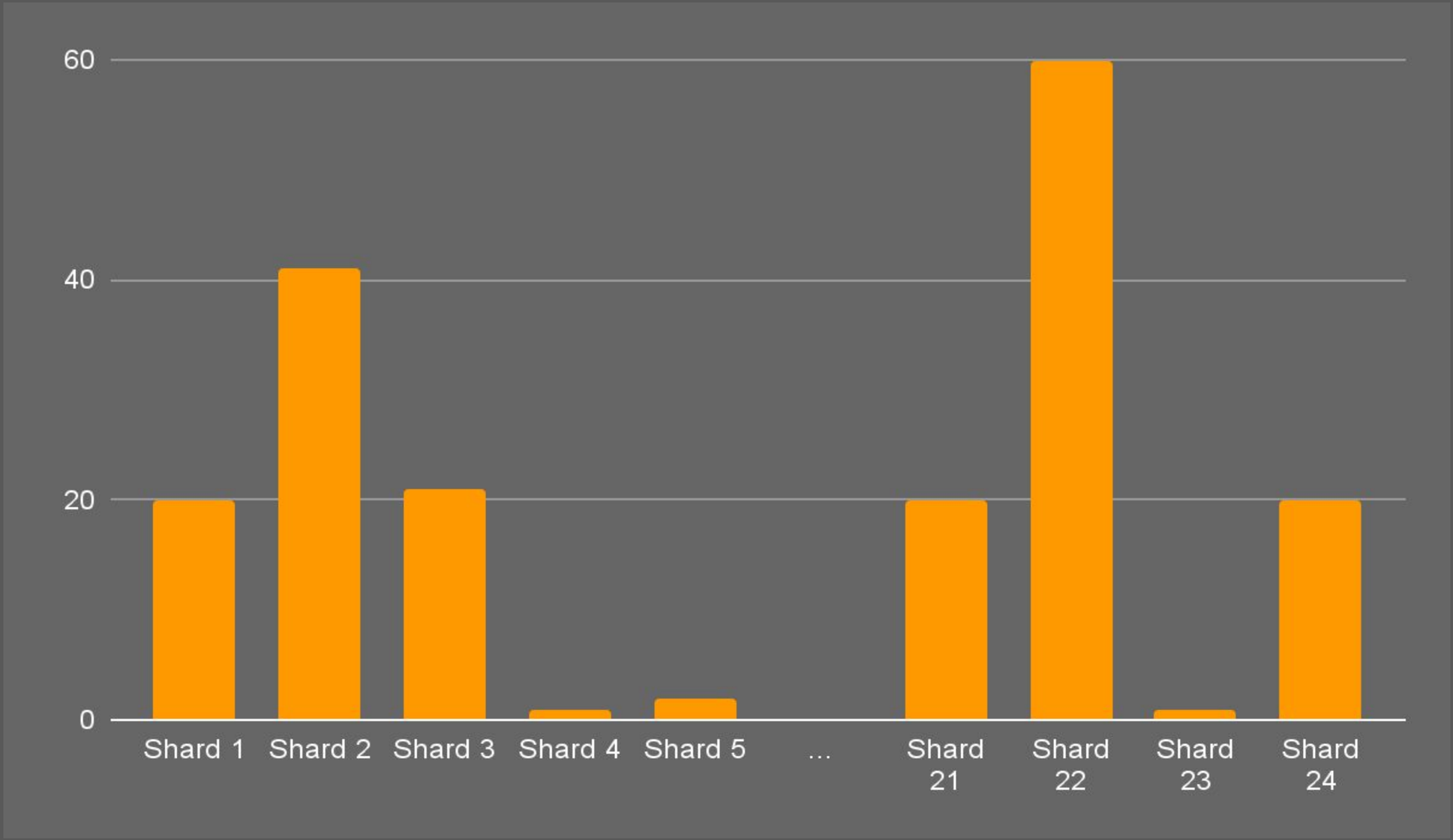
Количество данных на диске по shard-ам



Перекос в данных

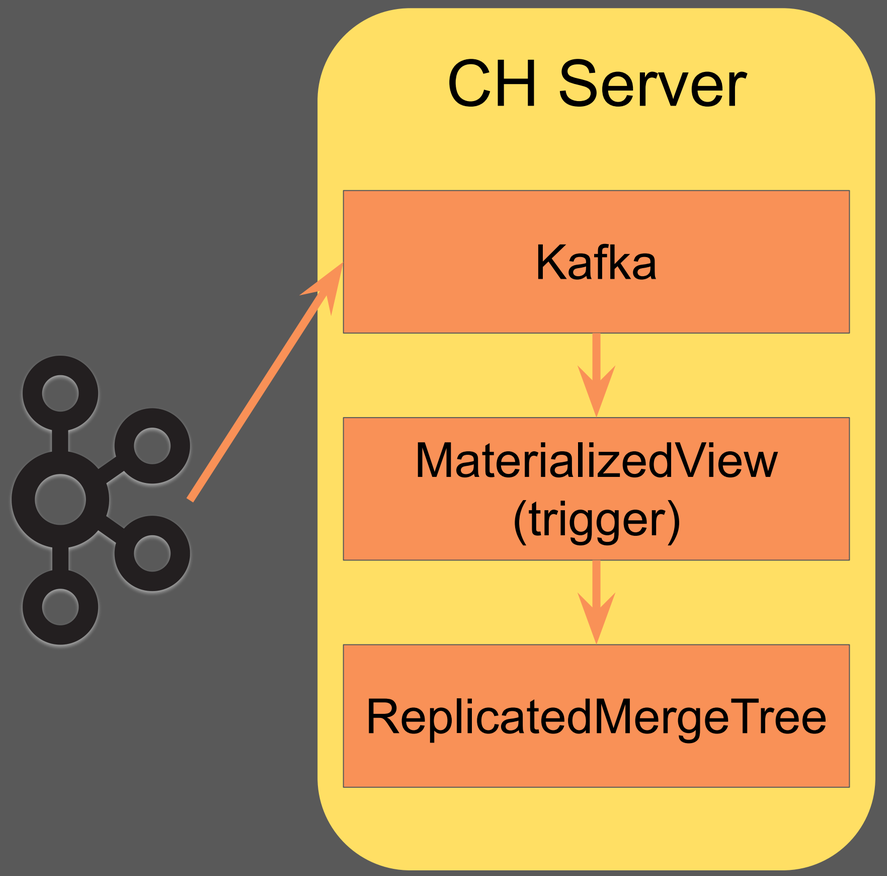
- Ожидаемо
- Нет инструментов для миграции
- Что делать с историей?
- Всё с ретеншеном
  - Готовы подождать

Прирост данных на диск по shard-ам в день

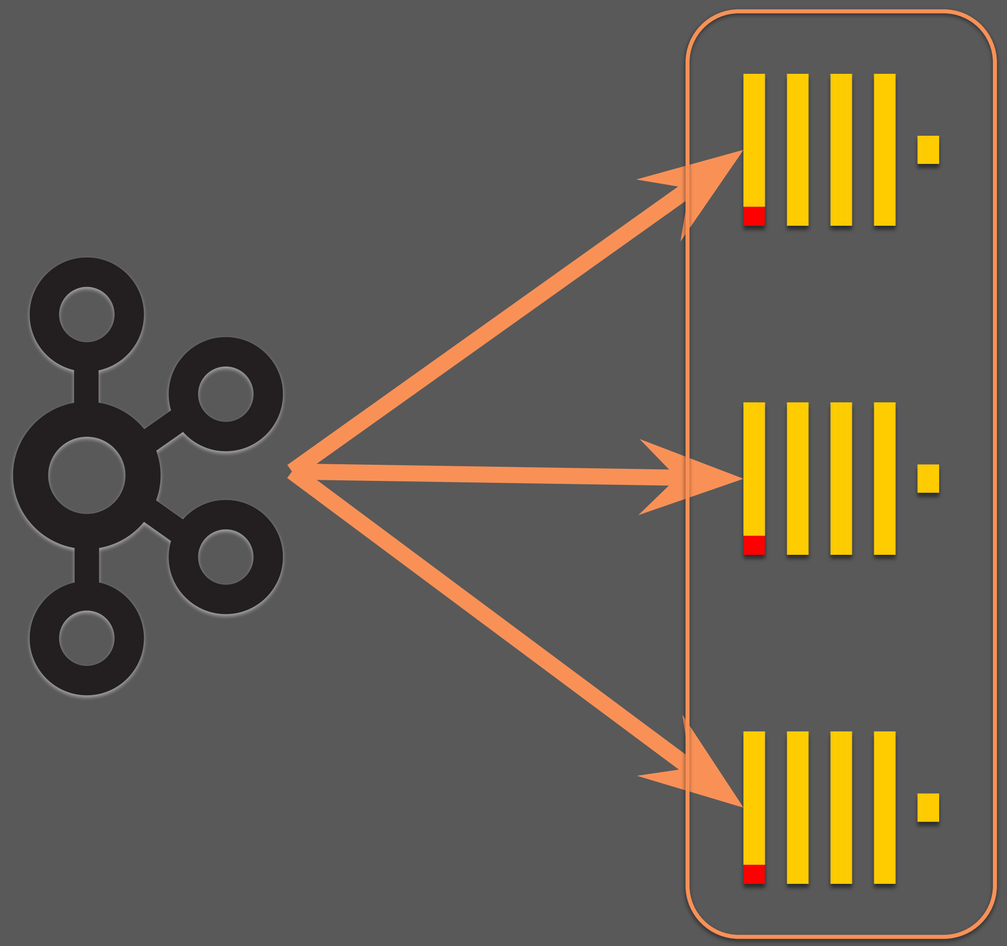


Давайте разбираться

- Посмотрим на «самую большую» таблицу
- Распределение как на диаграмме
  - x0 (совсем ничего), x1, x2, x3
- Вставка через Kafka Engine



ClickHouse Kafka Engine: Кластер с одним shard-ом

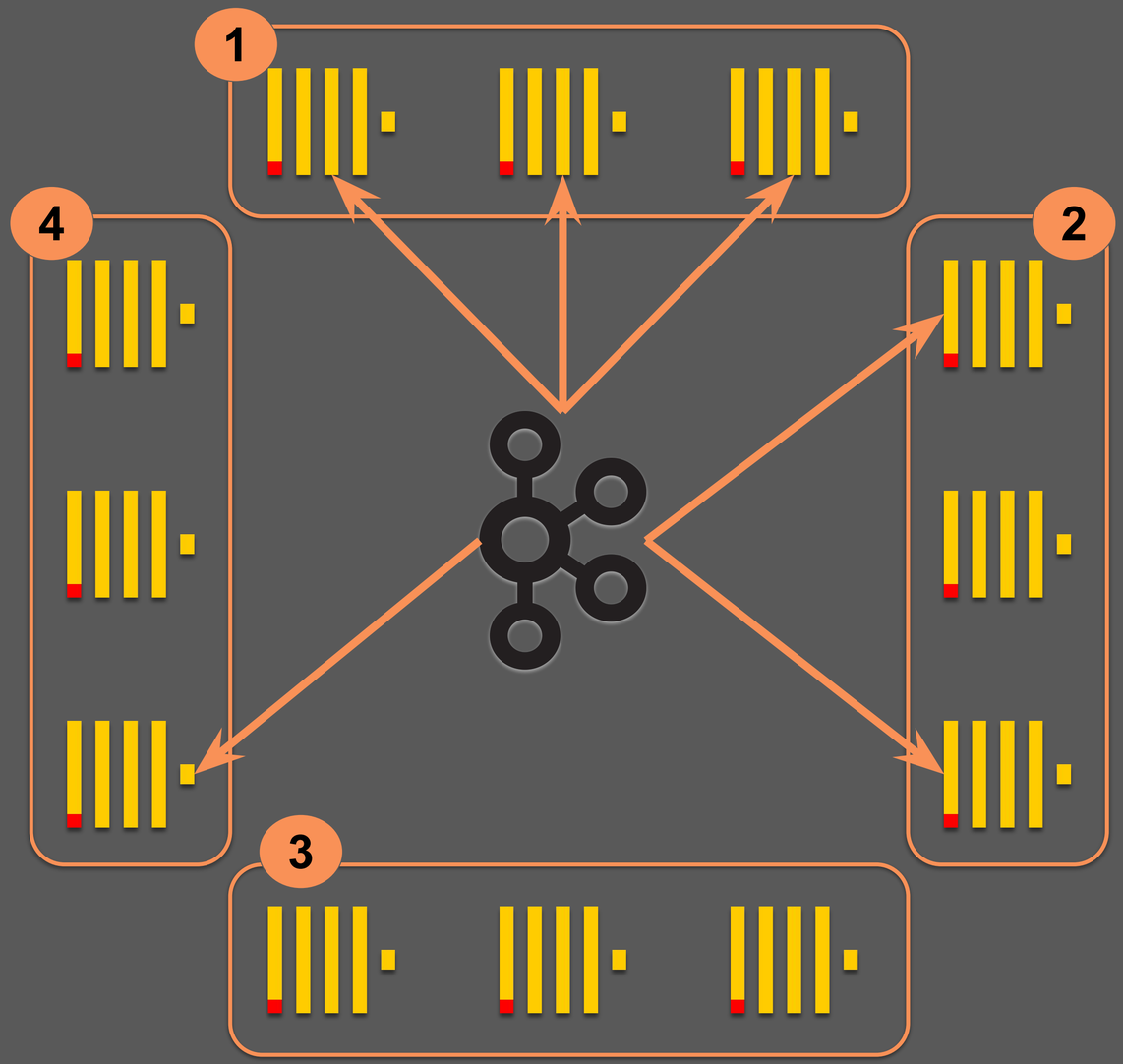


- Каждый сервер читает данные
- Одна Kafka consumer-группа
- ClickHouse не знает, кто и что читает

Пример: топик с 2 партициями

- CH Сервер 1 - Kafka partition 1
- CH Сервер 2 - Kafka partition 2
- CH Сервер 3 - не осталось партий
- ReplicatedMergeTree => ok

ClickHouse Kafka Engine: 4 shard-а в кластере

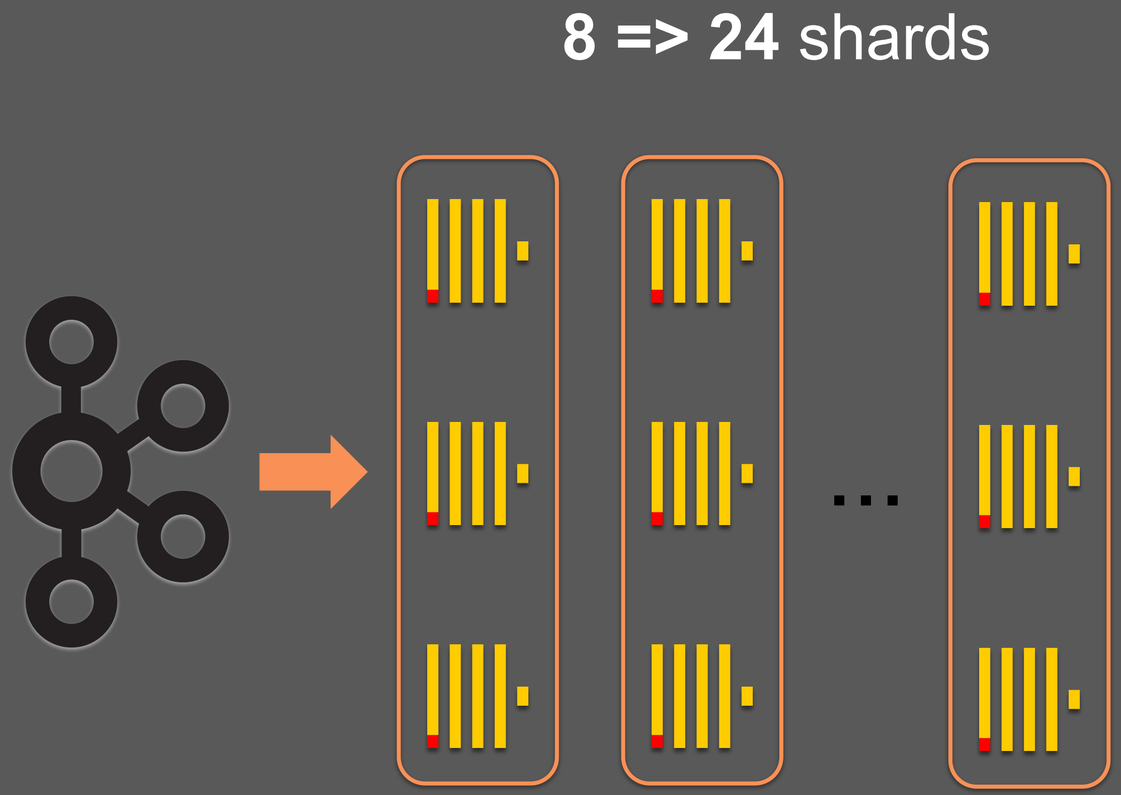


Example

- Kafka топик с 6 партициями
  - Партиции одинакового «размера»
- CH: 4 shard x 3 реплики (12 серверов)
- 12 consumer-ов и 6 партиций

Shard №	Volume
1	3x
2	2x
3	No data
4	1x

Наш случай



Проблема

- CH: 8*3 серверов => 24*3 серверов
- Kafka топик с **24** партициями
- **24 партиции и 72 consumer-а**
  - 1 сервер получает 0 или 1 партицию
  - 1 shard получает 0-3 партиции

Решение

- Kafka партиции: 24 => 72

«Транзитивные проблемы» © Data Platform

Транзитивная проблема 7 уровня...

1. Недостаточно места в ClickHouse
2. |-- Увеличиваем размер КХ кластера (должно быть просто, т.к. NoSQL)
3. |-- Получаем перекос в данных
4. |-- Партиции в Кафке % СН сервера != 0
5. |-- Увеличиваем кол-во партиций в Кафка топике
6. |-- Логика джобы зависит от партиционирования в Кафке
7. |-- Переписываем джобу

Kafka Engine: Архитектурные «ловушки»

Партиции в Kafka топике <=> CH сервера

Kafka партиции % CH сервера = 0

- Факторы для выбора количества партиций в Kafka
  - № Kafka брокеров
  - Replication factor
  - Бизнес логика завязанная на key-partitioning (плохая практика)
  - № Kafka racks
  - **+ № CH сервера**
- ❌ Low Coupling

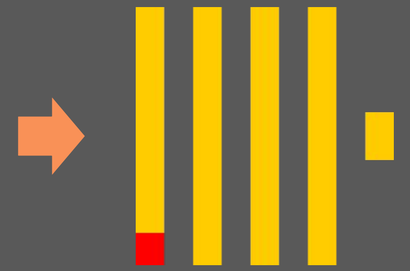
Другие проблемы

- DB и consumer-ы используют одни ресурсы
- Мониторинг
- Масштабирование

Kafka – ClickHouse интеграция: хорошее решение

Сторонний процесс

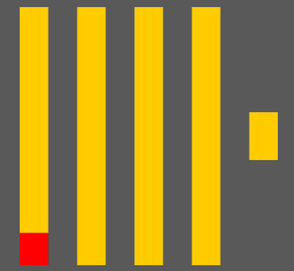
- Читает из Kafka – пишет в ClickHouse
- Stateless (почти)
- Знает про партиции / shard-ы =>
  - Обрабатывает изменение топологии
- Равномерное распределение по shard-ам
  - “Sharding key” конфигурация
- Горизонтальное масштабирование
- Exactly-Once Semantic (fault tolerant)
  - “Sorting key” конфигурация



Kafka – ClickHouse интеграция: реализация 1

Spark Structured Streaming

- Kafka: интеграция «из коробки»
- At-least-once семантика (трекает «оффсеты»)
- Batch «под капотом» - хорошо для ClickHouse
- Высокоуровневый API
- Хорошо известный инструмент



Kafka – ClickHouse интеграция: реализация 2

```
val streamDF = spark.readStream()
                    .format("kafka")
                    .option(<kafka brokers and topics>)
                    .load();


val writer = new SparkToClickHouseWriter(conf)

streamDF.forEachBatch(df -> writer.write(df))
```

Kafka – ClickHouse интеграция: реализация 3

```
def write(df: DataFrame) {
  val chServers = findChServerPerShard()
  val tasks = chServers
    .map((shardNum, chUrl) -> saveDatasetIntoShard(
                                  df,shardsNum,
                                  shardNum,chUrl)
        )
  runInParallel(tasks);
}
```

Kafka – ClickHouse интеграция: реализация 4

```
def saveDatasetIntoShard(df,shardsCnt,shardNum,chUrl) {
  val shardDf = df
      .filter(shardCondition)
      .repartition(parallelismLevel, shardingColumn)
      .sortWithinPartitions(deduplicationSortingKey)

  shardDf.write.jdbc(...)
}
```

Для работы
ClickHouse **insert_deduplicate** фичи

Kafka – ClickHouse интеграция: выводы

Плюсы

- Все что заказывали
  - Поддерживает изменение топологии
  - Равномерное распределение по shard-ам
  - Горизонтальное масштабирование
  - Exactly-Once семантика
- + Любая кастомная логика
- Учитывает особенности ClickHouse batching insert_deduplicate

Минусы

- Не поддерживает тип “Array”
- Дополнительные инструменты: Spark + HDFS
- Доп. ресурсы: CPU и RAM
  - Но это и плюс

Kafka – ClickHouse интеграция: выводы

Реализация

- ~200 строк кода на Java
- В проде 6 месяцев
- Открыт для расширения
  - например, дополнительная дедупликация
- `SparkToClickhouseWriter`: https://github.com/odnoklassniki/spark-to-clickhouse-sink

Выводы

ClickHouse – NoSQL?

- Нет «балансировщика» данных
  - Опять все «руками»
- Kafka Engine
  - В общем – работает
  - Но с оговорками

Итоги

- ClickHouse крут и «не тормозит» © !
- Порог входа высокий
- До NoSQL пока не дотягивает
  - Вставка в Distributed
  - Работа Kafka Engine
  - Перебалансировка данных (ее нет)
- Документация могла бы быть лучше
  - Следите за каждым словом!
- ClickHouse - open-source и мы сами можем сделать его лучше

Спасибо

От Data Platform с любовью

Всем «Технического превосходства»!

Ссылки

На меня

- ashutakGG
- artem.shutak@corp.mail.ru

К докладу

- `SparkToClickhouseWriter`: https://github.com/odnoklassniki/spark-to-clickhouse-sink