YDB

Методы и подходы к тестированию распределённой базы данных

Виталий Гриднев, YDB

Ведущий разработчик, руководитель команды выполнения запросов

Что такое YDB?

Распределённая SQL-база данных для операционных и аналитических нагрузок

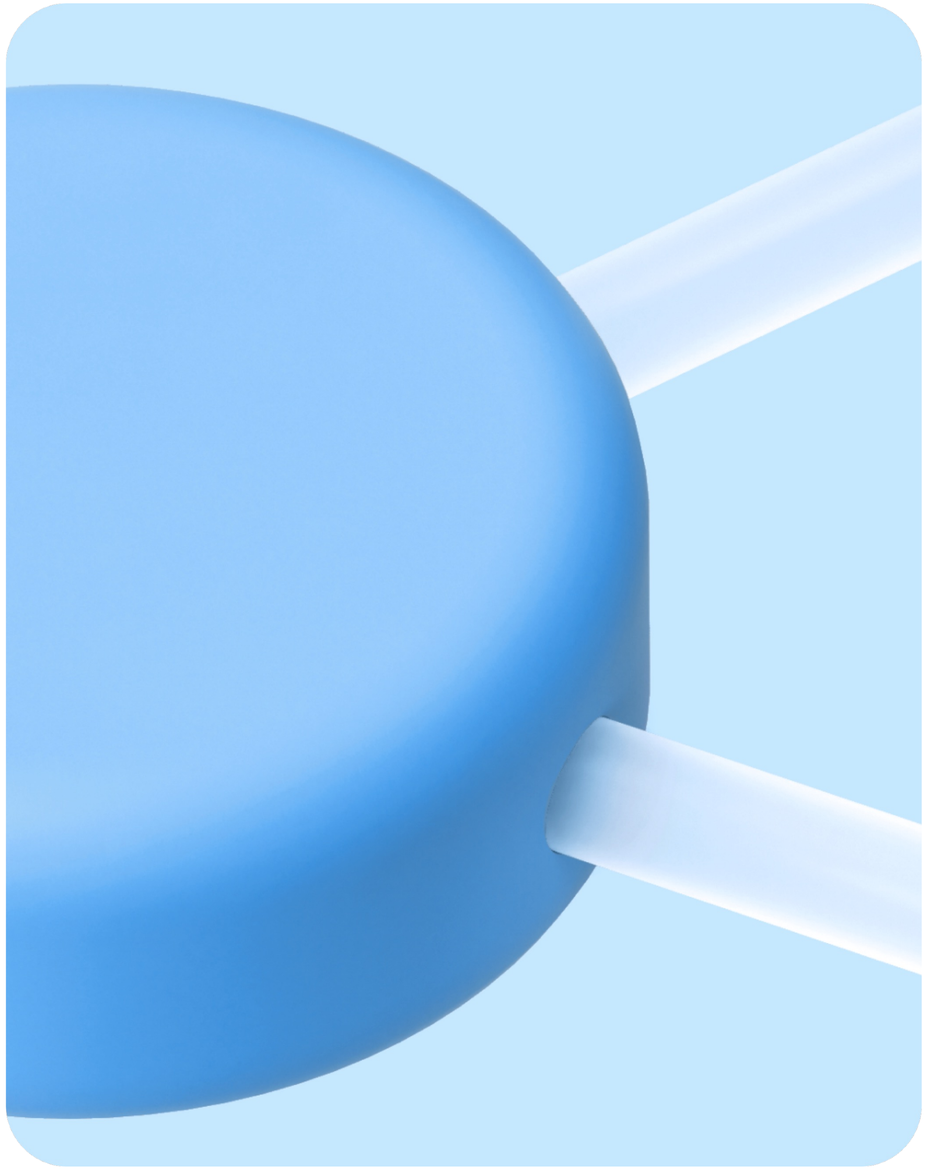
- Горизонтальное масштабирование на миллионы транзакций в секунду и петабайты данных
- Работоспособность и автоматическое восстановление при отказах
- Serializable-ACID-транзакции
- Open Source с лицензией Apache 2.0

ydb.tech/ru

Где используется

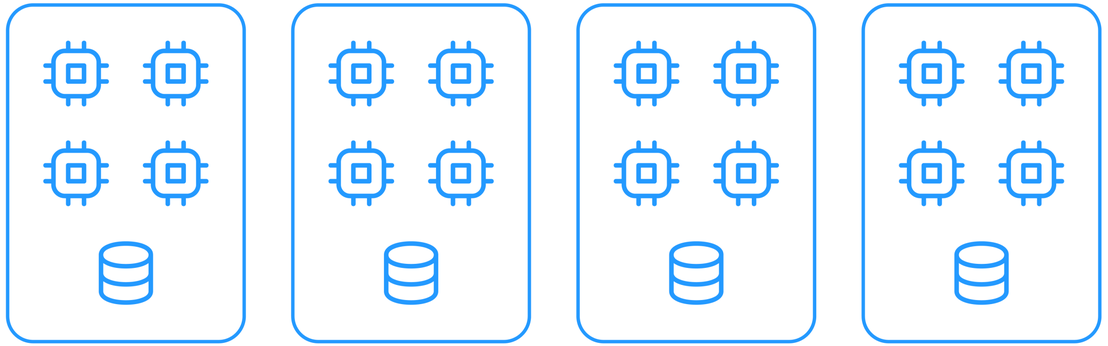
Банк	• Продуктовый движок • Инфраструктура
ID	• Яндекс Паспорт • Различные сервисы продуктового движка
Маркет	• 100х кратный рост нагрузки • 99,99% требования по доступности
Метрика	• Хранения и обработки событий • Рост транзакций в 100+ раз: от 10 тыс. TPS до 1,5+ млн RPS • Рост объёма данных: от 1 ТВ до 1500+ ТВ
Платежная система Яндекс	• >1000 платежей в секунду • Повышенные требования к надёжности и отказоустойчивости
Yandex Cloud	• Конфигурационная информация сервисов Yandex Cloud • Данные биллинга, включая ведение метрик

Особенности



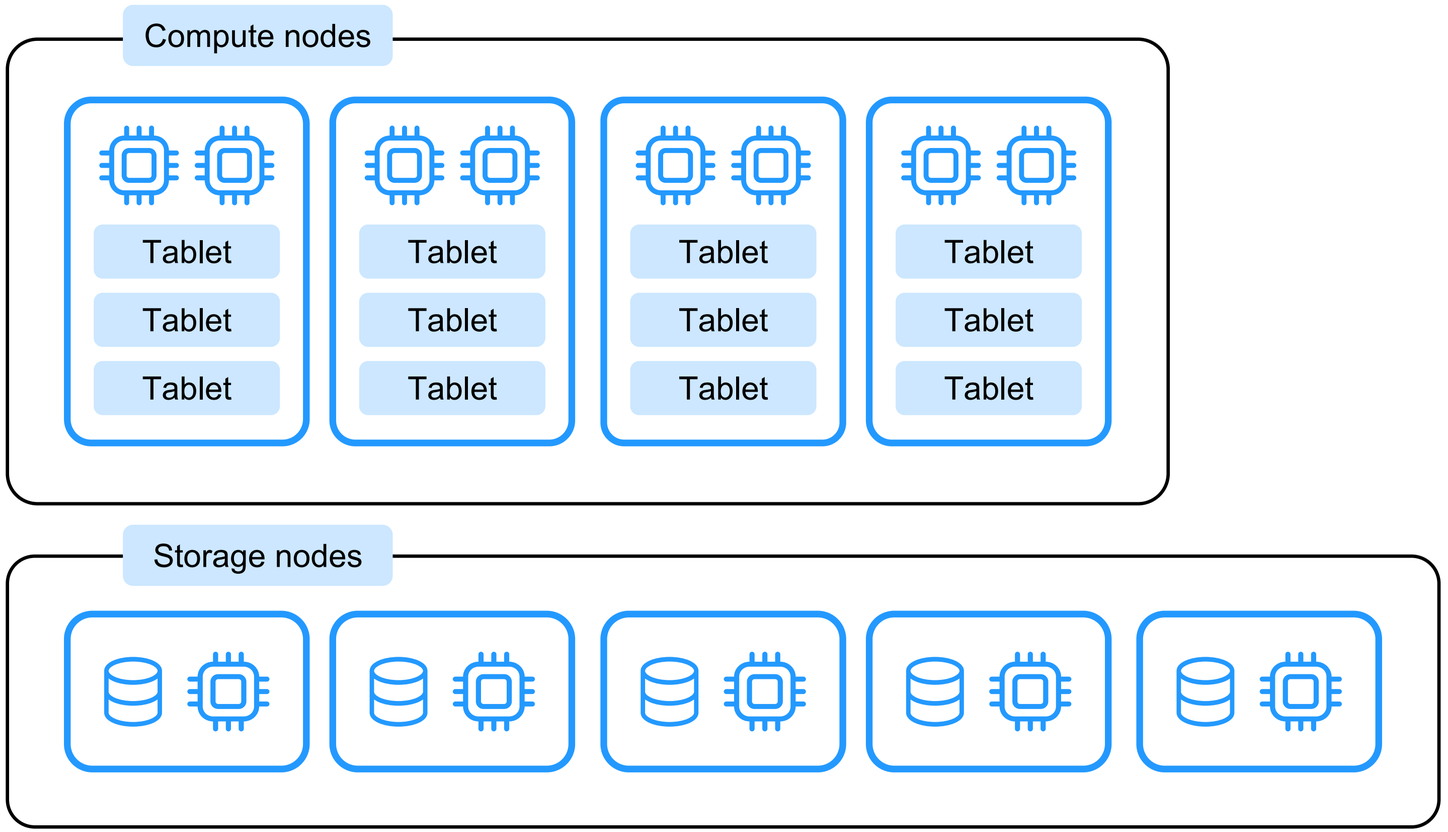
Физическая архитектура

- Кластер физических или виртуальных машин, K8s с контейнерами
- Обычные серверы
- Кластер хранит данные и обрабатывает пользовательские запросы



Разделение слоёв Compute и Storage

- Среды выполнения для таблеток и запросов запущены на вычислительных узлах
- Данные размещены на узлах хранения

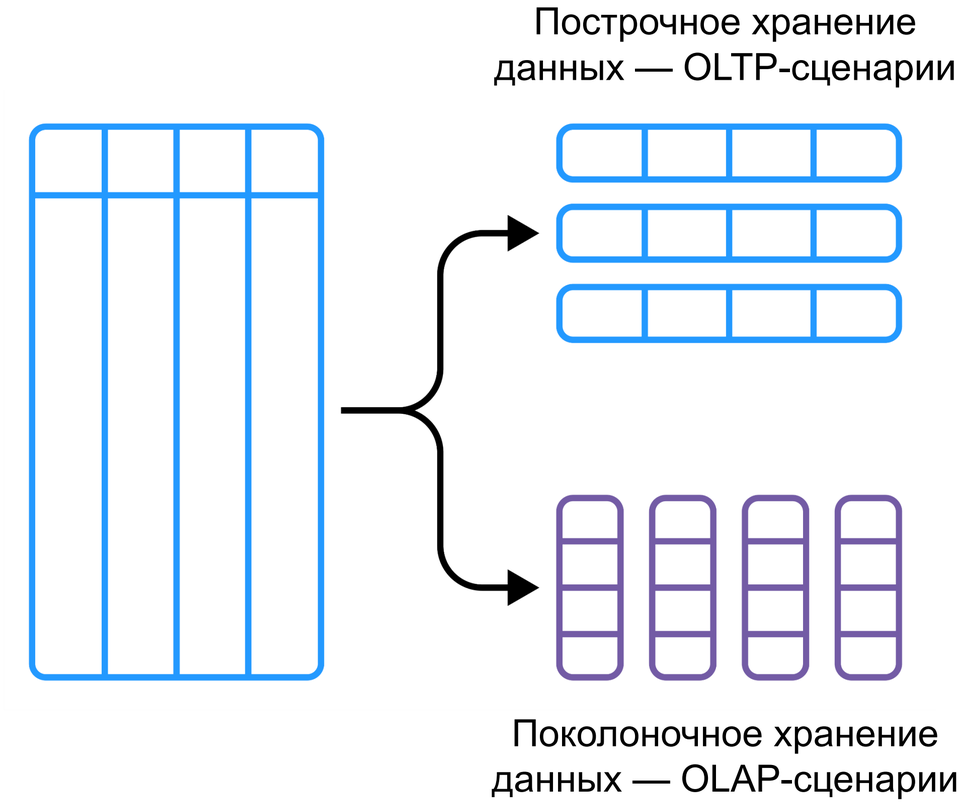


Отказоустойчивость

Переживает потерю зоны доступности и одной серверной стойки в любой другой зоне

Колоночные и строчные таблицы

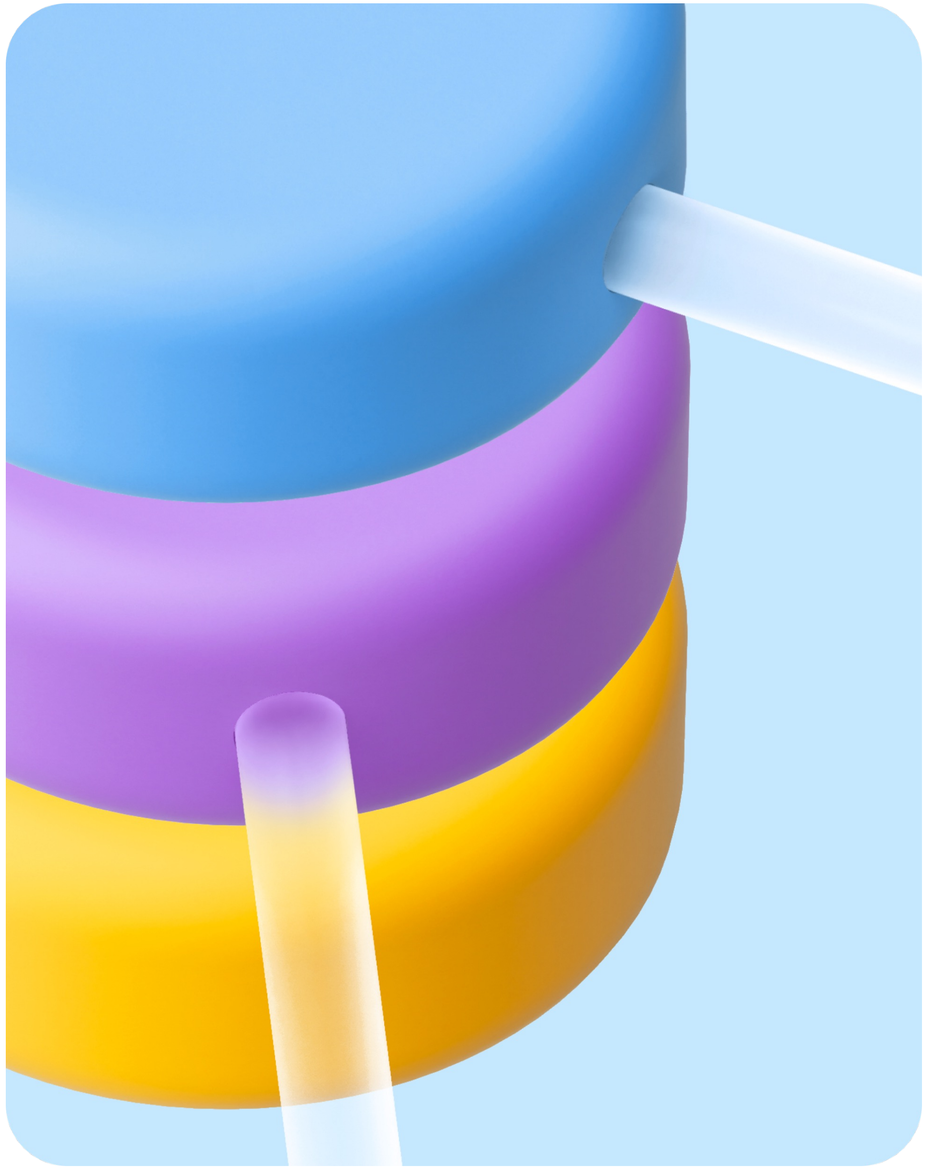
- Логические двумерные таблицы можно располагать на диске и в памяти либо построчно, либо поколоночно
- Строчное хранение подходит для OLTP-нагрузок, когда необходимо обрабатывать большой поток транзакций
- Колоночное хранение подходит для OLAP-сценариев, когда нужно анализировать большие объёмы данных



clck.ru/39stTm

Сложная система...

Как же её протестировать?



Юнит-тестирование

1. Простые в запуске
2. ~3100 юнит-тестов написано на C++
3. Может быть достаточно мощным и выполнять перебор значений
4. Просто внедрить ошибку или «подыграть» какой-то сценарий
5. Позволяет избегать регрессий

Юнит-тестирование

Software	% of failures reproducible by unit test
Cassandra	73% (29/40)
HBase	85% (35/41)
HDFS	82% (34/41)
MapReduce	87% (33/38)
Redis	58% (22/38)
Total	77% (153/198)

clck.ru/39suge

Юнит-тестирование

80%

проблем можно предотвратить юнит-тестами

Интеграционное тестирование

1. Запускает реальный кластер базы данных локально
2. Позволяет моделировать сценарии отказоустойчивости
3. Написано на Python с использованием Pytest
4. Позволяет протестировать end-to-end сценарии
5. Используют файлы для имитации диска или режим хранения только в памяти

Интеграционное тестирование

```
def test_restart_single_node_is_ok(self):

    create_tablets_and_wait_for_start(self.cluster.client)

    for node_id, node in self.cluster.nodes.items():
        # Act
        node.stop()
        logger.info("Node is stopped %s", node_id)
        # Assert
        wait_tablets_are_active(self.cluster)

        # Act
        node.start()
        logger.info("Node is started %s", node_id)
        # Assert
        wait_tablets_are_active(self.cluster)
```

На данный момент

- **Юнит-тестирование**
- **Интеграционное тестирование**

Санитайзеры

AddressSanitizer

Инструмент для поиска ошибок
обращения к памяти для C/C++

MemorySanitizer

Инструмент для поиска использования
неинициализированной памяти

ThreadSanitizer

Инструмент для поиска ошибок
Data Race или Deadlock (C/C++, go)

https://github.com/google/sanitizers

Санитайзеры

```
ERROR: AddressSanitizer: heap-use-after-free on address ..

READ of size 8 at 0x6060006128d8 thread T13 (ydbd.User) .....

\#1 0x36eb65c8 in NKikimr::NKqp::TKqpQueryCache::Insert.....

\#2 0x36e9a77d in NKikimr::NKqp::TKqpCompileService::Handle.....
\#3 0x36e987c6 in
NKikimr::NKqp::TKqpCompileService::MainState....
```

Санитайзеры

use-after-free

Возникает в C++ коде если память продолжает использоваться уже после ее освобождения

Уязвимость

Может повлечь за собой выполнение производного кода или падение процесса по segfault

Санитайзеры

```
Test crashed (return code: 100)
ERROR: LeakSanitizer: detected memory leaks
Indirect leak of 16759 byte(s) in 1 object(s) allocated from:
....
#4... in grpc_core::Arena::CreateWithAlloc
...
#9... in grpc::Channel::CreateCall
...
#13...in Ydb::Table::V1::TableService::Stub::AsyncDeleteSession
```

Санитайзеры

Неконтролируемые утечки памяти могут привести к падению процесса из-за превышения лимита на память

Санитайзеры

clck.ru/39sv5U

Не бесплатны, так как требуют:

- отдельных сборок с измененными флагами компиляции
- дополнительной памяти
- нужны ресурсы на запуск

Жутко тормозят

Но крайне важны:

~70% уязвимостей, которым Microsoft ежегодно присваивает статус CVE, это проблемы с безопасностью памяти

Санитайзеры

70%

уязвимостей, которым Microsoft ежегодно присваивает статус CVE это проблемы с безопасностью памяти

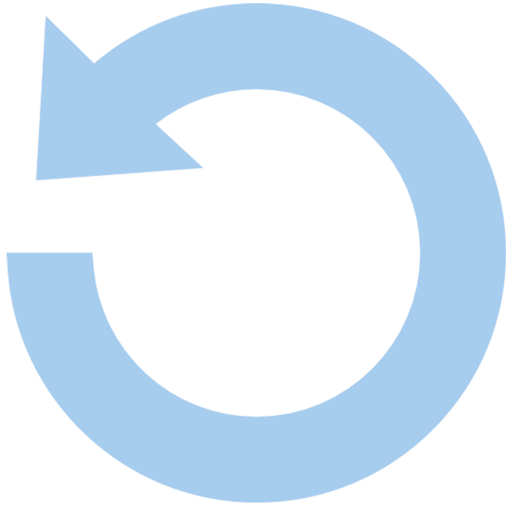
Query replay

Собираем информацию о запросах с PROD

- Текст запроса
- Схема таблиц
- Статистики
- Настройки выполнения

Позволяет без запуска системы проверить

- Парсится ли SQL запрос?
- Нет ли деградации в плане выполнения?
- Новый full scan в запросе?



Query replay: сравнение планов запросов

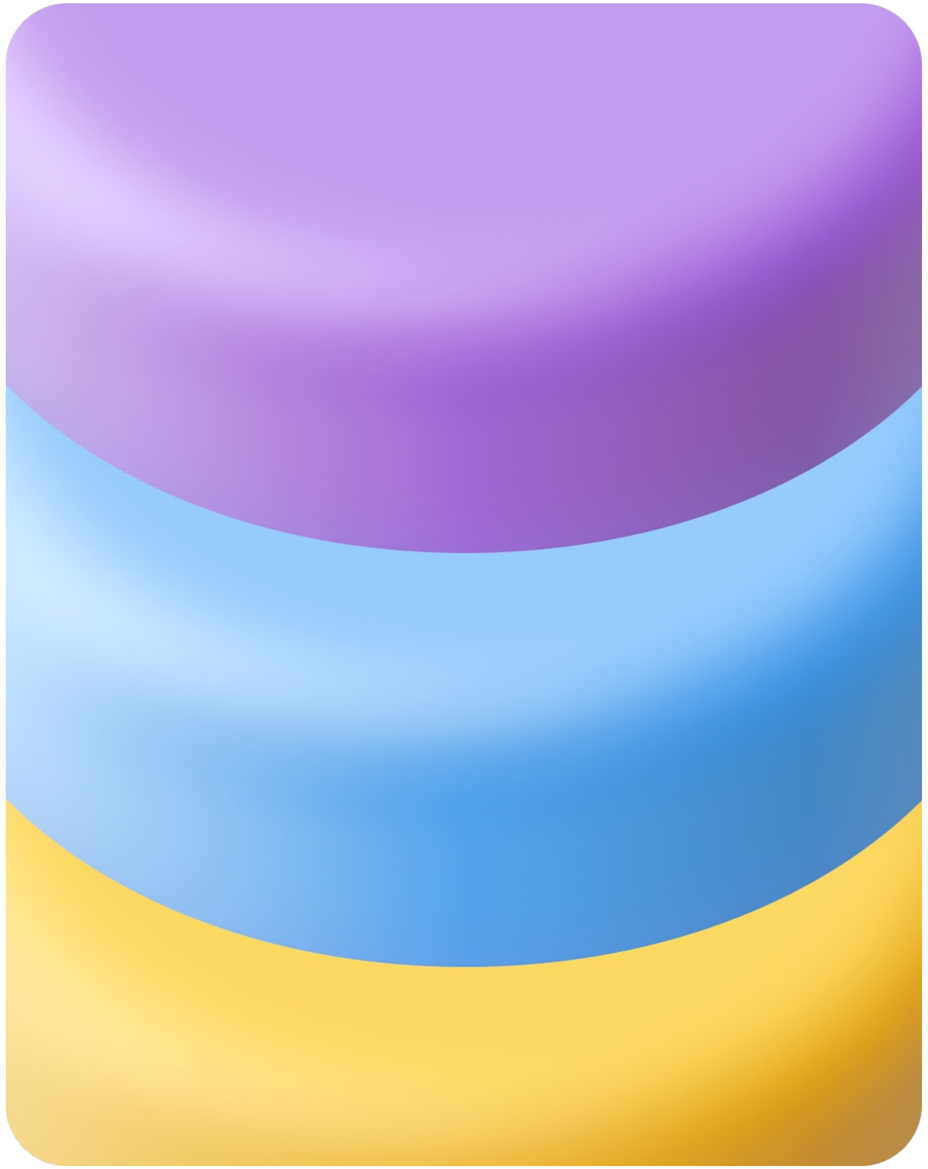
```
  },
"tables":
  [
    {
      "name":"..../DeviceGroups",
      "reads":
        [
          {
            "columns":
              [
                "device_id",
                "group_id",
                "uid"
              ],
            "scan_by":
              [
                "uid",
                "device_id",
                "group_id"
              ],
            "type":"FullScan"
          }
        ]
    },
```

```
    [
      {
        "name":"..../DeviceGroups",
        "reads":
          [
            {
              "columns":
                [
                  "device_id",
                  "group_id",
                  "uid"
                ],
              "type":"Lookup"
            }
          ]
```

На данный момент

- Юнит-тестирование
- Интеграционное тестирование
- Санитайзеры
- Query Replay

Как ещё протестировать?

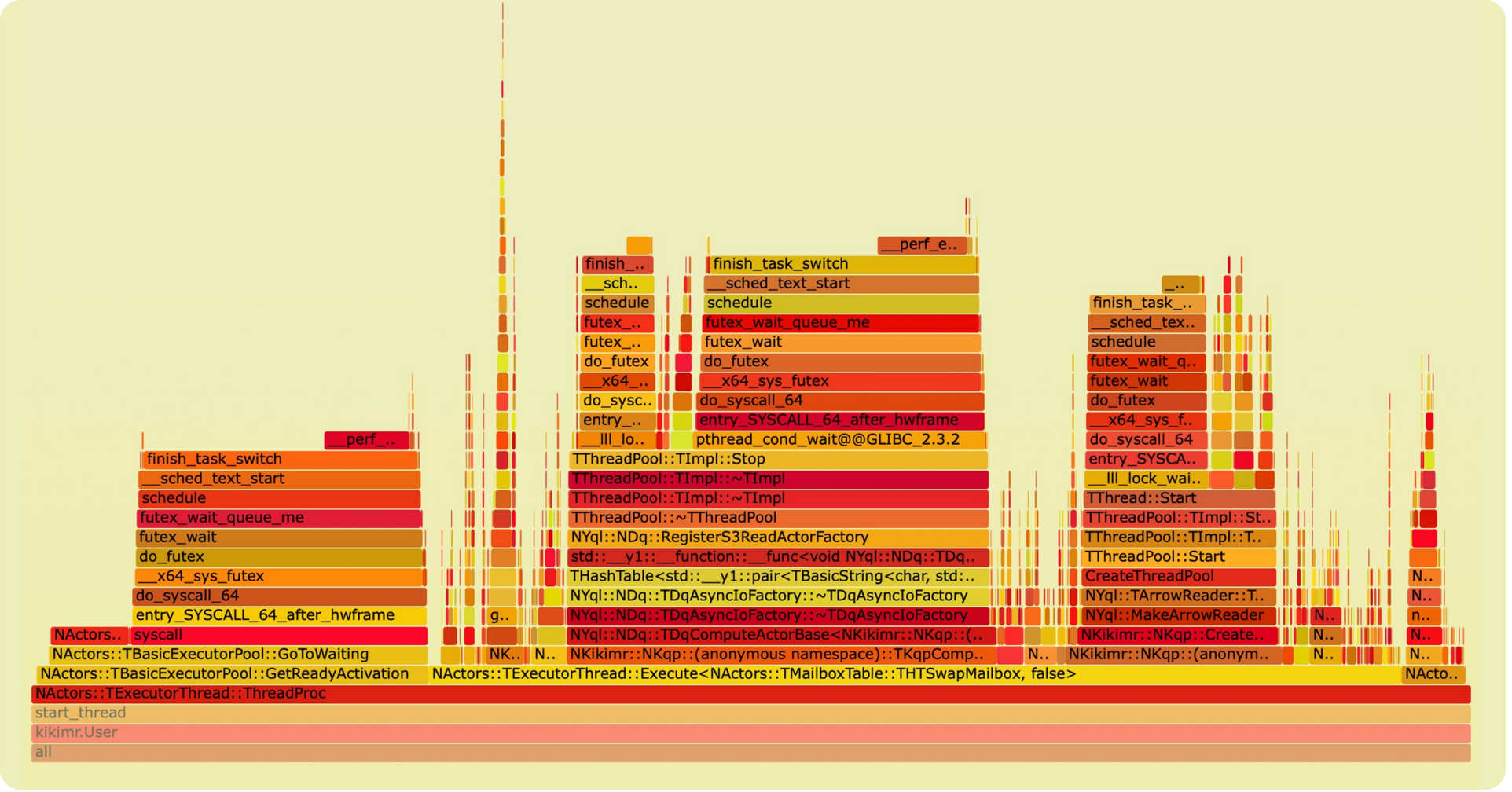


История о баге с тред-пулом

- Разработчик написал фичу и залил её в main
- Команда замечает, что запросы выполнятся в разы медленнее (×10 раз)
- Собираем программу с дополнительным флагом сборки -fno-omit-frame-pointer
- Собираем perf record с кластера
- С помощью визуализатора готовим FlameGraph

clck.ru/34mW4L

История о баге с тред-пулом

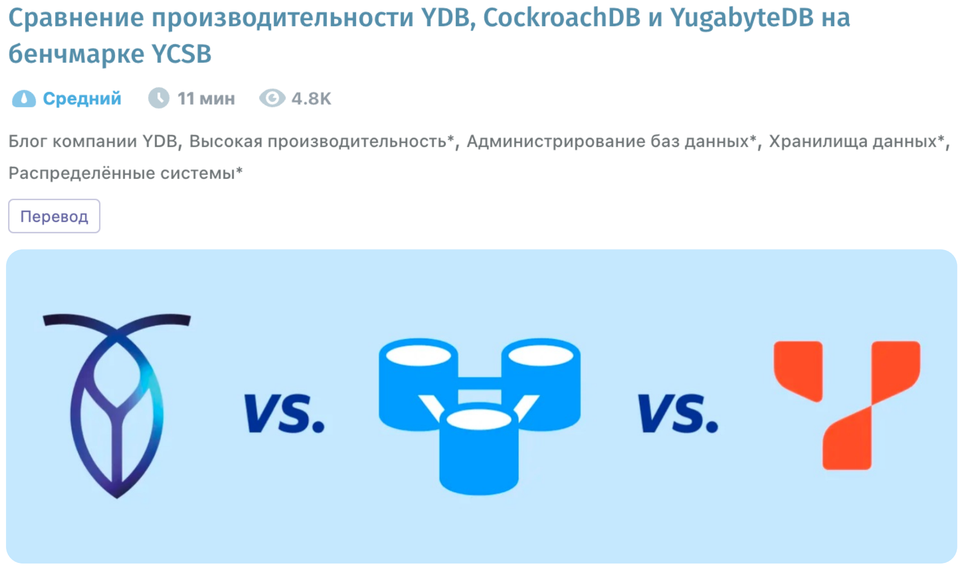


Performance tests

1. Сравнение с конкурентами side-by-side на публичных бенчмарках YCSB, ClickBench, TPC-C

2. Поиск регрессий производительности за счёт запуска на стандартном одинаковом «железе»

3. Анализ проблем производительности
   - FlameGraph
   - Линии в мониторинге для поиска узких мест
   - OpenTelemetry-трейсы

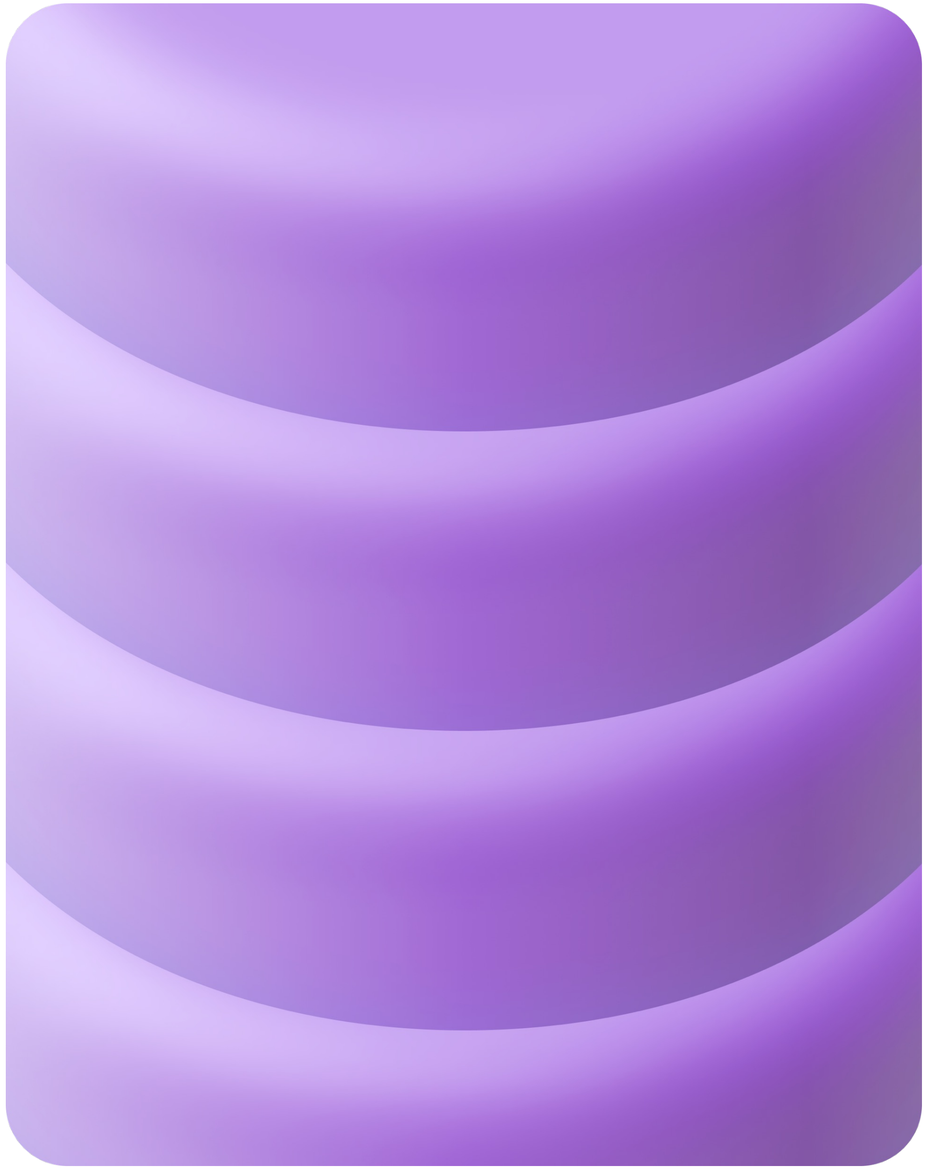


clck.ru/35cYB9

На данный момент

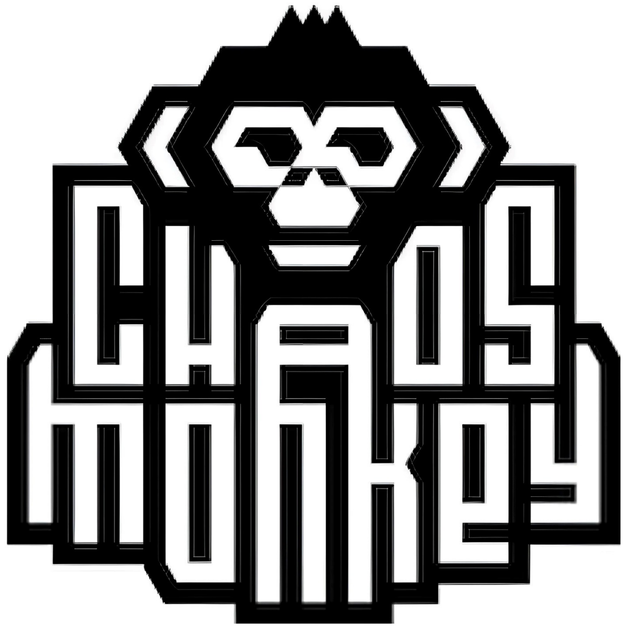
- Юнит-тестирование
- Интеграционное тестирование
- Санитайзеры
- Query Replay
- Performance тестирование

Chaos-тестирование?



Chaos-тестирование

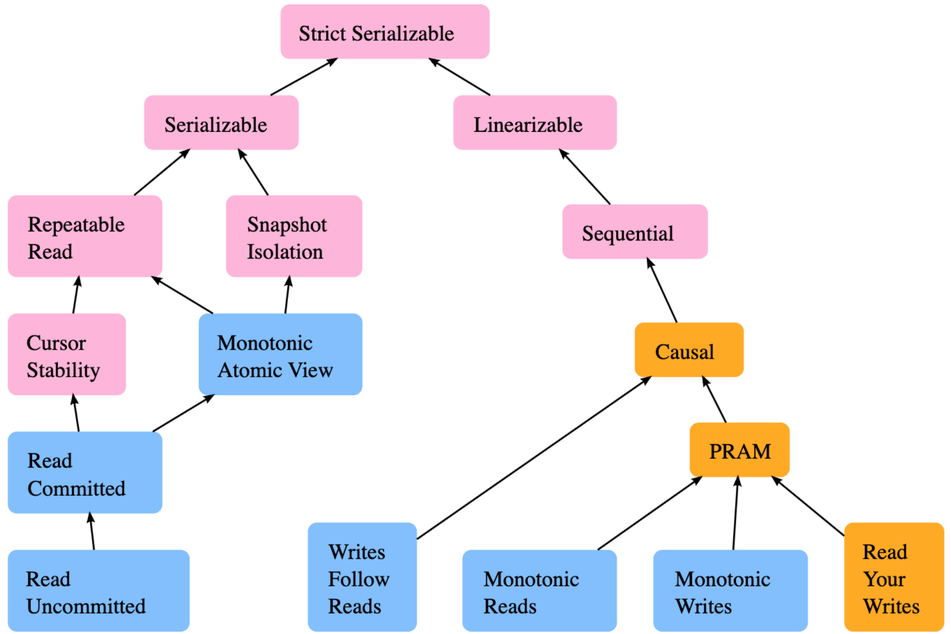
- Промышленный стандарт для тестирования распределённых систем
- Chaos Monkey от Netflix
- Chaos Mech для K8s
- Litmus Chaos для K8s



https://netflix.github.io/chaosmonkey/
https://chaos-mesh.org/docs/
https://docs.litmuschaos.io/

Chaos-тестирование: Jepsen

- Подход позволяет проанализировать свойства безопасности распределенных систем
- Позволяет проанализировать, какие аномалии распределенные системы допускают
- Использует nemesis для внедрения падений и неполадок в системе
- В процессе поддержка для YDB, смогли найти проблемы в новой версии до выкатки в PROD

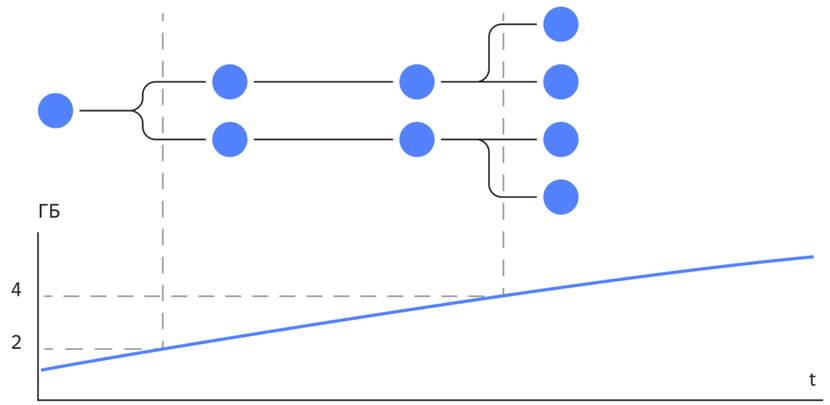


https://jepsen.io/consistency

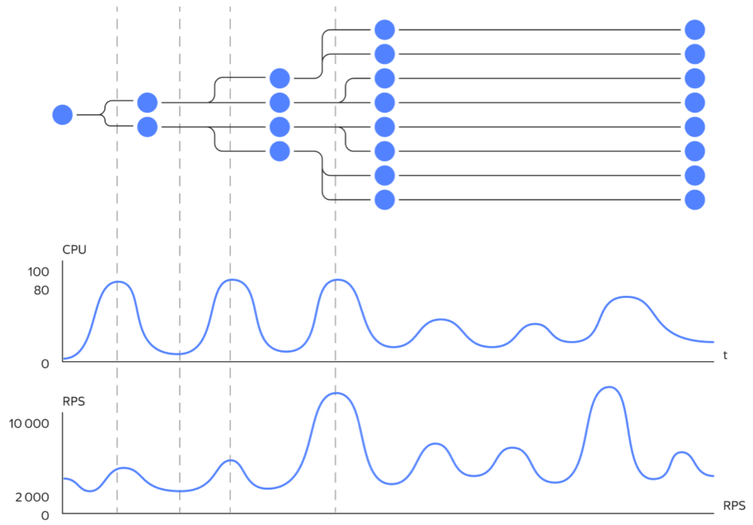
Chaos-тестирование: наш сценарий

Автоматическое партицирование

По объёму данных



По нагрузке

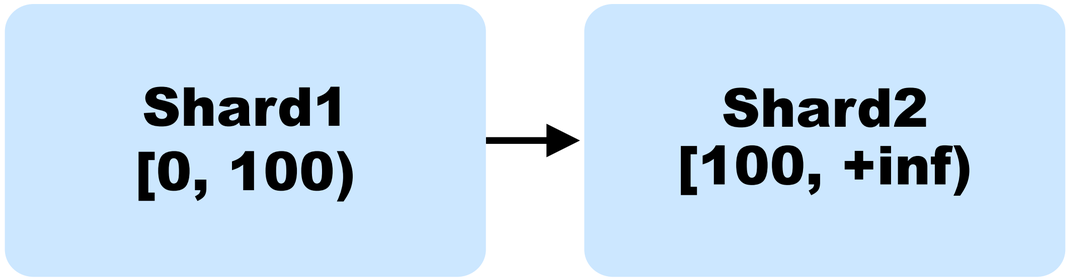


Нагрузка

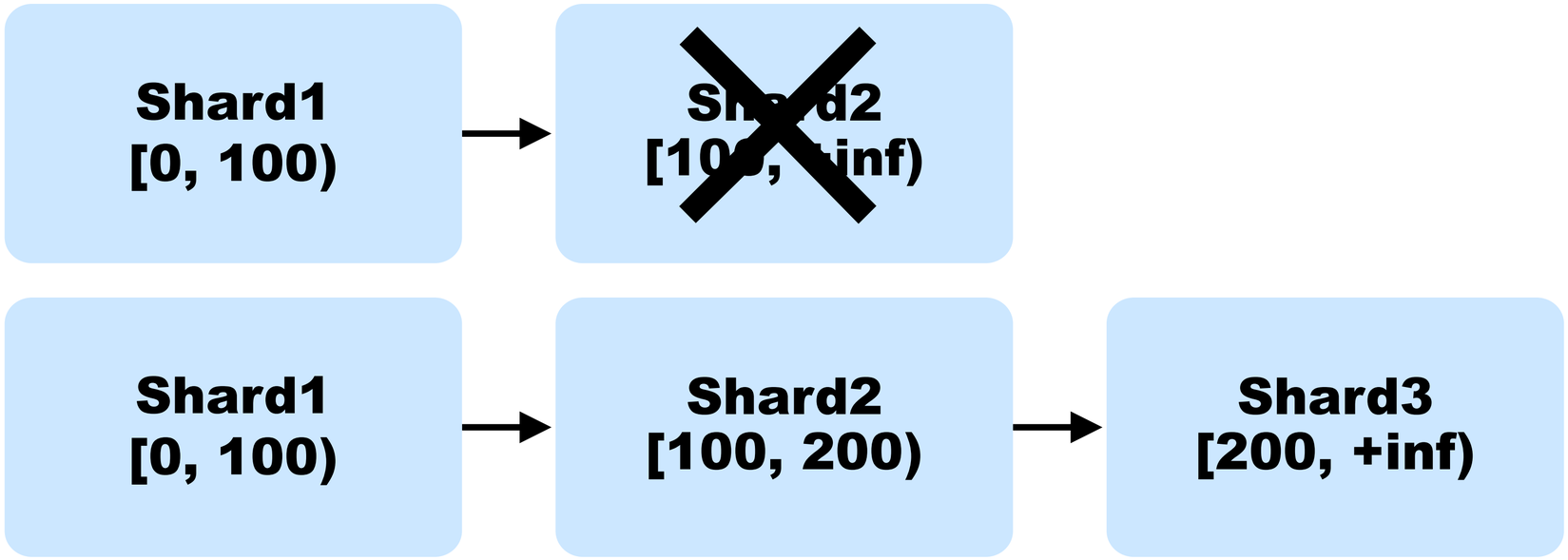
- Пишем новые значения всегда с монотонно растущим ID
- Читаем значения в текущем диапазоне ключей
- Удаляем из начала
- Добавляем операции Copy/Alter/Drop, запускаем их периодически над таблицами в случайном порядке

```
CREATE TABLE SimpleQueue (
      ID UInt64 NOT NULL,
      Value Bytes,
      PRIMARY KEY(Id)
);
```

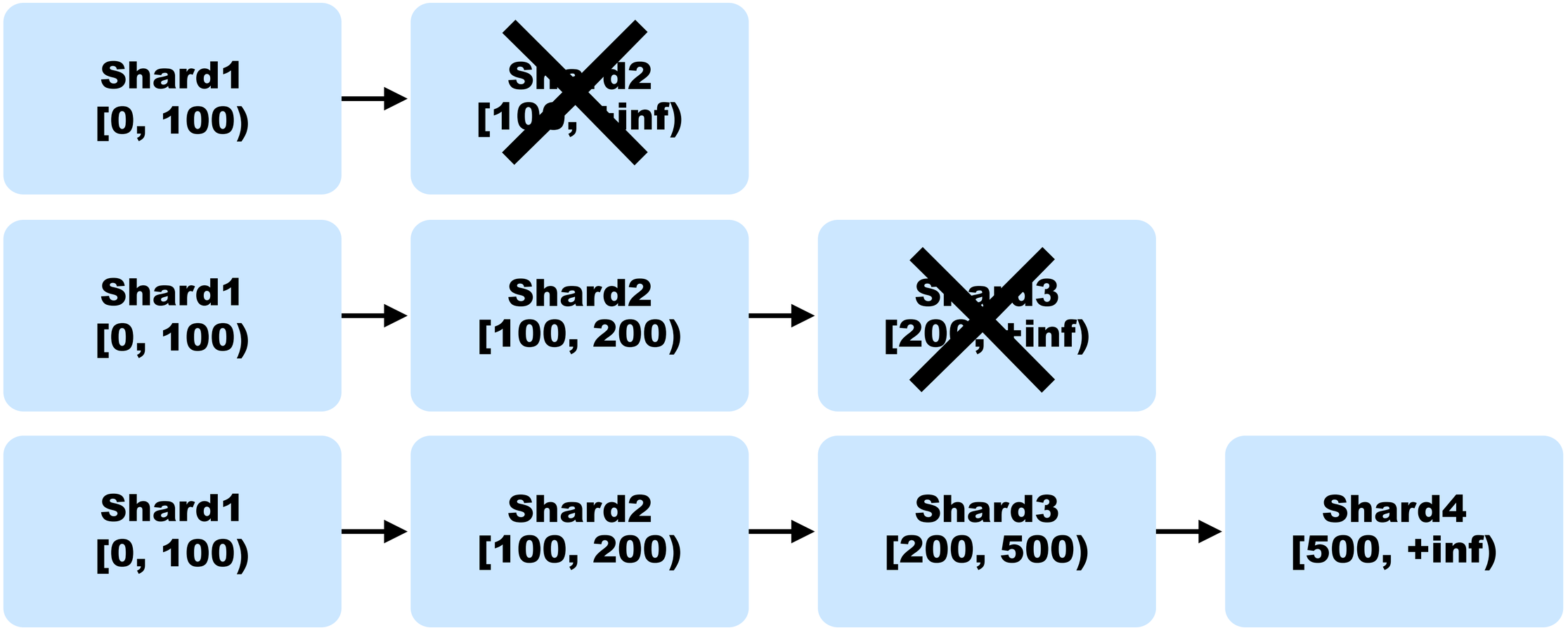
Нагрузка: split



Нагрузка: split



Нагрузка: split



Nemesis

Программа для внедрения ошибок в систему

Научились рестартить сервисы:

- kill -9
- service stop

Запускаем не на PROD

Добавили много специфичных
и полезных сценариев

- стираем диски через dd

Написали свой на Python

Nemesis

```
class Nemesis(object):
    def next_schedule(self):
        return next(self.__schedule)

    @abc.abstractmethod
    def inject_fault(self):
        pass

    @abc.abstractmethod
    def extract_fault(self):
        pass
```

Nemesis

```
class AbstractSafeEraseDataOnDisk(Nemesis, AbstractMonitoredNemesis):
    def all_disks_are_replicated(self):
        return blobstorage.cluster_has_no_unreplicated_vdisks(self._cluster)

    def extract_fault(self):
        pass

    def inject_fault(self):
        if self.all_disks_are_replicated():
            self._erase_data()
            self.logger.debug("Successful data erase")
        else:
            self.prepare_state()
            self.logger.debug("Do not erase data this time")
            return False
```

Nemesis + нагрузка

- Пишем Pytest
- Запускаем!
- Ищем баги!

```python
def test_simple_queue_workload(self):
    self._start_nemesis()

    for node_id, node in enumerate(self.kikimr_cluster.nodes.values()):
        node.ssh_command(
            'screen -d -m /Berkanavt/nemesis/bin/simple_queue --database /Root/db1',
            raise_on_error=True
        )
    sleep_time_min = 90

    logger.info('Sleeping for {} minute(s)'.format(sleep_time_min))
    time.sleep(sleep_time_min * 60)

    self._stop_nemesis()

@classmethod
def _start_nemesis(cls):
    for node in cls.kikimr_cluster.nodes.values():
        node.ssh_command("sudo service nemesis restart", raise_on_error=True)

@classmethod
def _stop_nemesis(cls):
    for node in cls.kikimr_cluster.nodes.values():
        node.ssh_command("sudo service nemesis stop", raise_on_error=False)
```

Баги: Safety

Свойство Safety говорит, что в системе не происходит ничего «плохого»

- Нет падения по segfault
- Нет abort процесса
- Нет падений по OOM Killer
- Нет ошибок INTERNAL_ERROR
- Нет критичных ошибок в логах

Баги: Safety

```
VERIFY failed: Unknown shard, shardIdx: .., TxType: TxAlterTable
kikimr/core/tx/schemeshard/schemeshard__init.cpp:1302
ReadEverything(): requirement Self->ShardInfos.contains(shardIdx) failed
```

Баги: Safety

Помогает избегать багов в PROD

- Нашли более 10 abort'ов и segfault'ов

Улучшается логирование в системе

Сценарии тестирования усложняются многократно

- Тесты на 500 строк кода C++
- >10 событий до воспроизведение бага

На проблемы пишутся регрессионные тесты

Баги: Liveness

Свойство Liveness говорит, что в итоге в системе должно происходить что-то хорошее

- Должны выполниться все транзакции
- Должны завершиться все операции split или merge
- Все таблетки запущены и работают

Для проверки потребуется внезапная остановка nemesis, так как баги могут маскироваться

Баги: Liveness

```
@property
def list_of_liveness_violations(self):
    total_tx_complete_lag = 0
    for node in self._cluster.nodes.values():
        sensors = node.monitor.get_by_name('SUM(DataShard/TxCompleteLag)')
        total_tx_complete_lag += sum(map(lambda x: x[1], sensors))

    if total_tx_complete_lag == 0:
        return []

    return [
        "Liveness violation for sensor SUM(DataShard/TxCompleteLag): "
        "actual value is %d" % total_tx_complete_lag,
    ]
```

Баги: Liveness

1. Нашли 5 багов, примеры:
   - транзакции зависали и не могли завершиться
   - split и merge не могли завершиться из-за deadlock
2. Расширили существующие метрики в Prometheus для liveness условий
3. Сложные сценарии, на которые пишут регрессионные тесты

Итоги

- Юнит-тестирование
- Интеграционное тестирование
- Санитайзеры
- Query Replay
- Performance тестирование
- Jepsen
- Тесты на Safety свойства
- Тесты на Liveness свойства

YDB

Спасибо!

Виталий Гриднев, YDB

Ведущий разработчик, руководитель команды выполнения запросов

clck.ru/39swHt