
Яндекс

Яндекс

Почему ClickHouse это модно?

Никита Михайлов, разработчик ClickHouse.

Обо мне

- 4 курс ФПМИ (ФИВТ) ПМИ

Обо мне

- 4 курс ФПМИ (ФИВТ) ПМИ
- Разрабатываю ClickHouse уже почти полгода

Обо мне

- 4 курс ФПМИ (ФИВТ) ПМИ
- Разрабатываю ClickHouse уже почти полгода
- Интересуюсь распределенными системами

Обо мне

- 4 курс ФПМИ (ФИВТ) ПМИ
- Разрабатываю ClickHouse уже почти полгода
- Интересуюсь распределенными системами
- Езжу на конференции

Обо мне

- 4 курс ФПМИ (ФИВТ) ПМИ
- Разрабатываю ClickHouse уже почти полгода
- Интересуюсь распределенными системами
- Езжу на конференции

Этим летом пошел стажироваться в команду ClickHouse и еще ни разу не пожалел об этом.

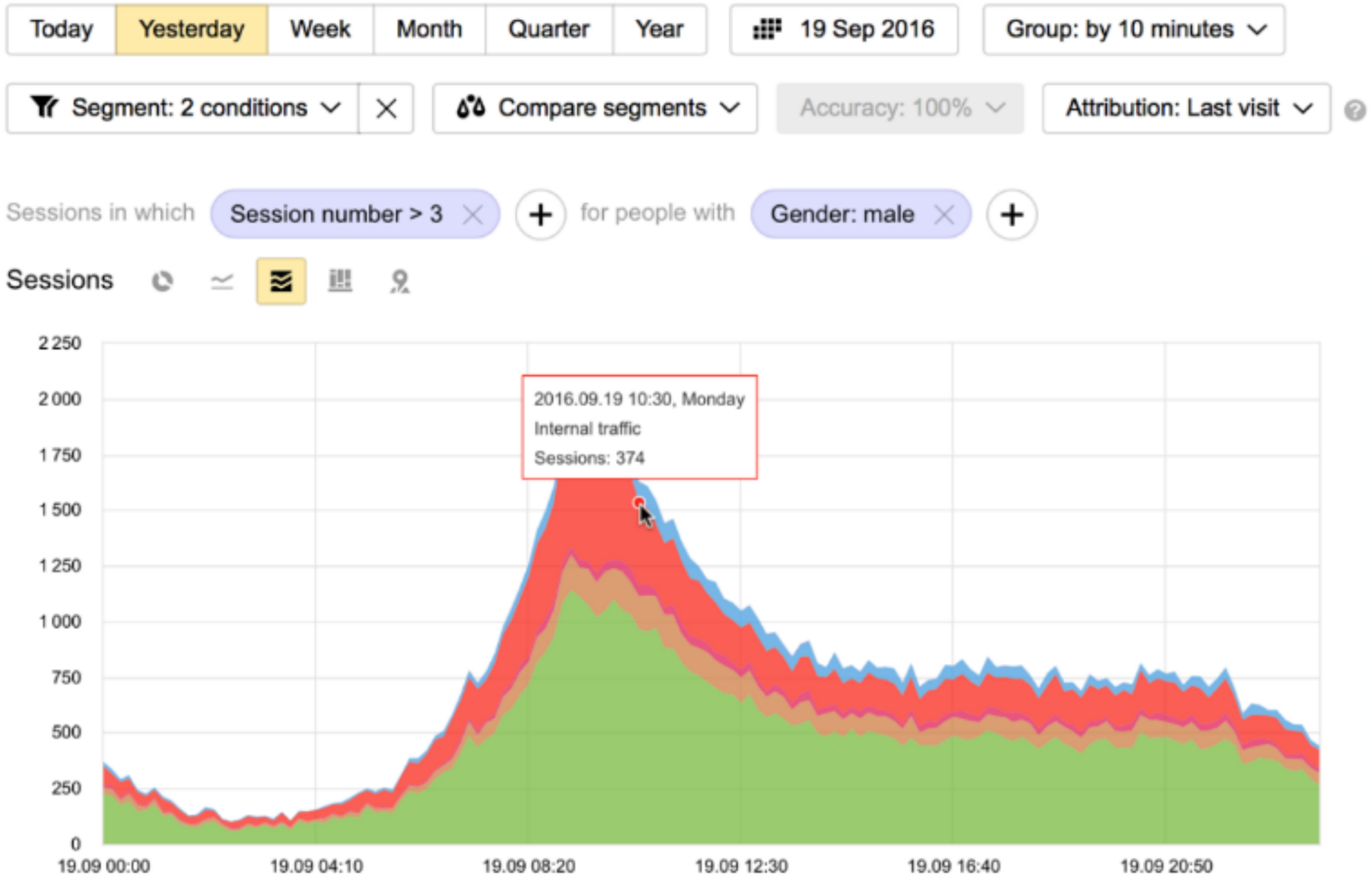
Что такое ClickHouse?

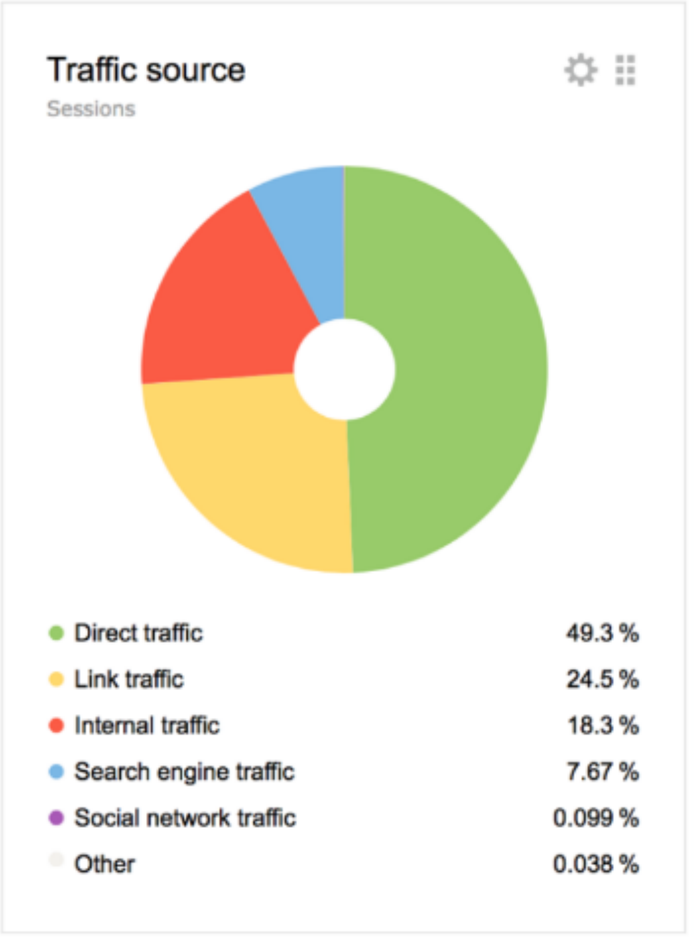
Яндекс.Метрика

Яндекс.Метрика - в тройке крупнейших веб-аналитических систем по количеству сайтов.

- Более 20 млрд. событий в день
- Более 1 млн. сайтов
- Более 100 000 аналитиков каждый день
- Более 1000 машин в кластере ClickHouse

Яндекс.Метрика



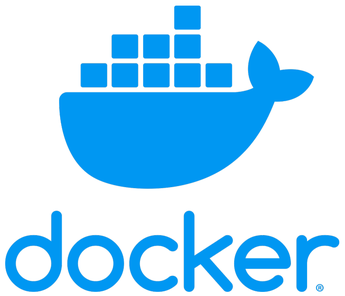


Почему ClickHouse обретает популярность?

Стек технологий

Используем самые «модные» технологии и библиотеки

- Основная кодовая база: C++17, boost, clang + sanitisers
- Распределенная синхронизация: ZooKeeper
- jdbc-driver: guava, mockito, lombok
- Тестирование: docker, vagrant, kazoo, iptables, minio



Про тестирование и наши технологии - отдельный доклад: https://youtu.be/2VPD6OZohT0

Алгоритмы и структуры данных.

Для каждой задачи используется уникальный алгоритм, а иногда и несколько

- **Очень** быстрые хэш-таблицы

  Их много, каждая - под свою задачу

- **Четыре** разных алгоритма для поиска подстроки в строке

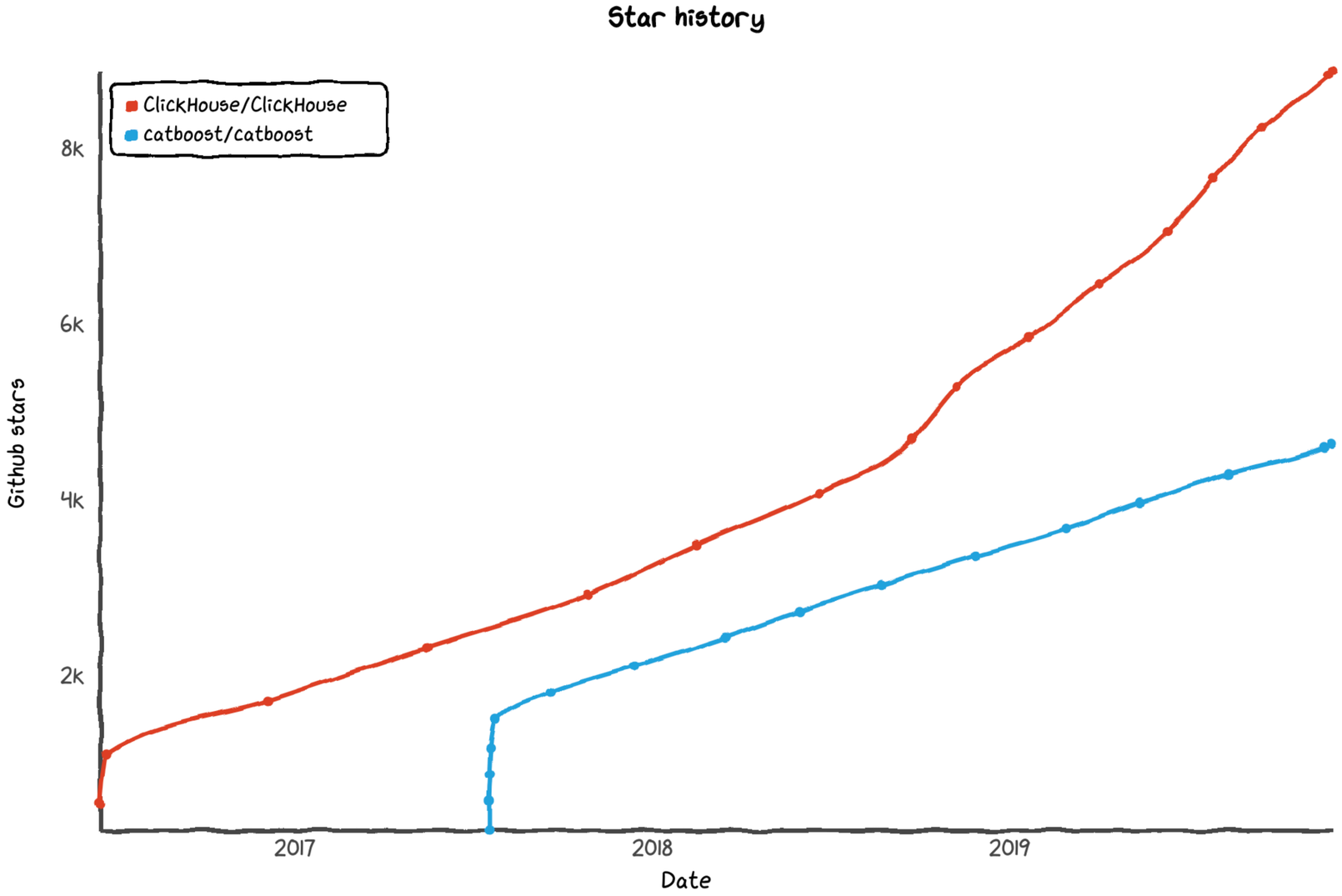
  https://habr.com/en/company/yandex/blog/466183/

- Thompson Sampling для оптимизации разжатая LZ4

  https://habr.com/en/company/yandex/blog/452778/

Наши усилия оправданы

На момент подготовки доклада у ClickHouse 8874 звездочки на github.com



https://star-history.t9t.io/#ClickHouse/ClickHouse&catboost/catboost

Поговорим про логи

Логи - это важно!

Предположим, что мы аналитики. Или разработчики, у которых все сломалось.

Как понять, что случилось?

- Читать логи глазами?
- «Грепать»?
- Писать скрипт на питоне?

Нет! Все вышеперечисленное - **очень медленно**. А ничто и никто тормозить не должны.

clickhouse-local

Лучшее из ClickHouse непосредственно с одним файлом.

```
$ clickhouse-local
--file ~/hits_v1.tsv
--structure 'WatchID UInt64, JavaEnable UInt8, ...'
--query 'SELECT UserID, count() FROM table GROUP BY UserID, SearchPhrase'
```

```
Read 8873898 rows, 7.88 GiB in 5.208 sec., 1704038 rows/sec., 1.51 GiB/sec.
UserID                  count()
8410854169855355129     3
```

github.com/ClickHouse/clickhouse-presentations/tree/master/2019_singapore_meetup

https://clickhouse.github.io/clickhouse-presentations/highload2019/

А если надо посмотреть логи ClickHouse?

system.text_log

Сколько ошибок было на сервере сегодня?

```
SELECT count(*)
FROM system.text_log
WHERE (level = 'Error') AND (event_date = today())
```

Ни одной! Потому что это ClickHouse.

system.metric_log

Self-monitoring.

С настраиваемым интервалом собирает все внутренние метрики ClickHouse, буферизирует и складывает в системную таблицу.

Удобно строить графики состояний сервера.

Логи в системных таблицах

Мы очень любим логи.

- system.part_log
- system.query_log
- system.query_thread_log
- system.trace_log
- system.text_log
- system.metric_log

Описание других системных таблиц доступно по ссылке ниже.

https://clickhouse.yandex/docs/ru/operations/system_tables/

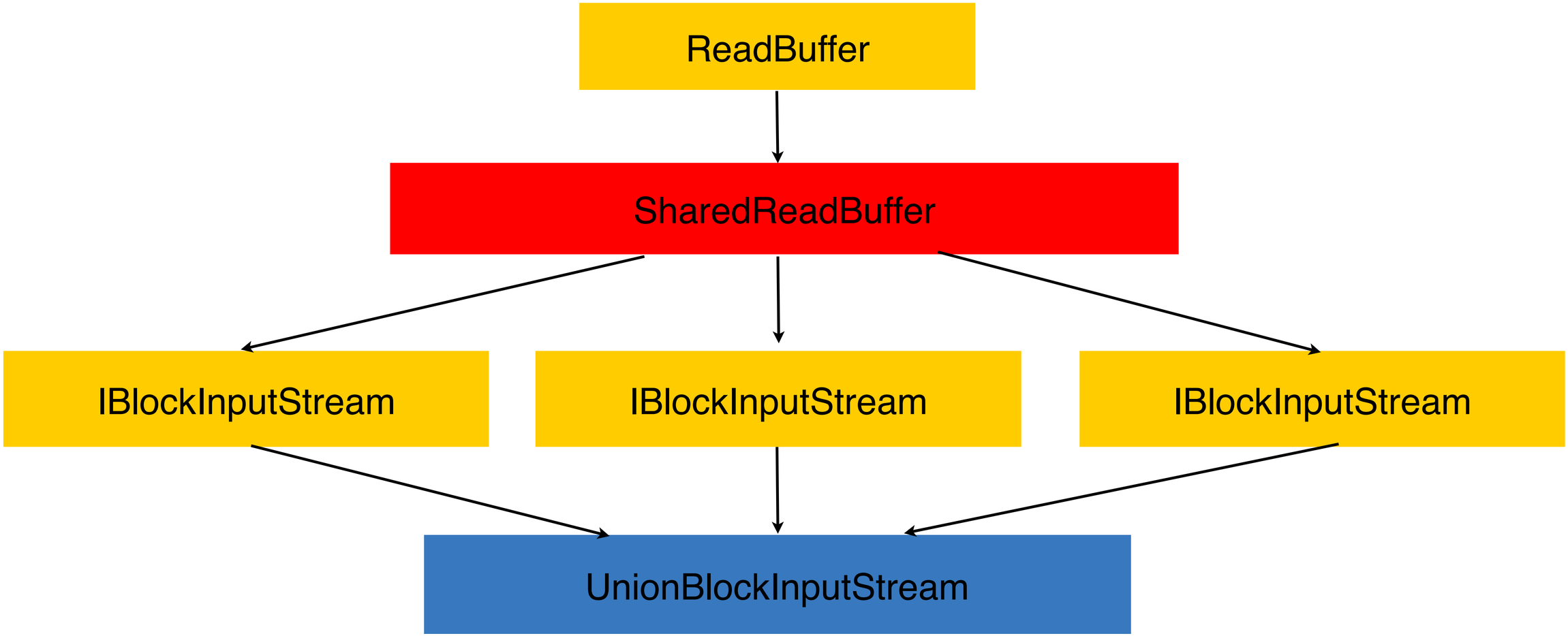
Что еще полезного?

Параллельный парсинг форматов данных (но не всех)

Введение:

- ReadBuffer - абстрактный класс для буфферизованного чтения
- IRowInputStream - интерфейс потока для чтения по строкам
- IBlockInputStream - интерфейс потока для чтения по блокам

Что было раньше?



std::mutex

Примитив синхронизации, который обеспечивает эксклюзивный доступ к разделяемым данным в многопоточной среде

Класс, реализующий mutex, имеет методы .lock() и .unlock()

Правильное использование mutex - с помощью std::lock_guard или std::unique_lock.

Что с этим не так?

Код выглядит примерно так:

```
bool nextImpl() override
{
  std::lock_guard<std::mutex> lock(mutex);
  return getFileSegment();
}
```

- Вся синхронизация - один единственный mutex.
- Большой контеншн.
- Потерян порядок вставки блоков в таблицу.

Как сделать хорошо?

Подумать.

Выделим три роли:

"Сегментатор"

"Парсер"

"Читатель"

Для чего?

> "Читатель" - реализует функцию nextImpl().

> "Парсер" - превращает сегмент памяти с данными в Block.

> "Сегментатор" - идет по файлу с огромной скоростью, разбивая его на куски.

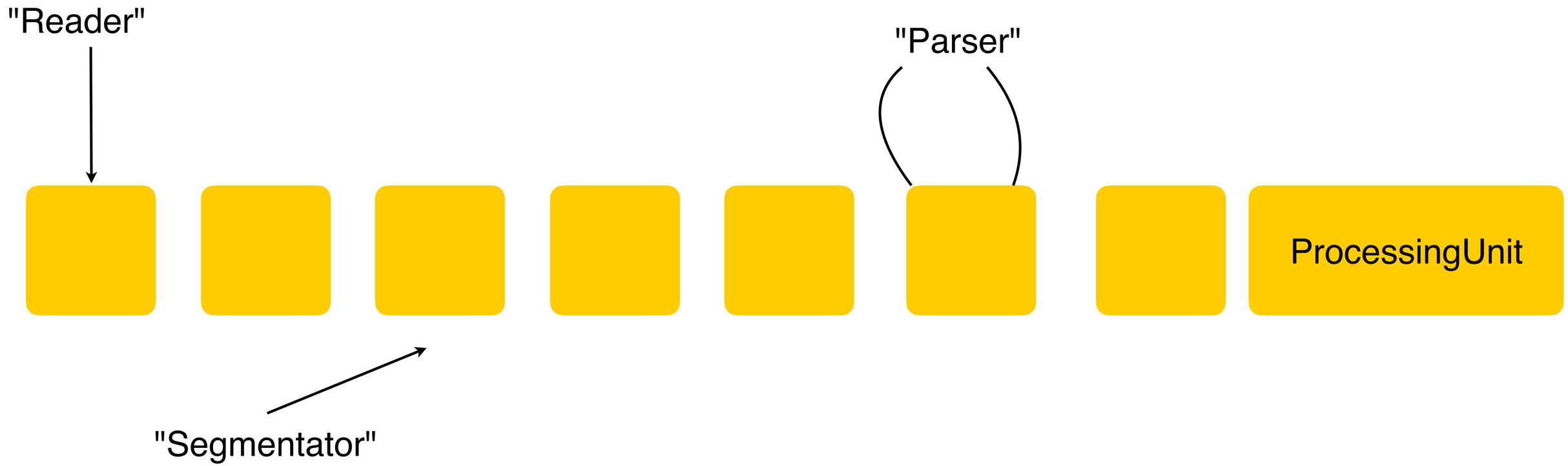
Вспомогательные структуры

```
enum ProcessingUnitStatus
{
  READY_TO_INSERT,
  READY_TO_PARSE,
  READY_TO_READ
}
```

```
struct ProcessingUnit
{
  Block block;
  Memory<> segment;
  ProcessingUnitStatus status;
  bool is_last;
}
```

Все ProcessingUnit сложим в контейнер, назовем его WorkingField.

Схема взаимодействия.



Потоки «ходят» WorkingField в одну сторону для сохранения порядка, но с разной скоростью.
Нужна синхронизация!

std::condition_variable

Примитив синхронизации, который используется для блокирования множества потоков до наступления одного из событий.

- будет получено извещение из другого потока
- тайм-аут
- произойдет ложное пробуждение

У класса, реализующего condition_variable есть методы .wait, .notify_all, .notify_one

Правильное использование метода wait:

- condvar.wait(lock, [](){ return predicate() })

Как выглядят потоки?

```
void ParallelParsingBlockInputStream::segmentatorThreadFunction()
{
  while (!finished)
  {
    auto unit = getNextUnit();
    {
      std::unique_lock lock(mutex);
      segmentator_condvar.wait(lock,
          [&]{ return unit.status == READY_TO_INSERT || finished;}
    }

    getFileSegment();

    unit.status = READY_TO_PARSE;
    sheduleParserThreadForCurrentUnit();
  }
```

ThreadPool

Очень полезный паттерн, который обеспечивает фиксированным количеством потоков, готовых выполнять полезную работу асинхронно.

- Количество потоков задается в конструкторе
- Потоки заранее инициализируются
- Позволяет сосредоточиться на задаче, а не на синхронизации потоков

Какие были сложности?

Сложности с C++.

- Отсутствие move и copy конструктора у ProcessingUnit. Какой контейнер выбрать для WorkingField?
- Битовое сжатие std::vector<bool>. Неупорядоченный доступ к ячейке памяти из разных потоков - data race.

Сложности в схеме взаимодействия:

- Как выбрасывать исключение при парсинге из другого потока?
- Как завершить работу класса в случае исключения?
- Какого размера сделать WorkingField?

Баги

Еще немного про баги

Performance тесты не завершаются в CI. Почему? Локально не воспроизводится.

```
$ docker ps -a // Узнаем hash контейнера
$ docker exec -u root -it <hash> bash
$ ps aux | grep clickhouse // Узнаем PID процесса clickhouse-server
$ sudo -u clickhouse gdb -p <PID>
$ thread apply all backtrace
```

Оказалось, что один поток заблокировался на condition_variable на бесконечное время. Другой поток бесконечно ждет первый.

Самое главное - зачем?

SELECT * FROM table_TabSeparated	x3.50 (0.389 s.)	(0.111 s.)	OK
SELECT * FROM table_TabSeparatedWithName	x1.01 (0.409 s.)	(0.406 s.)	OK
SELECT * FROM table_TabSeparatedWithName	(0.393 s.)	x1.04 (0.409 s.)	OK
SELECT * FROM table_CSV	x4.44 (0.590 s.)	(0.133 s.)	OK
SELECT * FROM table_CSVWithNames	x1.01 (0.587 s.)	(0.580 s.)	OK
SELECT * FROM table_Values	(0.370 s.)	x1.03 (0.381 s.)	OK
SELECT * FROM table_JSONEachRow	x1.86 (0.776 s.)	(0.418 s.)	OK
SELECT * FROM table_TSKV	x4.37 (0.625 s.)	(0.143 s.)	OK

Теперь ClickHouse не тормозит еще больше.

Спасибо!

Никита Михайлов

Разработчик ClickHouse

 jakalletti@yandex-team.ru