YATALKS

Искусство бенчмаркинга распределённых баз данных на примере YDB

Евгений Иванов
Старший
разработчик YDB

YfD */

YDB

Категории разработчиков

* Разработчики СУБД
* Пользователи СУБД (разработчики приложений, использующих БД)
* Разработчики без СУБД (наверняка мечтают о БД)

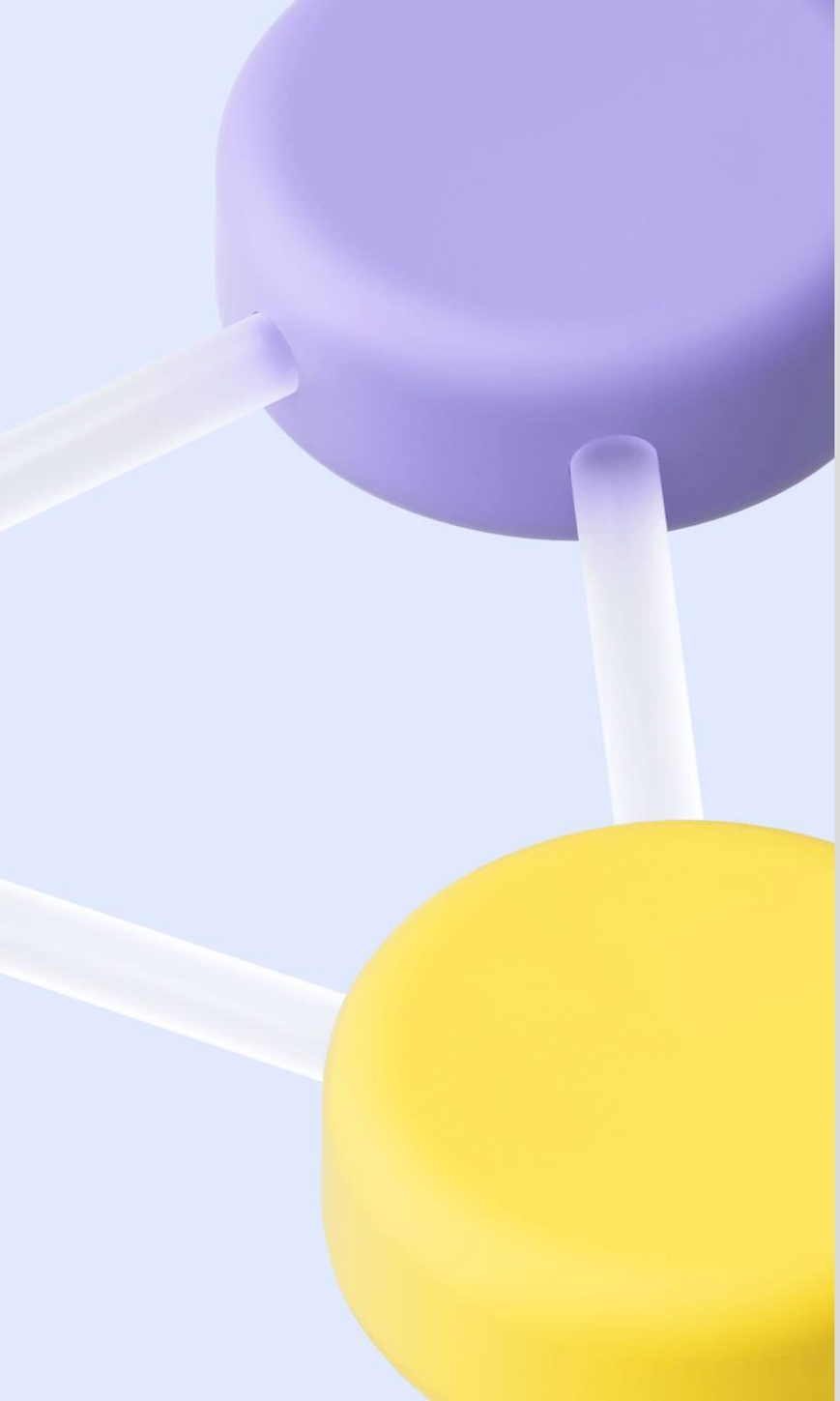
Что же у них общего, кроме багов?

Бенчмарки!

Содержание

01

Обзор YDB



YDB

YDB — Open-Source Distributed SQL Database:

* Реляционная СУБД
* Работает на кластере
* Строгая консистентность

* Лицензия Apache 2.0
* github.com/ydb-platform/ydb

Строгая консистентность

* CAP-теорема — выбираем CP

Строгая консистентность

* CAP-теорема — выбираем CP
* Serializable уровень изоляции транзакций

Высокая доступность и отказоустойчивость

* YDB работает в нескольких зонах доступности (дата-центрах)
* Переживает отказ одного ДЦ и стойки в другом ДЦ:
    - без участия человека
    - сохраняет доступность на чтение-запись

Mission critical database

* Подходит для проектов, требующих доступность 365×24×7
* Не требует окна обслуживания

Не только OLTP

* OLAP в активной разработке
* YDB – платформа:
    - YDB Topic Service (персистентная очередь)
    - Сетевые диски в Yandex Cloud (NBS)
    - Хранение временных рядов

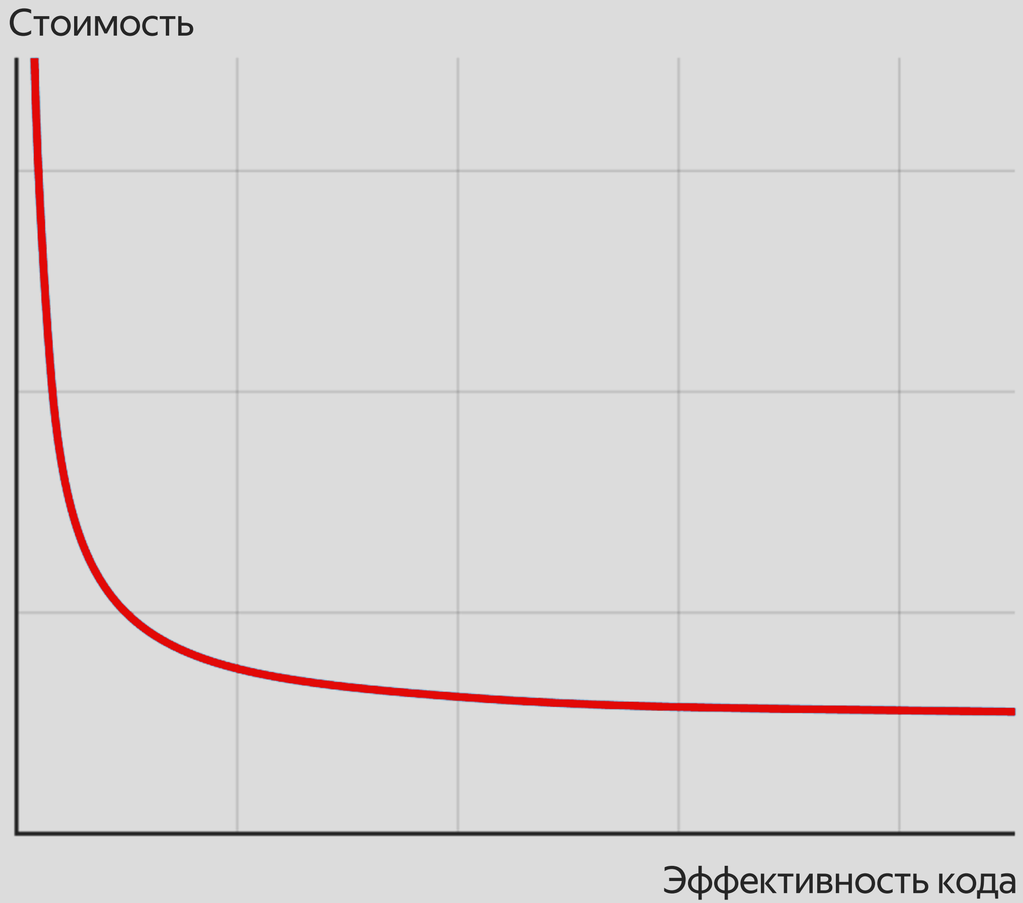
02

Эволюция бенчмарков YDB

Performance-цели

* **throughput:**
  обслуживать бесконечно большое число запросов
* **latency:**
  запрос ещё не отправили, а уже получили ответ

Стоимость эксплуатации СУБД



До выхода в опенсорс

* Основной фокус на фичах и масштабируемости
* Самодельные бенчмарки, преимущественно против регрессий
* Проверка улучшений на testing- и prestable-кластерах

YDB prestable

250

серверов

> 1000

баз данных

500 ТВ

хранимых данных

После выхода в опенсорс

* Повышение эффективности
* Сравнение с другими СУБД

Выбрали:

* Yahoo! Cloud Serving Benchmark (YCSB) — key-value нагрузка
* TPC-C — золотой стандарт для распределенных транзакций

Оборудование для бенчмарков

YDB – бенчмарк бенчмарков

Ожидание: берем бенчмарки и улучшаем YDB

Реальность: берем YDB и улучшаем бенчмарки

03

YCSB

Yahoo! Cloud Serving Benchmark

Yahoo! Cloud Serving Benchmark

* Популярный key-value бенчмарк
* Изначально для NoSQL баз данных, но используется всеми
* Поддерживает почти все современные СУБД

Почему начали с key-value?

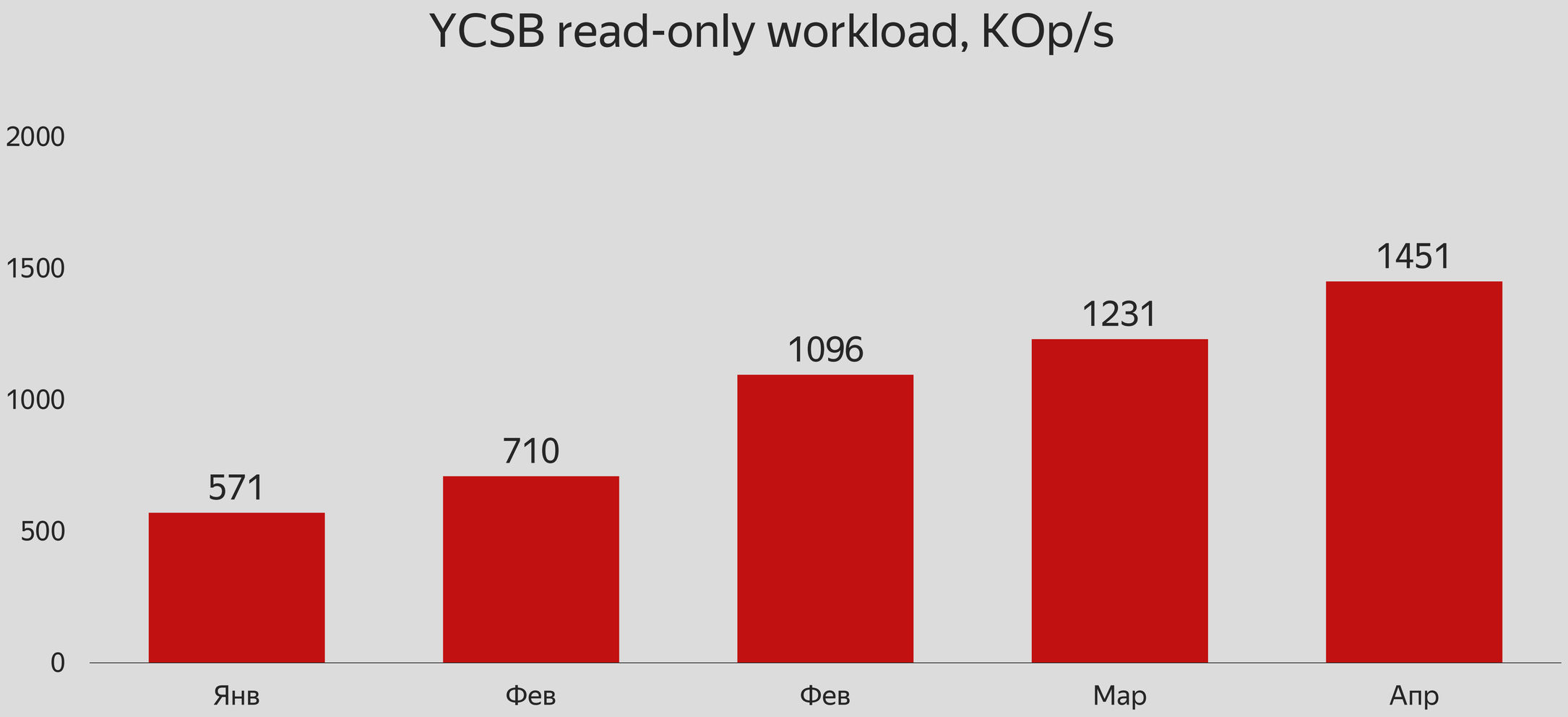
* Key-value нагрузка по-прежнему актуальна
* Она проста для анализа
* Нельзя сделать быстрые распределенные транзакции с плохой производительностью key-value запросов

YCSB workloads

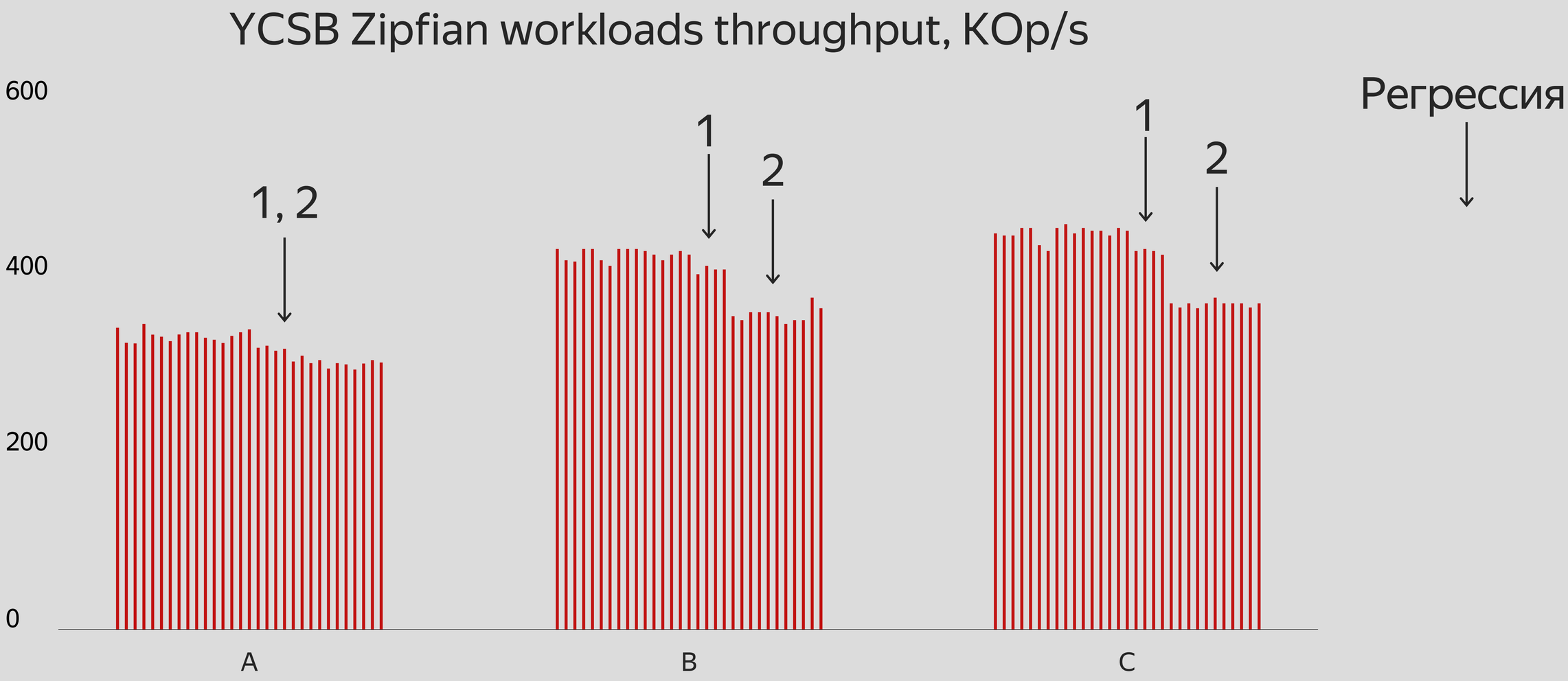
- A **много обновлений:** 50% чтений, 50% обновлений
- B **преимущественно чтения:** 95% чтений, 5% обновлений
- C **только чтения**
- D **чтение свежих записей:** 95% чтений, 5% вставок
- F **чтение-изменение-запись:** 50% чтений, 50% чтение-обновление
- E **короткие диапазоны:** 95% сканов, 5% вставок

Везде, кроме D, по умолчанию zipfian-распределение ключей.
Но мы тестируем и с uniform.

Поиск и устранение узких мест

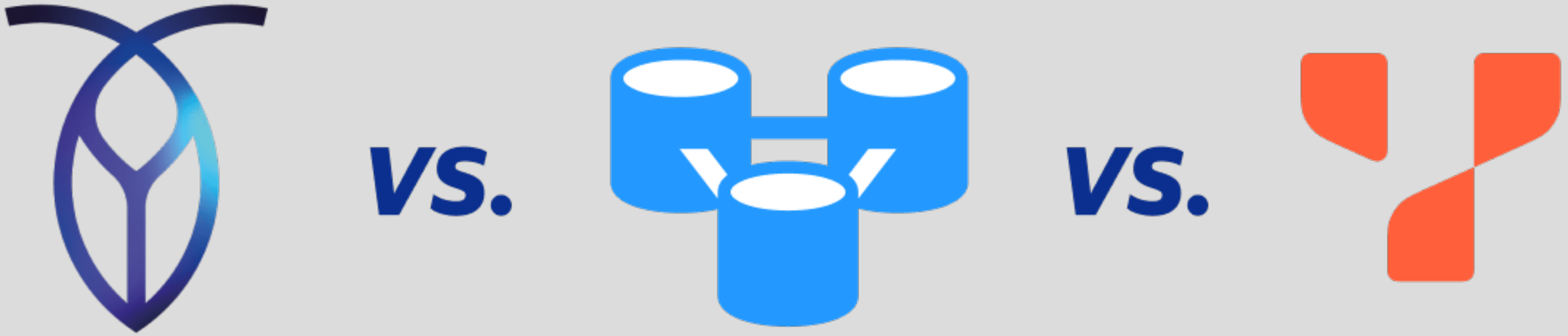


Регрессионное тестирование



Изменение throughput с течением времени

Сравнение с другими СУБД



Критерии выбора оппонентов

* Распределенные СУБД
* Реляционные
* Опенсорс
* Зрелые: есть продакшен и сравнение производительности в различных открытых источниках для валидации наших результатов

Если Вы в процессе выбора СУБД, то скорее всего в Вашем списке именно эти СУБД

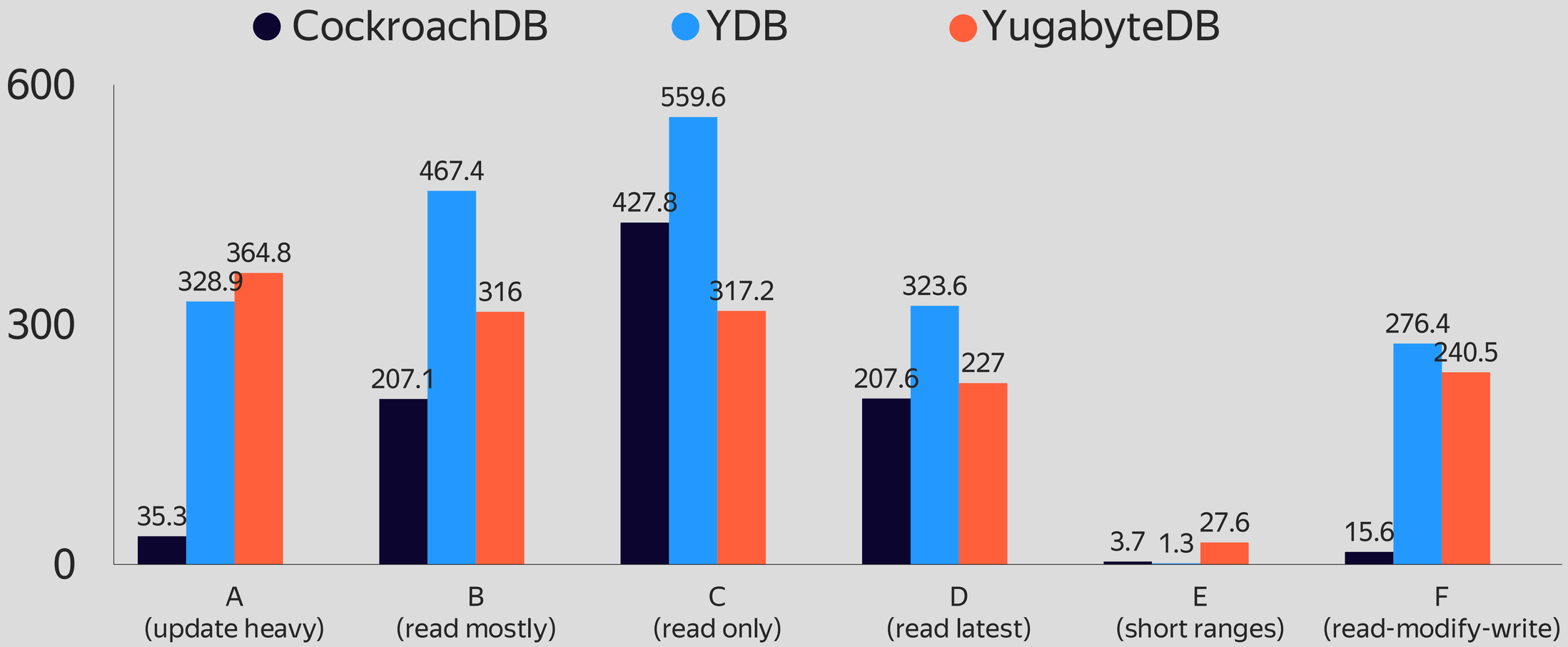
Тестовый сетап

3 сервера, каждый со следующей конфигурацией:

* 128 логических ядер (2x Intel Xeon Gold 6338 2,00 ГГц)
* 4x NVMe Intel
* 512 ГБ ОЗУ
* Сеть 50 Гбит/с
* Включены transparent huge pages

Результаты YCSB (300M строк)

Throughput, KOp/s (больше лучше)



Нюансы реализации СУБД

Нужно понимать, как работает СУБД:

* Проблемы с вертикальным и с горизонтальным масштабированием YugabyteDB
* Проблемы в YDB на workload E (уже исправили)

Нюансы SQL

Недостаточно просто запустить бенчмарк, надо понимать, как он реализован: INSERT vs. UPDATE vs. UPSERT

Нюансы реализации бенчмарка

* Сам бенчмарк – высоконагруженное приложение
* Оригинальная реализация YCSB на Java
* pingcap/go-ycsb – реализация на Go
* ЯП и среды выполнения сильно различаются по эффективности
* Получили разные результаты из-за особенностей SDK и ЯП

Функции SDK

* Реализует логику discovery и балансировку
* Многопоточная реализация транспорта
* Пул сессий
* Поддержка как синхронного, так и асинхронного API
* Ретраи запросов

Нагружатель

* Один или несколько процессов / инстансов бенчмарка
* Обсуждаем потребление CPU / RAM именно нагружателя

Проблемы в нагружателе YCSB

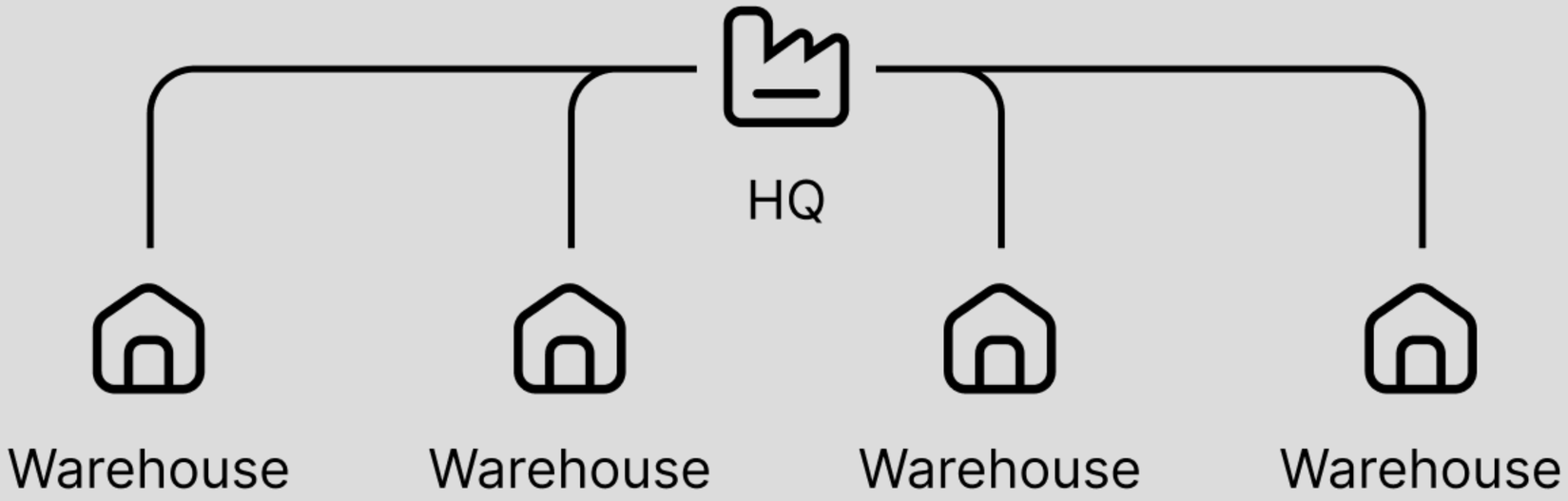
* Нагружатель YCSB потребляет много RAM и CPU
* Нагружатель YCSB использует большое число физических потоков
* Долгая загрузка начальных данных из-за особенностей реализации нагружателя

Нагружатель TPC-C страдает от тех же проблем

04

TPC-C

Эмуляция e-commerce организации



TPC-C

Стандарт появился в 1992 году

«Единственная объективная методика оценки производительности OLTP», — CockroachDB

«Наиболее реалистичное и объективное измерение производительности OLTP-систем», — YugabyteDB

Логика TPC-C

* Число складов задаётся при запуске
* Каждый склад обслуживает 10 районов, примерно 100 МБ данных
* В каждом районе есть терминал
* Пользователи делают заказы и оплачивают их
* Иногда проверяется статус заказа
* Выполняется доставка
* На складах проводится инвентаризация

Транзакции TPC-C

44%
Заказ

4%
Статус заказа

4%
Доставка

4%
Инвентаризация

44%
Платеж

Транзакции TPC-C

* Требуется уровень serializable
* Многошаговые
* Примерно 2:1 — соотношение чтений и записи
* Измеряем только число заказов в минуту — tpmC

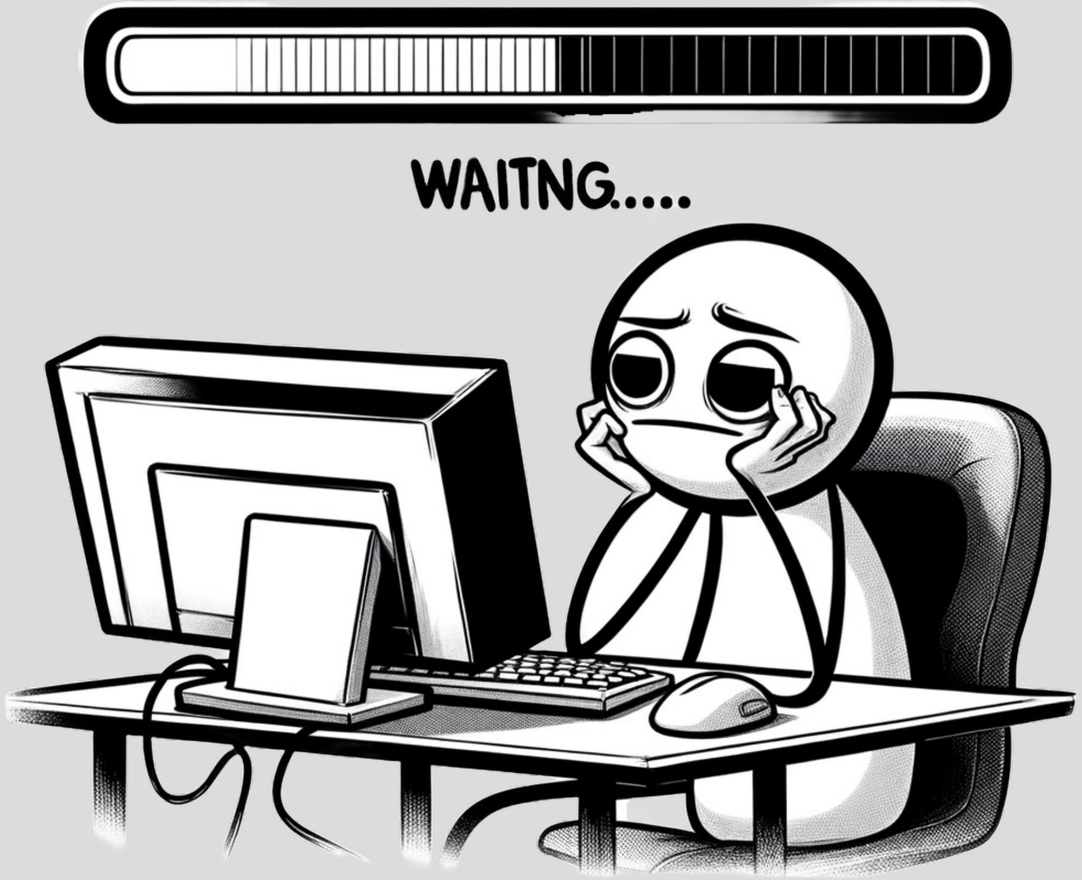
Проект CMU Benchbase

* Multi-DBMS SQL Benchmarking Framework via JDBC
* Написан в Carnegie Mellon под руководством проф. Энди Павло
* Легко добавлять новые СУБД и бенчмарки
* Единственная широко известная реализация TPC-C
* YugabyteDB использует форк Benchbase
* Мы пошли тем же путём

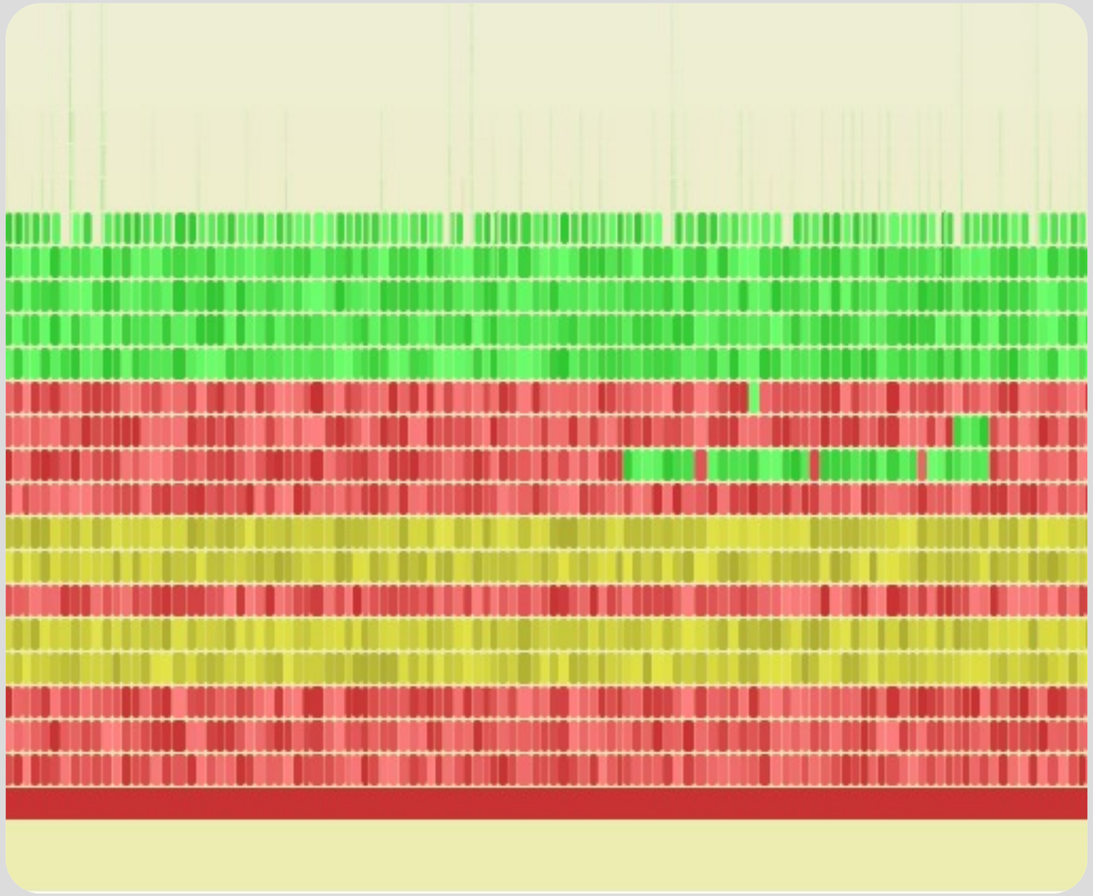
Проблема 1: импорт данных через INSERT

* Базы предлагают более быстрые операции наподобие bulk upsert в YDB

* Размер начальных данных – терабайты



Нагружатель потребляет весь CPU и поэтому медленно наполняет БД



Lcom/oltpbenchmark/benchmarks/tpcc/TPCCUtil:::randomStr
Lcom/oltpbenchmark/benchmarks/tpcc/TPCCLoader:::loadStock
Lcom/oltpbenchmark/benchmarks/tpcc/TPCCLoader$2:::load
Lcom/oltpbenchmark/api/LoaderThread:::run
Lcom/oltpbenchmark/util/ThreadUtil$LatchRunnable:::run
Interpreter
Interpreter
Interpreter
call_stub
JavaCalls::call_helper
JavaCalls::call_virtual
thread_entry
JavaThread::thread_main_inner
Thread::call_run
thread_native_entry
start_thread
Thread-191
all

У нагружателя потоков много, но лок один

* Потоки генерируют случайные строки
* И делят между собой `java.util.Random`
* Правильнее использовать `ThreadLocalRandom`

Проблема 2: в нагружателе модель — терминал-поток

* И YCSB, и TPC-C используют синхронные запросы к БД
* В TPC-C терминал эмулируется потоком
* 15 000 складов (1,5 TB данных) — 150 тысяч терминалов

Синхронно или асинхронно?

* Хотим concurrency, не создавая большое число потоков
* В старых ЯП писать асинхронные программы сложнее
* Модель Future/Promise
* Goroutines — просто и эффективно
* Java virtual threads — попытка догнать Go

Проблема 3: хранение статистики в нагружателе TPC-C

40 МБ
на один склад

15 000
складов

600 ГБ
RAM

Требования к запуску 15 000 складов

Итого:

* 150 000 потоков
* 600 GB RAM
* Мы были вынуждены запускать это на 5 серверах (каждый 128 ядер и 512 GB RAM)

Масштабируемся

* Хотим прогрузить СУБД, в которой 9, 15, 30, 60, 81 серверов
* YDB, CockroachDB, YugabyteDB
* В AWS один такой эксперимент стоит $10,000
* И одним экспериментом не обойтись

Минимум изменений — максимум пользы

* Java virtual threads (Java >= 21)
* 1 терминал — 1 виртуальный поток
* Храним гистограмму с временем выполнения транзакций

* 6 МБ на склад вместо 40
* 1 ядро CPU на 1000 складов
* 15 000 складов — 90 GB, 15 потоков

Максимум пользы и дедлок

* Java virtual threads — серебряная пуля для русской рулетки
* Оказалось, что очень легко получить дедлок

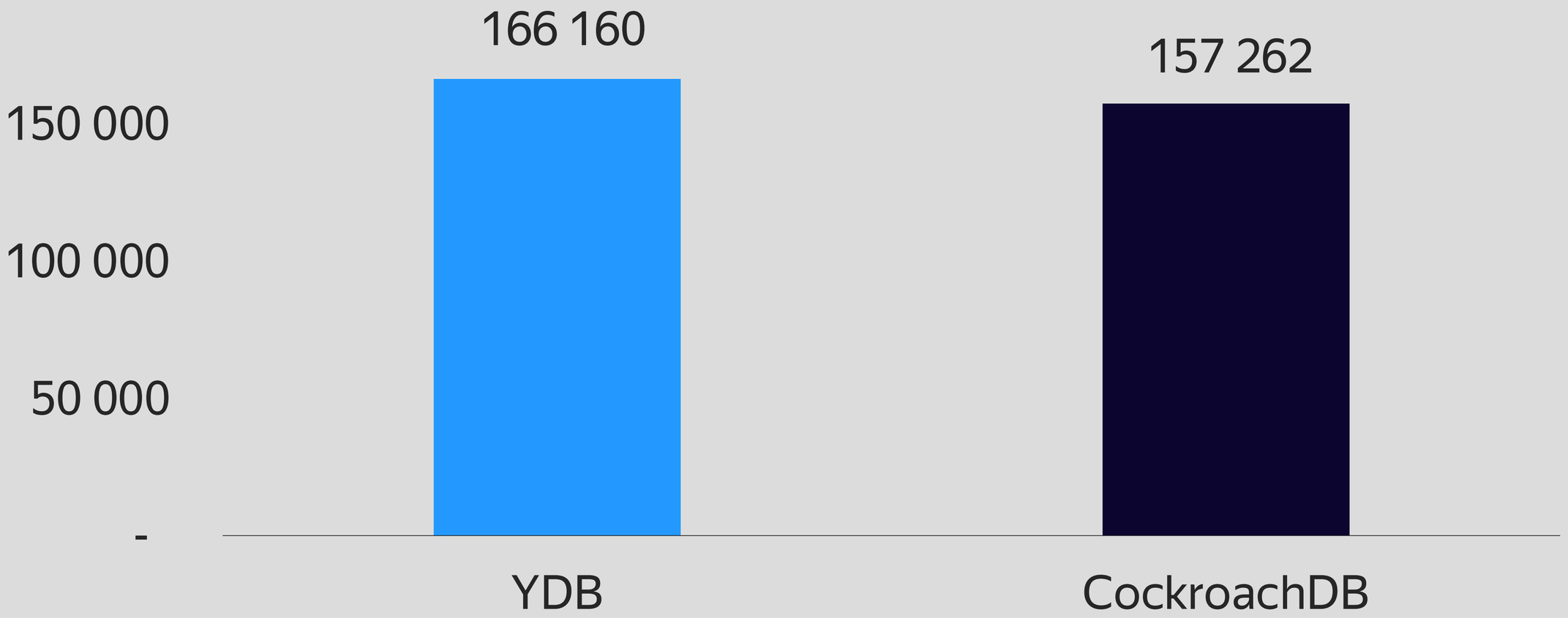
* Число сессий ограничено
* Одни vthreads, удерживая сессию, ожидают I/O и теряют carrier thread
* Другие vthreads в ожидании сессии делают Object.wait() и блокируют carrier thread

Наш форк и апстрим

* github.com/ydb-platform/tpcc
* Планируем постепенно затянуть наши улучшения в апстрим
* Но при отсутствии Java >= 21 в апстриме, будем поддерживать свой форк

Результаты TPC-C *

Max tpmC (больше лучше)



** Результаты не являются официально принятыми TPC результатами и несопоставимы с другими результатами теста TPC-C, опубликованными на сайте TPC*

05

В мире ARM

Смена архитектуры и производительность

Почему ARM?

* Apple silicon - попробуйте YDB на ноутбуке
* Всё больше облаков предлагают сервера на ARM
* Низкое энергопотребление, стоимость и масштабируемость серверов на базе ARM

Казалось бы

* YDB написан на C++
* Чаще упираемся в высокоуровневую логику, память и диск
* Компиляторы обычно творят волшебство
* Достаточно ли запускать бенчмарки на x86-64?

Особенности AARCH64

* В некоторых местах performance сильно зависит от SIMD (Single Instruction/Multiple Data)
* Arm Neon не поддерживает SSE4.2, AVX2, AVX512
* Потенциально разная стоимость атомарных операций, вызовов виртуальных функций и доступа к невыровненной памяти
* Наличие кода, оптимизированного для x86-64

Найденные проблемы

* Низкоуровневые библиотеки не всегда оптимизированы под ARM: сжатие, хеш-функции и хеш-таблицы
* Нашли проблему в библиотеке CRC — исправили, и простой `SELECT COUNT (*)` стал в среднем на 15% быстрее

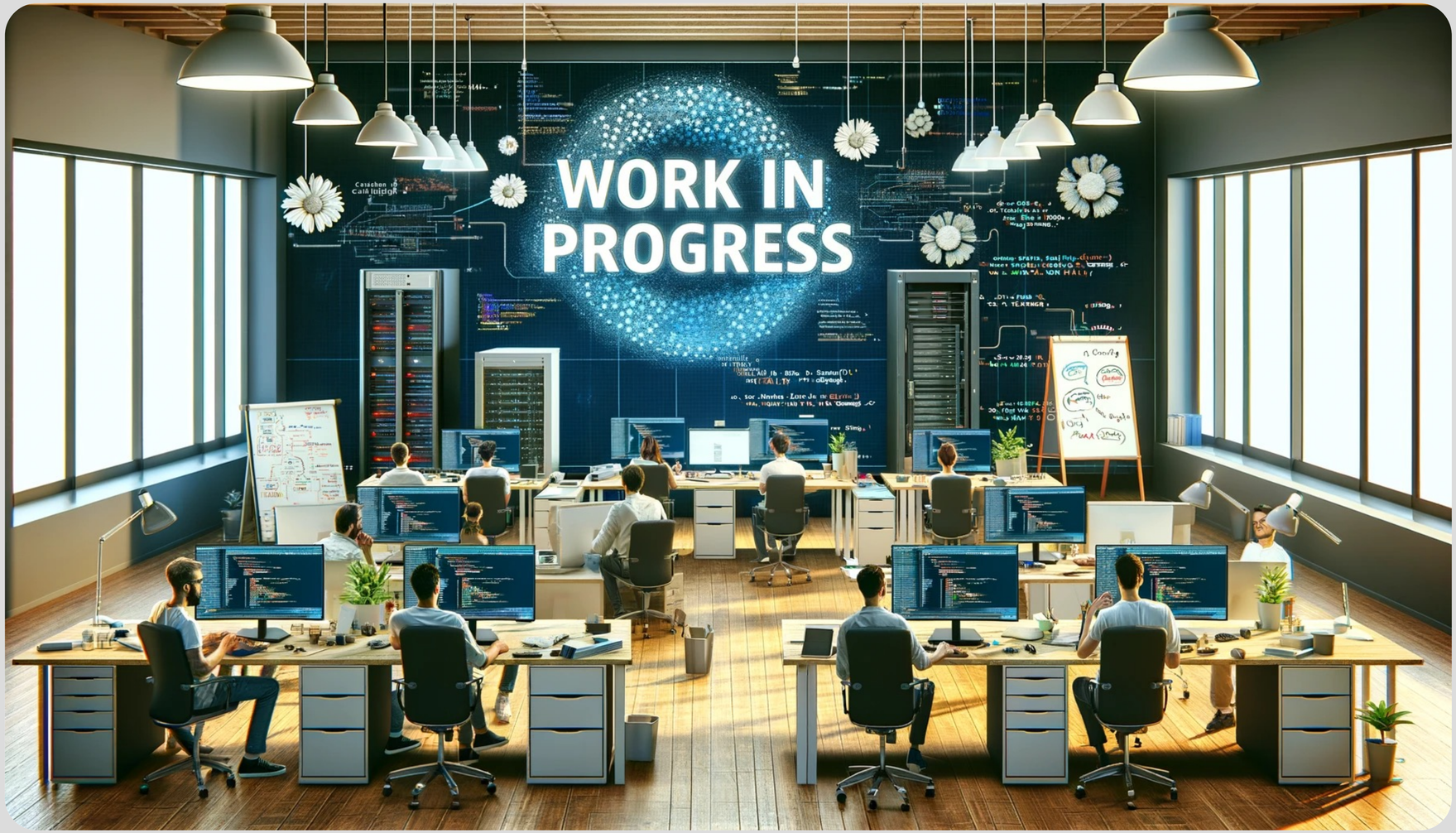
SQL и JIT

* СУБД активно используют JIT для выполнения SQL
* Для ARM требуется свой JIT

Виды сравнения

* Часто сервера на ARM дешевле x86-64
* Первый вид сравнения: одинаковые ресурсы CPU/RAM
* Второй вид сравнения: одинаковые по стоимости системы (хорошо согласуется с идеологией TPC-C)

Результаты ARM: следите за blog.ydb.tech



06

Заключение

Заключение

* Вы либо уже бенчмаркаете базы, либо скорее всего займетесь этим
* Будьте готовы улучшать опенсорс бенчмарки или писать свои
* Не стоит писать бенчмарк на Go, если приложение на Java
* Бенчмаркайте на той архитектуре, которую используете
* Старайтесь писать бенчмарки так, чтобы они потребляли существенно меньше ресурсов, чем тестируемые СУБД
* При выборе СУБД смело используйте YCSB и TPC-C

Спасибо

Евгений Иванов

Старший разработчик YDB

i@eivanov.com

@eivanov89

YfD */

YDB.tech: блог, сообщество,
видео других выступлений

YDB