



Stroppy

a database stress testing tool

Переосмысление тестирования производительности СУБД

Константин Осипов, СТО, picodata.io **Илья Хапров**, strop.py

strop.py, cloud.strop.py, github.com/strop.py-io



О чём этот доклад

01

Задача

Как выбирают СУБД и почему чужим числам не верят

02

Обзор инструментов

Что не так с `pgbench`, `sysbench`, `HammerDB`, `YCSB` и `go-tpc`

03

Архитектура Stroppy

Четыре компонента: CLI, набор тестов, `Stroppy Cloud` и `Graphene CI`

04

Доверие и прозрачность

Эффективность инструмента, открытая архитектура и открытый код

01 / ЗАДАЧА

Популярная задача: выбрать оптимальную СУБД

...и самая частая ошибка: выбрать её по чужому бенчмарку...

Критерии выбора СУБД



Производительность

Пропускная способность и задержки под длительной нагрузкой на данных рабочего объёма



Надёжность

Отказы узлов и сети, обновление без простоя и корректность данных после сбоя



Удобство

Развёртывание, миграции, инструменты и порог входа для команды



Стоимость владения

Оборудование, лицензии, люди, обновления и сопровождение

Измерить проще всего производительность. Но и её каждый измеряет по-своему.

Как проверить?



Отчёты вендоров

Своя методика, своё оборудование и выигрышный сценарий. Два таких отчёта несопоставимы.



Открытые тесты

Другие версии, другие нагрузки и другие машины. Часто устарели на год или два.



Собственные измерения

Честнее всего, но дорого: стенд, данные, настройка каждой СУБД и недели работы.

В итоге выбирают по чужому отчёту и собственной интуиции.

Почему чужим числам не верят

01

Критерии субъективны

Что важнее: p99 или средняя, чтение или запись, пустая СУБД или терабайт данных

02

Нет единой шкалы

tps, qps, итерации и tpmC в каждом инструменте определены по-своему

03

Клауза ДеВитта

Лицензии ряда коммерческих СУБД запрещают публиковать результаты без согласия вендора

04

Нет доверия к тестам

Опубликованный бенчмарк почти всегда сделан заинтересованной стороной

Пока нет общей шкалы, чужие числа невозможно проверить.

Цель: одна точка сравнения в четырёх измерениях

01

Тесты

Известный тест вызывает доверие. TPC-S знают все, но он масштабируется по складам и почти не связывает данные. Banking test связывает любые счета.

Правила TPC: неаудированный прогон называется «derived from TPC-S», метрика tpms и сравнение с официальными результатами запрещены.

02

СУБД и тюнинг

У каждой СУБД свой диалект SQL, свои лучшие практики и свой тюнинг. Доверие нужно и к переносу теста на диалект, и к настройке СУБД.

Клауза ДеВитта: лицензии ряда коммерческих СУБД запрещают публиковать результаты без разрешения вендора.

03

Оборудование

Размещение, виртуализация, вертикальное или горизонтальное масштабирование, shared nothing или shared storage. 20 физических ядер и 20 виртуальных в трёх ЦОД это разные профили.

За 100 долларов: три ЦОД и 100 бизнес-операций в секунду на известном тесте.

04

Облако и цена

Один и тот же профиль стоит по-разному у каждого провайдера и on-premise. Как только считаются деньги, облако становится отдельным измерением.

Стенд поднимается по шаблону у любого провайдера, где есть квота.

Инструмент даёт проверенные сообществом шаблоны по каждому измерению. Пользователь выбирает одну точку и сравнивает яблоки с яблоками.

Из чего складывается доверие



Воспроизводимость

Один seed, одни данные и один сценарий. Прогон на той же машине даёт то же число



Равное оборудование

Одинаковые ядра, память и схема репликации для всех участников сравнения



Открытость

Код инструмента, методика, конфигурации и сырые результаты доступны всем



Корректность

Инварианты проверяются, ошибки считаются.
Throughput без этого ничего не значит

Сначала инструмент должен быть проверяемым, и только потом быстрым.

Но у нас уже есть утилиты дома

Инструментов много, объективного сравнения
нет

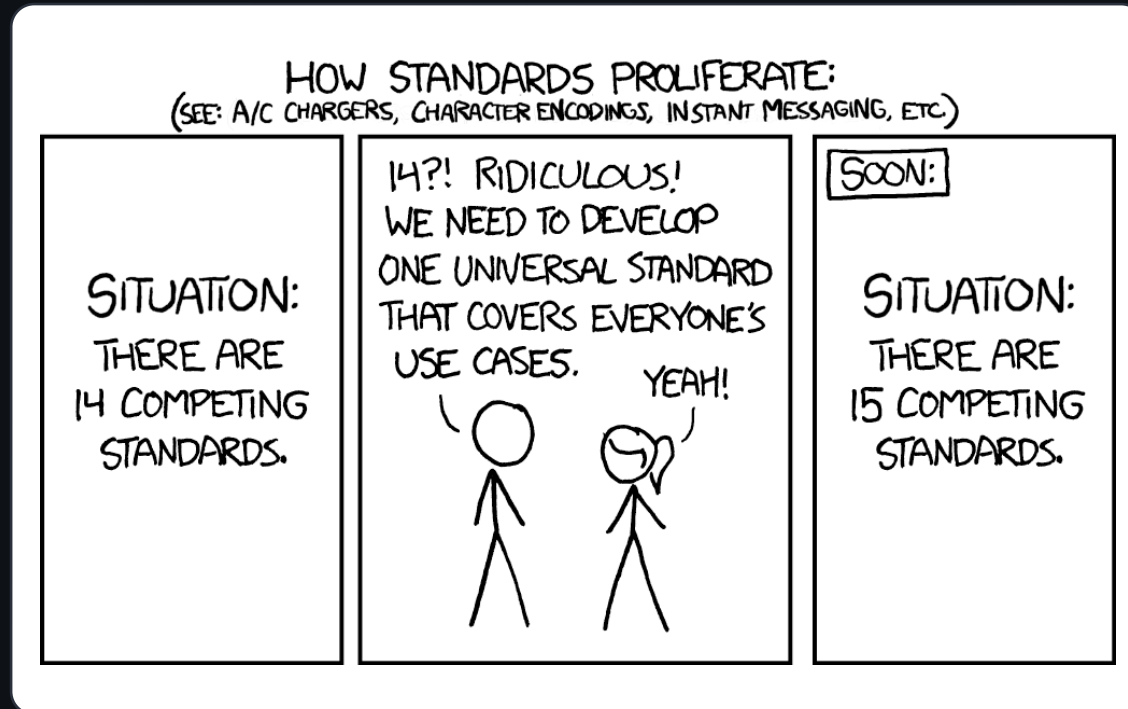


Что умеют существующие инструменты

	Скрипты	Генерация данных	Инфраструктура	Observability
pgbench	SQL-файл в цикле	TPC-B	Руками	Терминал, summary *
Sysbench	Lua	Случайные данные	Руками	Терминал, файл
HammerDB	Tcl	TPC-C, TPC-H	Руками	Устаревший интерфейс
YCSB	Java	Key-value, blob	Руками	Histogram *
Go-TPC	Исполняемый файл	Защита в код	Руками	Терминал, файл
Stroppy	Go + tagged SQL	DDL-aware генераторы	Оркестрация	Grafana, VictoriaMetrics, OTel

* метрики в формате Prometheus

Да, мы написали четырнадцатый инструмент



xkcd 927 «Standards», CC BY-NC 2.5

Что отличает Stroppy от инструментов из таблицы:

- 01 Оркестрация стенда**
Облако, машины и СУБД поднимаются по шаблону
- 02 Генерация по DDL**
Генераторы знают схему и дают одинаковые данные при одном seed
- 03 Панель и лидерборд**
Результаты хранятся онлайн и сравниваются между прогонами
- 04 Накладные расходы инструмента**
Сначала измеряются накладные расходы инструмента, затем СУБД

Архитектура Stroppy

CLI, набор бенчмарков, панель и облачный оркестратор

Компоненты Stroppy



Stroppy CLI

Один исполняемый файл на Go, около 40 МБ. Генерирует нагрузку, готовит данные и считает метрики. TPC-B, TPC-C, TPC-H, TPC-DS и банковский тест.



Набор бенчмарков

Описанные нагрузки, схемы и конфигурации стендов. Одинаковые для всех участников, с открытой методикой.



Stroppy Cloud

Панель cloud.stroppy.io хранит результаты и ведёт публичный лидерборд. Принимает своё оборудование, своё облако и свой продукт.



Graphene CI

Облачный оркестратор. Поднимает стенд и СУБД у любого провайдера и запускает на них нагрузку.

Stroppy CLI и Stroppy Cloud: принцип разделения ответственности



Сценарии → Репозиторий шаблонов

CLI выполняет сценарий нагрузки, облако хранит шаблоны стендов и нагрузок, из которых он собирается

Метрики → Стек наблюдаемости

CLI считает счётчики и гистограммы, облако принимает их по OTLP и показывает в Grafana

Параметры запуска → Управляемая инфраструктура

CLI принимает адрес СУБД и профиль нагрузки, облако поднимает стенд и подставляет их само

Результаты запуска → Публичный лидерборд

CLI отдаёт отчёт по одному прогону, облако хранит его и ставит в один ряд с другими

CLI работает сам по себе. Облако добавляет стенд, хранение и сравнение.

Цикличность истории



Picostress остаётся отдельным проектом. Ниже о том, чему научил каждый этап.

Banking test: корректность под нагрузкой



АЛИСА

было 1 000 ₹

900 ₹

Инвариант: сумма 2 000 ₹ не изменилась

Проверяется один раз, после всей нагрузки.

100 ₹



БОБ

было 1 000 ₹

1 100 ₹

01 Инвариант проверяется в конце

Сумма всех счетов после прогона равна сумме до него. Одна проверка после всей нагрузки заменяет трассировку каждой операции.

02 Три теста в одном прогоне

Производительность, отказоустойчивость и корректность измеряются вместе, на одной и той же нагрузке.

03 Производительность в условиях отказов

Отказы узлов и сети вносятся во время прогона. Числа отражают работу СУБД под сбоями, инвариант отвечает, была ли она правильной.

Фото: Stefan Brending, CC BY-SA 3.0 de; Bob Marley, 1976, public domain

Jepsen, Antithesis и FaultFS проверяют корректность, но отрывают профиль производительности от реальности.

Stroppy CLI

Один исполняемый файл: нагрузка, данные, метрики и отчёты

Встроенные тесты и драйверы

ВСТРОЕННЫЕ НАГРУЗКИ

TPC-B

TPC-C

TPC-H

TPC-DS

Banking

TPC-H dbgen и TPC-DS dsdgen встроены в нагрузки

ДРАЙВЕРЫ

PostgreSQL

MySQL

Picodata

YDB

Быстрый путь загрузки у каждого: COPY, BulkUpsert



TPC-C: 1 000 000 пользователей в один склад

PostgreSQL не справляется



TPC-DS: Аналитические запросы на полном наборе

YDB не справляется

Конфигурация и сценарий: Go и tagged SQL

```
// bench.Test: сценарий на Go
var TPCB = bench.Test{
    Name: "tpcb/tx",
    Define: func(d *bench.Define) {
        d.Step("create_schema", sql.Exec("create_accounts"))
        d.Step("load_data", gen.FromRows(accounts, 100_000))
        d.Step("workload", tx.Run("update_balance",
            "insert_history"),
            bench.Executor("constant-vus"))
        d.Step("cleanup", sql.Exec("drop_tables"))
    },
}
```

```
-- workload create_schema
-- query create_accounts
CREATE TABLE accounts (
    id INTEGER NOT NULL PRIMARY KEY,
    balance INTEGER );
-- query end

-- workload workload
-- query update_balance
UPDATE accounts SET balance = balance + :amount
WHERE id = :account_id;
-- query end
-- query insert_history
INSERT INTO history (account_id, amount, created_at)
VALUES (:account_id, :amount, CURRENT_TIMESTAMP);
-- query end
```

СЦЕНАРИЙ: ШАГИ, ИСПОЛНИТЕЛИ, ДРАЙВЕР

SQL: ДИАЛЕКТ И ЗАПРОСЫ ПОД ЗАДАЧУ

Иллюстрация структуры, не точный код из репозитория. Копия SQL-файла в рабочей директории имеет приоритет над встроенной. Параметры :name связывает драйвер.

Запуск и три счётчика

```
stropky run tpcc/tx -d pg -D url="$DB_URL" \  
  --scale-factor 16 --steps workload \  
  --executor constant-vus --vus 8 --duration 60s
```

8 executors, 60 секунд, только этап нагрузки

tps

Успешные логические транзакции в секунду

Повторы внутри одной транзакции не считаются несколькими успехами

iterations_per_second

Все завершённые итерации в секунду

Включая ошибочные. Поэтому рядом всегда стоит счётчик ошибок

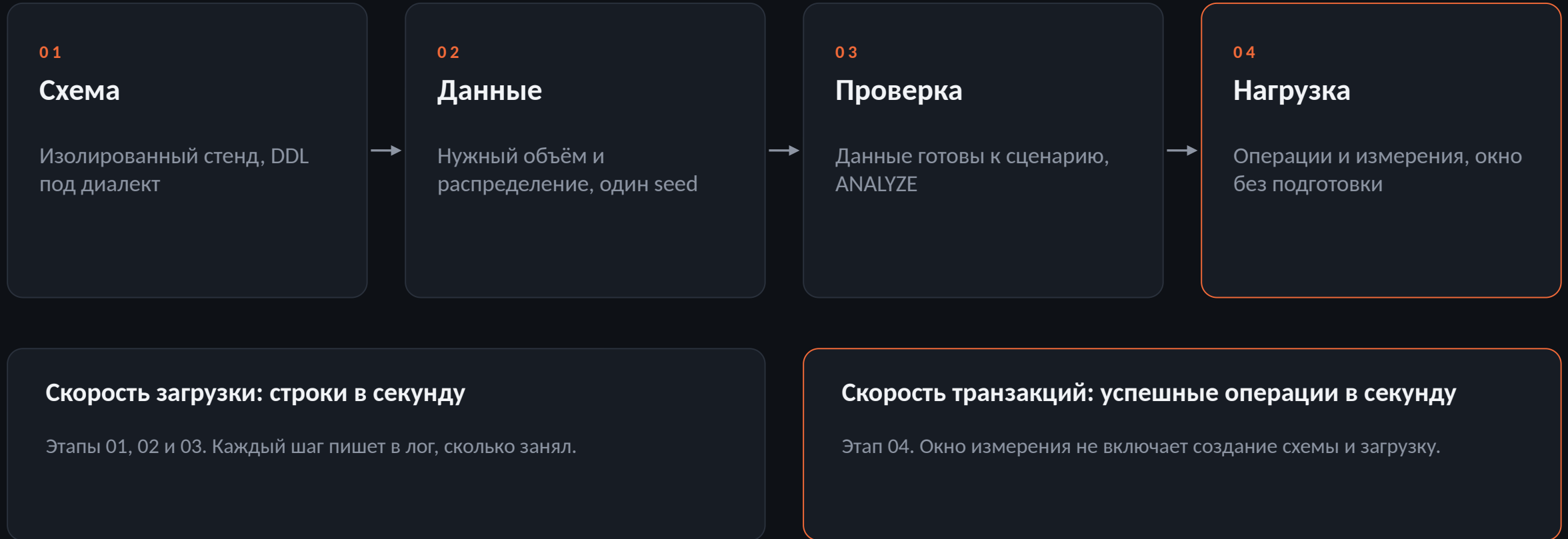
queries_per_second

Успешные запросы в секунду

Одна транзакция TPC-C это несколько запросов

Логическая транзакция не равна COMMIT. tps в Stropky не является официальным tpmC.

Явные этапы: подготовка отдельно от измерения



История первая: после bulk load СУБД продолжает работу

01

Что увидели

Нагрузка запущена сразу после загрузки данных. Числа не сходились от прогона к прогону и зависели от способа загрузки.

02

В чём причина

После bulk load PostgreSQL в фоне достраивает индексы, собирает статистику и пишет контрольные точки. Это и попадало в измерение.

03

Что изменили

Между загрузкой и нагрузкой стоит этап проверки. СУБД даётся время завершить фоновую работу, и только потом открывается окно измерения.

Нагрузка сразу после загрузки измеряет фоновое построение индексов вместо рабочего сценария.

Способ загрузки данных становится частью методики и попадает в отчёт вместе с результатом.

Панель и оркестратор

cloud.stroppy.io: результаты, лидерборд и своё оборудование

Панель: конвейер нагрузочных тестов



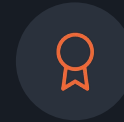
Автоматизация

Запуск по шаблону, очередь прогонов и квоты на ресурсы. Инструмент для команд разработки СУБД.



Своё оборудование, свой продукт

Своё облако, свой стенд или своя СУБД. Панель запускает ту же нагрузку на том, что принёс участник.



Лидерборд

Результаты хранятся онлайн и видны всем. Прогон этой недели лежит рядом с прошлым.

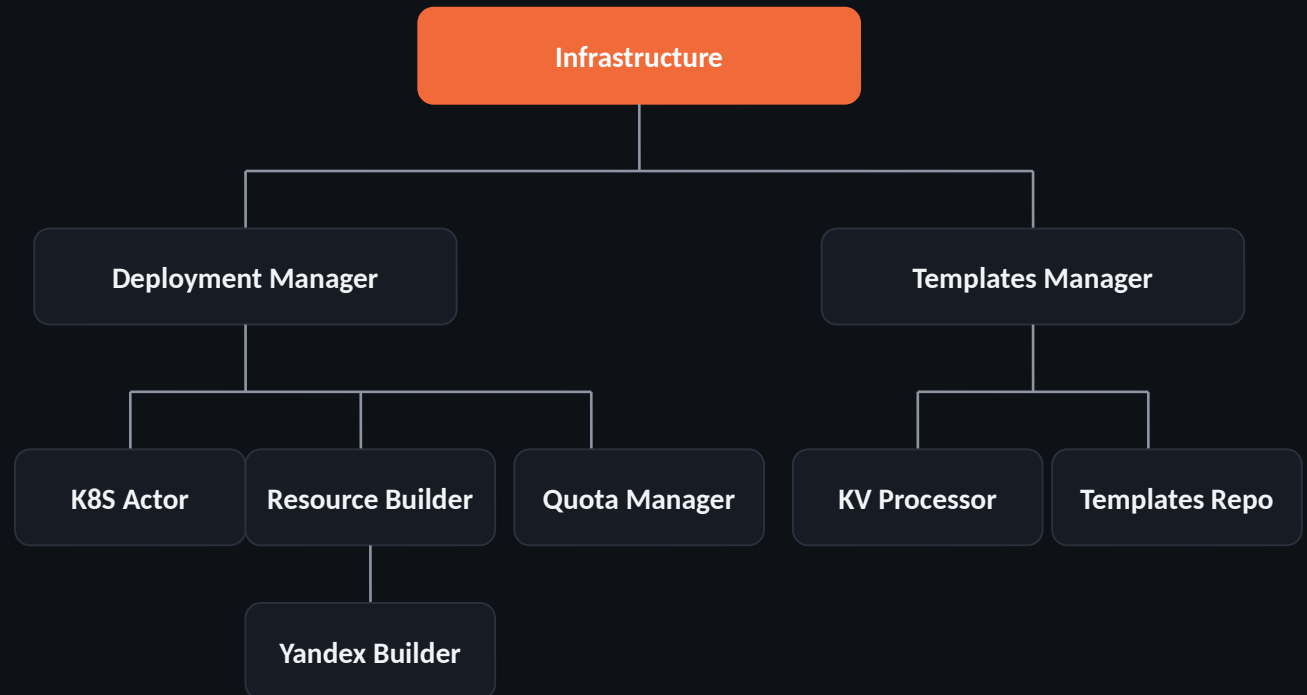
cloud.stroppy.io: оценка производительности и стабильности СУБД, доступная после регистрации.

strop.py

И

Как устроена панель

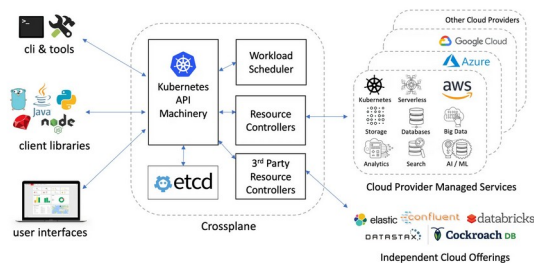
- Кластер Kubernetes, развёрнутый в облаке
- В кластере Crossplane с provider Yandex Cloud
- PostgreSQL рядом с панелью хранит состояние
- Backend на Go, frontend на React
- Deployment Manager разворачивает ресурсы в облаках
- Quota Manager ограничивает потребление ресурсов
- Template Manager превращает запрос в набор ресурсов



Инфраструктурные зависимости backend панели

Graphene CI: оркестратор в любом облаке

- Основа: Crossplane, проект CNCF
- Декларативное управление инфраструктурой через композитные ресурсы
- Любое облако через provider Crossplane
- Поднимает стенд, разворачивает СУБД, запускает нагрузку и освобождает ресурсы после прогона



СУБД, КОТОРЫЕ РАЗВОРАЧИВАЕТ GRAPHENE CI

PostgreSQL

OrioleDB

YDB

Greenplum

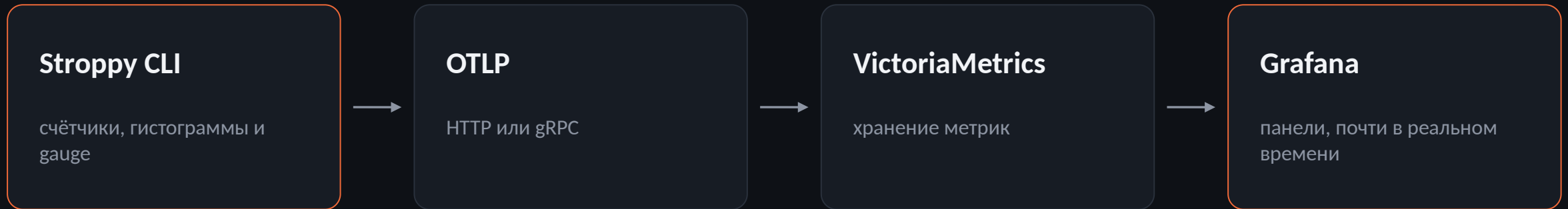
TiDB

YugabyteDB

CockroachDB

Новая СУБД добавляется шаблоном развёртывания, без изменений в панели. Вендор может принести свой шаблон.

Наблюдаемость: от CLI до Grafana



Локальный отчёт всегда

Резюме в stderr и JSON в ~/.stropPy/reports. Экспорт по OTLP необязателен. Отчёт формируется в любом случае.

Один идентификатор на запуск

service.instance.id связывает метрики, отчёт и строку в панели. Метаданные запуска без секретов.

История вторая: disk footprint после bulk load

в разы

больше объёма данных занимал диск после загрузки

Disk footprint OrioleDB после bulk load многократно превышал объём загруженных данных. Данные те же, разница только в том, что осталось на диске.

Симптом

Инструмент считает не только транзакции, но и место на диске после каждого этапа

Причина

Ошибка в продукте, найденная по воспроизводимому сценарию загрузки

Результат

Исправлено в OrioleDB. Повторный прогон подтвердил исправление

Бенчмарк полезен и тогда, когда вместо числа находит ошибку.

Доверие и прозрачность

Сначала накладные расходы инструмента, затем СУБД. Результаты открыты целиком

Жизнь с Grafana k6

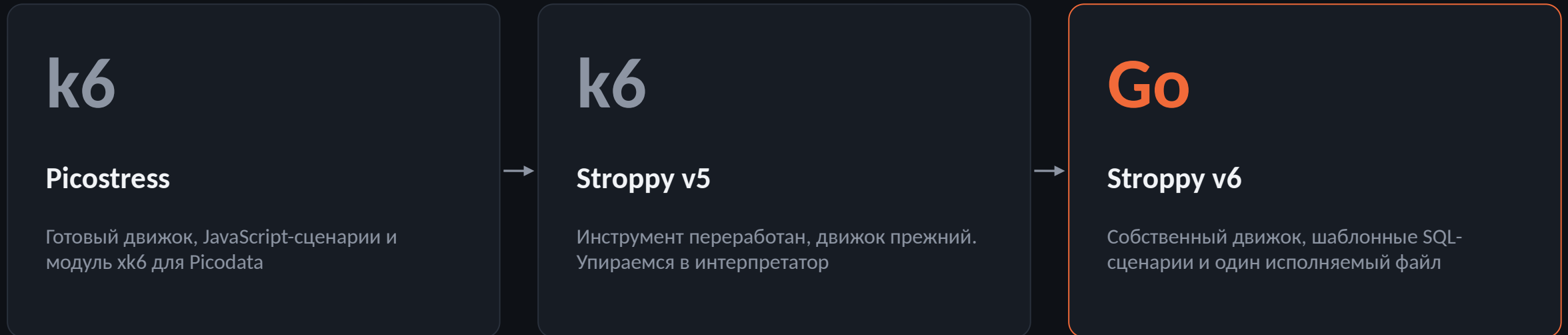
ГОТОВЫЙ МИР

- Экосистема Grafana
- Сценарии виртуальных пользователей, VU
- Executors: фиксированное время или фиксированный рейт
- Управление жизненным циклом из коробки
- Метрики и отчёты

ЕГО ЦЕНА

- «Вы очень медленные»
- Нужно опускаться до перехвата `os.Args` и `os.Exit`
- Добавить функцию значит менять семь слоёв
- Интерпретатор без JIT
- Жизненный цикл из трёх этапов, а нужно пять
- «Вы очень медленные»

Три переписывания ради нулевых накладных расходов



Инструменту верят, когда у него нет собственных накладных расходов. Это и было целью каждого переписывания.

Само переписывание ничего не доказывает. Дальше о том, как измеряются накладные расходы инструмента.

Жизнь после k6

33,3x

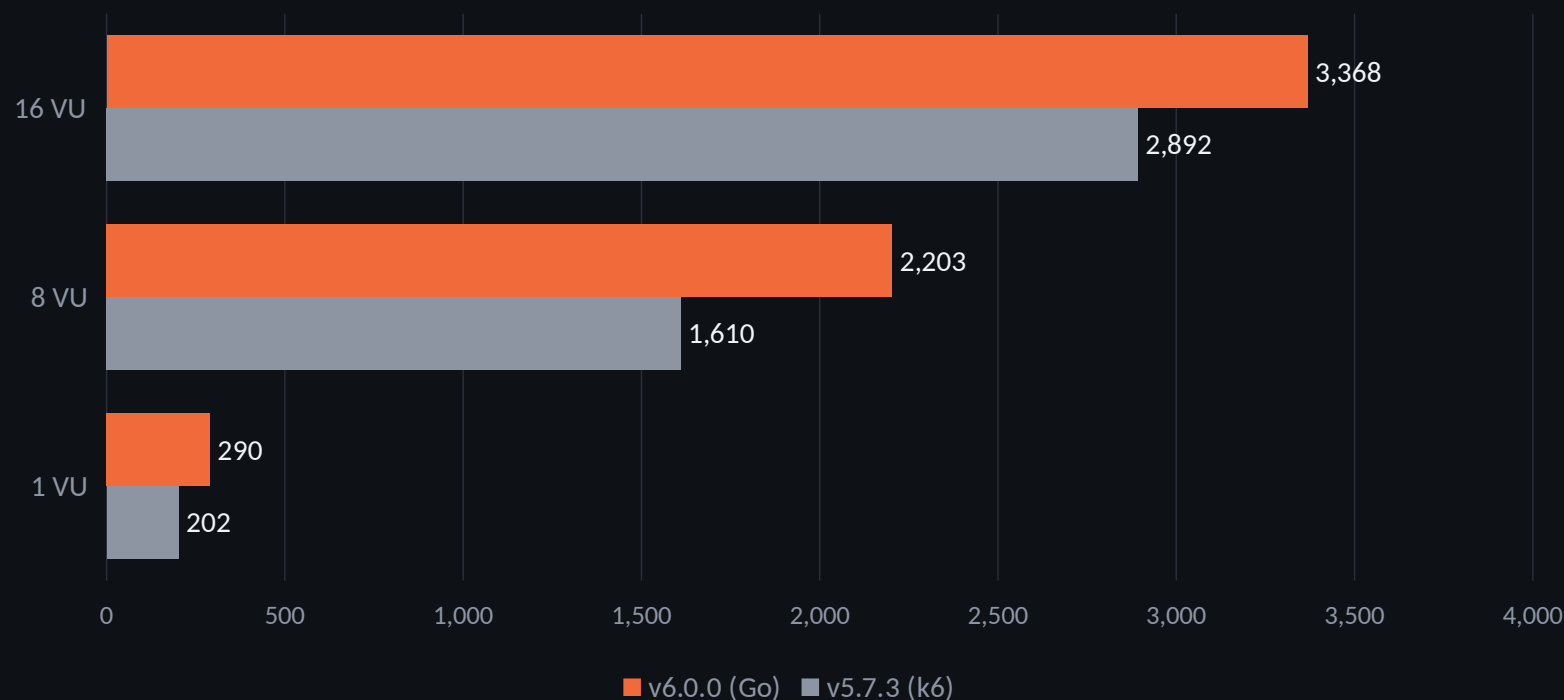
TPC-B, поор, 1 VU: 4 368 → 145 524
итераций/с

13,4x

TPC-B, поор, 24 VU: 38 694 → 517 627
итераций/с

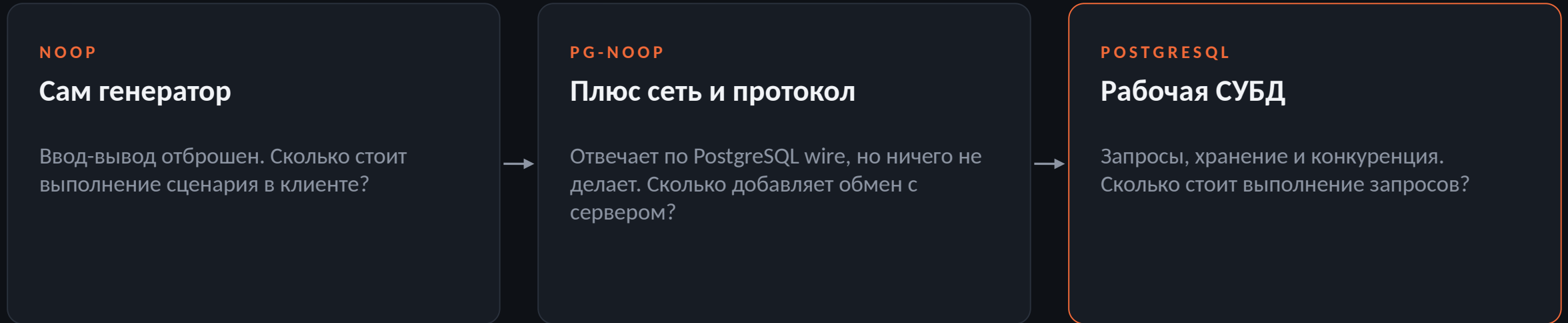
Без сети и СУБД. Это предел самого
генератора.

TPC-C на PostgreSQL 16, завершённые итерации/с



На заглушке выигрыш большой, на PostgreSQL небольшой. Так и должно быть.

/dev/null вместо СУБД



Идеальная СУБД работает со скоростью сети. Идеальной СУБД нет, но зазор до /dev/null и есть её цена.

stropky baseline прогоняет первые два уровня на вашей машине и показывает, сколько запаса оставляет клиент. Это не лидерборд СУБД.

Два часа под нагрузкой: загрузка и транзакции

Data load

4,690,210,000 rows · load_data wall time

Noop

4.11 min to load

19.01M

pg-noop

8.07 min to load

9.69M

PostgreSQL 18.6

41.03 min to load

1.91M

0 5 10 15 20 25
M rows / second

Transactions

Successful logical transactions / second · full-run average

Noop

avg 0.129ms · p99 2.5ms

239,443

pg-noop

avg 0.269ms · p99 1ms

117,992

PostgreSQL 18.6

avg 3.749ms · p99 25ms

8,528

0 60 120 180 240 300
K successful transactions / second

`/dev/null` принимает 19 млн строк в секунду, PostgreSQL около 2 млн. В транзакциях клиент оставляет СУБД запас в 28 раз.

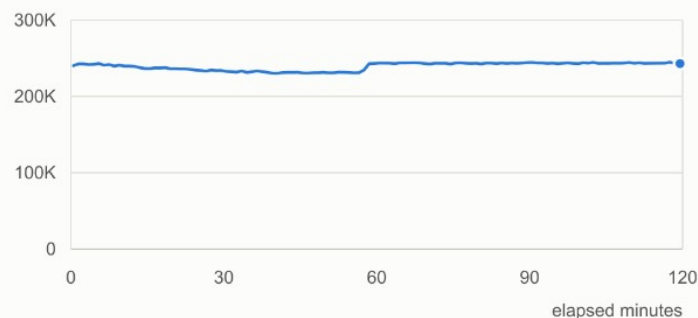
p99: 2,5 мс у noop, 1 мс у pg-noop и 25 мс у PostgreSQL. Ошибочных итераций ноль. Один прогон на цель.

TPC-C, 10 000 складов, 32 executors, два часа на цель. Два AMD EPYC 9374F, 751 GiB RAM, NVMe и отдельные 32-ядерные наборы для клиента и сервера. 4,69 млрд строк, 6 октября 2026

Два часа под нагрузкой: пропускная способность по минутам

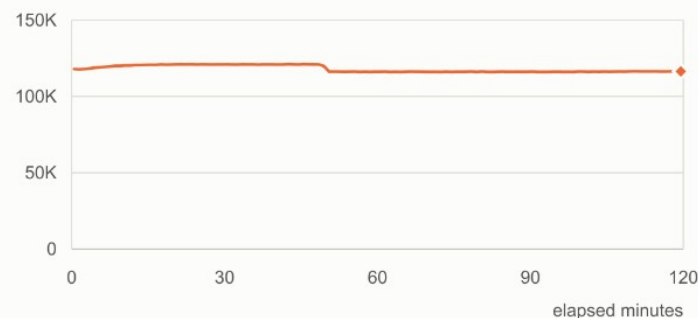
Noop

successful tx/s · scale 0–300,000



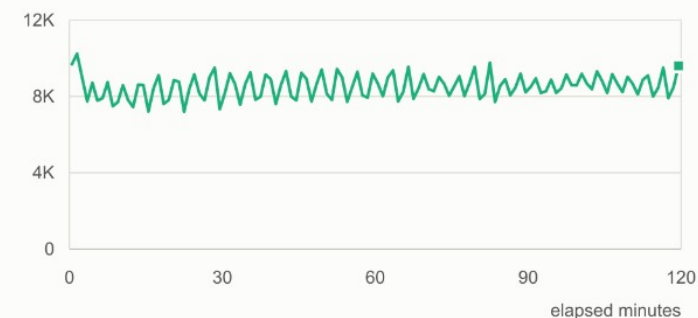
pg-noop

successful tx/s · scale 0–150,000



PostgreSQL 18.6

successful tx/s · scale 0–12,000



Noop

Ровная линия на всём интервале, от 230 до 245 тыс. транзакций в секунду. Это предел клиента

pg-noop

Такая же ровная линия, от 116 до 121 тыс. в секунду. Разница с noop и есть цена протокола и сети

PostgreSQL 18.6

Пропускная способность колеблется от 7 до 10 тыс. в секунду, p99 растёт до 25 мс

Интервалы около минуты, один прогон на цель и отдельные шкалы от нуля. Публикуется весь прогон целиком. Подробности: stroppy.io/blog

История третья: warmup как метрика

Старт нагрузки

холодная раскладка таблиц

Часы resharding

продукт подстраивает раскладку под нагрузку

Установившийся режим

номинальная скорость

Ответ разработчиков: «Так задумано»

01

Это свойство продукта

Ошибки нет ни в тесте, ни в продукте.
Адаптивный resharding остаётся допустимым инженерным решением.

02

Значит, это метрика

Время до установившегося режима
измеряется и публикуется отдельным числом
рядом с пропускной способностью.

03

Две точки в лидерборде

Холодный старт и установившийся режим.
Каждая смена профиля нагрузки в
эксплуатации повторяет холодный старт.

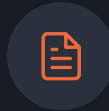
«Так задумано» отвечает инженеру. Бенчмарк отвечает пользователю, сколько часов до номинальной скорости.

Что всплыло, когда инструмент перестал быть узким местом



PGwire

Реализации протокола расходятся в деталях



Журналирование

Под нагрузкой становится отдельной нагрузкой



Spinlock

Ожидания в драйверах и пулах соединений



COPY

Единственный быстрый путь загрузки



Горизонты транзакций

Долгие прогоны и накопление версий



Bulk inserts

Ограничения на число параметров в запросе

По каждому пункту есть короткая история из практики.

Публичный лидерборд на cloud.stroppy.io

Stroppy CloudLeaderboardArchitectureDocsENRUSign inGet started

Public rating

Runs their authors made public. Numbers are compared only within one load scenario, and each row says what hardware produced it.

ScenarioDatabaseVersionProviderStroppyHardware7d30d90d1yAll time

TPC-C 5 results - Numbers compare only within one scenario; each row shows its hardware

#	Run	Database	Workload	Throughput	Hardware	Date
1	Stroppy Team Public report	postgres 18 postgres - 2 nodes	TPC-C 100 WH - 128 VU - 10 min (pg) TPC-C - stroppy 6.1.0	751 tps	1 x 16 vCPU - 64 GB yandex	1 hour ago
2	Stroppy Team Public report	postgres 17 postgres - 2 nodes	TPC-C 100 WH - 128 VU - 10 min (pg) TPC-C - stroppy 6.1.0	688 tps	1 x 16 vCPU - 64 GB yandex	2 hours ago
3	Stroppy Team Public report	postgres 17 postgres + 1 replica + haproxy - 4 nodes	TPC-C 100 WH - 128 VU - 10 min (pg) TPC-C - stroppy 6.1.0	598 tps	2 x 16 vCPU - 64 GB + 1 x 4 vCPU - 8 GB yandex	34 minutes ago
4	Stroppy Team Public report	mariadb 11.4 mariadb - 2 nodes	TPC-C 100 WH - 128 VU - 10 min (mysql) TPC-C - stroppy 6.1.0	422 tps	1 x 16 vCPU - 64 GB yandex	5 minutes ago
5	Stroppy Team Public report	mysql 8.4 mysql - 2 nodes	TPC-C 100 WH - 128 VU - 10 min (mysql) TPC-C - stroppy 6.1.0	386 tps	1 x 16 vCPU - 64 GB yandex	2 hours ago

В лидерборд попадают прогоны, которые авторы сделали публичными. Числа сравниваются только внутри одного сценария, и у каждой строки указано оборудование, на котором она получена.



cloud.stroppy.io/leaderboard

Свой прогон добавляется кнопкой «Get into the rating». Та же нагрузка, своё оборудование и строка в общем лидерборде.

Доверие, основанное на прозрачности



Открытое ПО

Apache 2.0: код CLI и панели, бенчмарки, методика и сырые результаты каждого прогона



Открытая архитектура

Оборудование, облако, СУБД или сценарий. Вендору и энтузиасту для участия нужна одна команда запуска



Открытый лидерборд

Результаты видны всем и воспроизводимы. Спор о числах заменяется повторным прогоном

Объективный бенчмарк не пишется в одиночку: его делает сообщество.



Стандарт №14. Ну разве он не прекрасен? Давайте сделаем его ТАКИМ ВМЕСТЕ



stroppy.io

документация, бенчмарки и блог



cloud.stroppy.io

Stroppy Cloud: панель и лидерборд



github.com/stroppy-io/stroppy

исходники и релизы



Discord

сообщество

Поставьте звезду и сделайте форк: github.com/stroppy-io/stroppy



cloud.stroppy.io/leaderboard

Публичный лидерборд

Спасибо. Вопросы?

Резерв: методика сравнения v5 и v6

01

Один стенд, разные наборы ядер
KVM, EPYC Genoa. Клиент: 16 ядер, GOMAXPROCS=16. PostgreSQL: 8 ядер

02

Общий dataset и масштаб
PostgreSQL 16.15, TPC-C SF=16, pool = VU; fsync и synchronous_commit включены

03

Пять прогонов по 20 секунд
Warmup 5 с, порядок версий чередуется, медиана и диапазон min...max

VU	v5.7.3, итераций/с	v6.0.0, итераций/с	Прирост
1	201,60 [199,00 ... 202,80]	290,30 [276,50 ... 291,35]	+44%
8	1 609,75 [1 608,30 ... 1 625,15]	2 203,15 [2 180,50 ... 2 247,65]	+37%
16	2 892,10 [2 861,80 ... 2 899,15]	3 368,35 [3 203,75 ... 3 394,25]	+16%

Завершённые итерации смеси в секунду, медиана и диапазон пяти прогонов. Не COMMIT/с и не tpmC. Сравнение релизов целиком. Эффект от удаления k6 отдельно не выделен.