ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет Санкт-Петербургская школа
физико-математических и компьютерных наук
Департамент информатики

Габитов Даниил Алексеевич

РАЗРАБОТКА СИСТЕМЫ АНТИФРОДА ЯНДЕКС.МЕТРИКИ

Выпускная квалификационная работа - БАКАЛАВРСКАЯ РАБОТА
по направлению подготовки *01.03.02 Прикладная математика и информатика*
образовательная программа «Прикладная математика и информатика»

Научный руководитель:
доктор физико-математических наук,
профессор, департамент информатики,
Новиков Б. А.

Рецензент:
ООО «Яндекс.Технологии»,
разработчик,
Пересторонин П. Г.

Санкт-Петербург
2023

Оглавление

Аннотация

Яндекс.Метрика — крупнейшая система веб-аналитики в России и вторая по величине система веб-аналитики в мире. Нагрузка, с которой должна справляться система, постоянно растет, и со временем многие компоненты системы нуждаются в изменении для соответствия новым требованиям пропускной способности. Одной из таких компонент является веб-сервер, отвечающий за фильтрацию фродовых событий. Сервер поддерживает свое состояние в оперативной памяти, что приводит к сложности масштабирования системы.

В данной работе реализуется новое решение с использованием базы данных YDB в качестве места хранилища состояния веб-сервера, а также приводится описание основных архитектурных решений, принятых для эффективного использования вычислительных ресурсов.

Ключевые слова: высоконагруженные системы, базы данных, YDB, фильтр Блума, кэширование

Yandex.Metrika is the largest web analytics system in Russia and the second largest web analytics system in the world. The system load is continuously growing. Over time many system components need to be changed to meet new bandwidth requirements. One of these components is a web server responsible for filtering fraud events. The server maintains its state in RAM, which leads to scaling limitations. This paper explores the possibility of using the YDB database as a state storage, and describes main architectural decisions made for the efficient use of computing resources.

Keywords: highload, databases, Bloom filter, caching.

Введение

Яндекс.Метрика [4] — крупнейшая система веб-аналитики в России и вторая по величине система веб-аналитики в мире. Ее задача — обработка данных о событиях, происходящий в интернете, и формирование структур данных, позволяющих владельцам сайтов выполнять сложную аналитику в реальном времени.

Веб-сайты могут посещать не только люди, но и программные средства, направленные на искажение аналитики веб-сайта. Для создания надежных отчетов Яндекс.Метрика должна уметь отличать реального клиента от программы. Для этой цели в Яндекс.Метрике реализована собственная система антифрода.

Для каждого события Метрика собирает дополнительную информацию о пользователе. В том числе 12 идентификаторов (UserID, IP, WindowResolution...), которые мы будем называть “ключами”. Для каждого значения ключа хранятся и обновляются “статистики” (время первого и последнего события, кол-во событий...). Чтобы определить является ли событие фродовым, статистики его ключей проходят через CatBoost [1] модель.

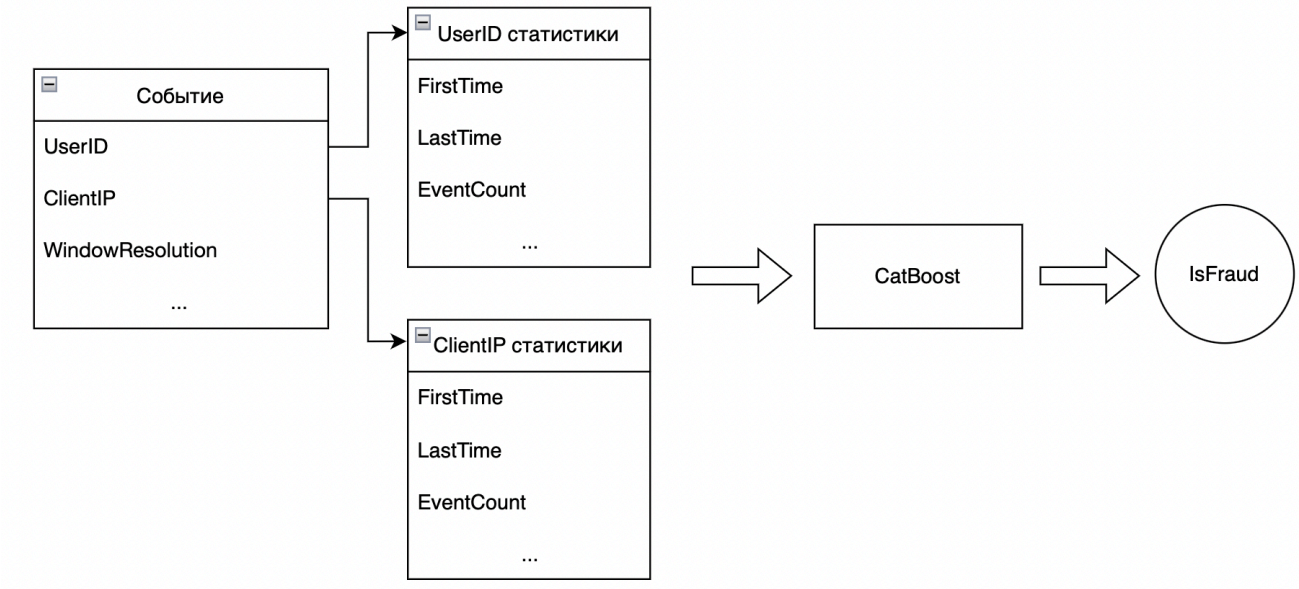

Рис. 1: Модель фильтрации фродовых событий

Задачей хранения и обновления статистик занимается антифрод сервер. Это веб-сервер, который получает в запросе события, обновляет для них статистики, а затем возвращает их в ответе. Свое состояние (посчитанные статистики для всех ключей) антифрод сервер поддерживает в оперативной памяти.

Однако, у данного решения есть проблема. Выделенной оперативной памяти серверу бывает недостаточно и тот периодически крашится, теряя все статистики. При этом сервер поддерживает только вертикальное масштабирование (добавление большего количества оперативной памяти). Очевидно, у такого масштабирования есть разумный предел.

Лимит размера хранимых статистик на сервере ощутимо ограничивает аналитиков Яндекс.Метрики в выборе методологии фильтрации фродового трафика.

К тому же данное решение не обладает “эластичностью”. Метрика представляет собой конвейер по обработке данных, состоящий из множества различных сервисов. Любой сбой в такой системе приводит к серьезным отставаниям всего конвейера. После устранения причин сбоя, нагрузка на систему увеличивается в разы, ведь ей необходимо обрабатывать не только обычную для себя нагрузку, но и те данные, которые копились за время простоя. При длительных сбоях (например, поломке дата-центра) отставания конвейера могут достигать нескольких часов. Система должна быть способна преодолевать такие отставания. Зачастую сервисы нуждаются в кратковременном увеличении пропускной способности для скорейшего устранения последствий поломки конвейера. Кратковременное добавление вычислительных нод для увеличения пропускной способности и есть та самая “эластичность”.

Оглядываясь на проблемы существующего решения, появилась идея хранить статистики не в оперативной памяти, а в базе данных. В качестве базы данных была выбрана YDB [5]. YDB — это горизонтально масштабируемая отказоустойчивая СУБД, разработанная в Яндексе и

активно использующаяся в проектах Яндекс.Метрики.

Цели и задачи

Целью данной работы является реализация антифрод сервера с использованием базы данных YDB в качестве хранилища состояния веб-сервера.

Задачи:

- Реализовать антифрод сервер с использованием базы данных YDB.
- Оптимизировать потребления вычислительных ресурсов полученного решения.
- Сравнить новое решения с оригинальным сервером.

Достигнутые результаты

В рамках данной работы был реализован новый антифрод сервер, использующий YDB в качестве хранилища состояния веб-сервера. Это позволило переложить задачи горизонтального масштабирования и отказоустойчивости на базу данных.

Было оптимизировано потребления вычислительных ресурсов полученного решения при помощи следующих методик:

- Кэширование позволило уменьшить количество SELECT запросов в базу данных с 250 тысяч запросов в секунду до 5 тысяч.
- Фильтр Блума сократил наполовину количество хранимых статистик в оперативной памяти, а также позволил уменьшить потребление памяти YDB в 2.5 раза.

Были выявлены преимущества и недостатки полученного решения: удалось снизить обьем используемой оперативной памяти с 200 до 40 гигабайт. Однако, среднее время ответа сервера увеличилось с 128 милисекунд до 1 секунды (без потери в пропускной способности).

Новый антифрод сервер был успешно интегрирован в инфраструктуру Яндекс.Метрики.

Структура работы

В главе 1 описывается разработка архитектуры нового антифрод сервера.

В главе 2 описаны оптимизации потребления вычислительных ресурсов.

В главе 3 полученное решение сранивается с оригинальным антифрод сервером.

В заключении представлено краткое описание достигнутых результатов по реализации нового антифрод сервера.

1. Архитектура нового антифрод сервера

1.1. Нагрузка

Метрика – это огромная система, состоящая из порядка 400 сервисов, отвечающий за процессинг данных пользователей. Logprocessd – сервис, ответственный за ”обогащение” веб-событий дополнительными данными. В том числе и выставление флага о фродовости события.

Чтобы определить, является ли событие фродовым logprocessd необходимо передать статистики его ключей в CatBoost модель. За этими статистиками logprocessd обращается по HTTP к антифод серверу.

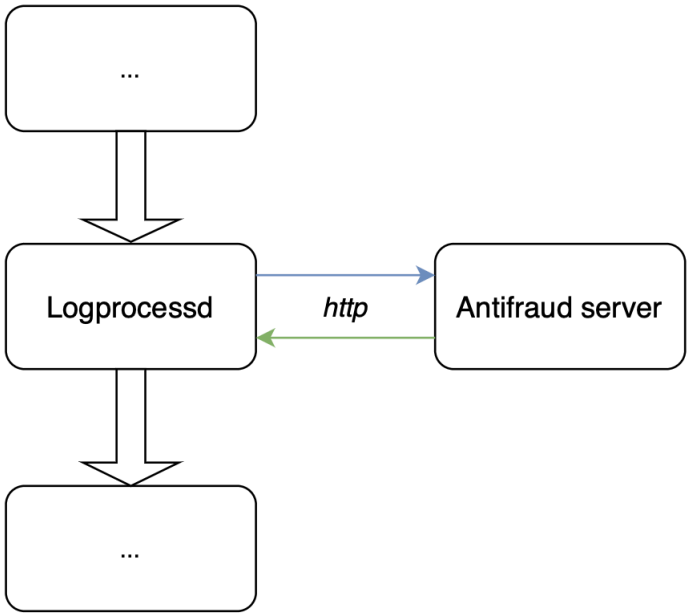

Рис. 2: Logprocessd и антифрод сервер. Клиент-серверная модель.

Сервис logprocessd шардирован по UserID на 85 шардов. Каждый шард обрабатывает данные батчами по пять тысяч событий и одновременно может обрабатывать до 72-х таких батчей. Соответственно, для каждого батча logprocessd совершает один запрос в антифрод сервер.

Антифрод сервер должен быть способен выдерживать нагрузку 1.5x от обычной нагрзуки на Яндекс.Метрику. То есть, порядка 1 миллиона событий в секунду.

1.2. Оригинальное решение

Оригинальный антифрод сервер был написан в 2011 году и с тех пор не придерживался координальных изменений.

Сервер хранит статистики в оперативной памяти и использует хеш-таблицу с отображением: ключ → статистика.

В качесте имплементации хеш-таблицы используется разработанная компанией Google Dense-Hash-Map. Dense-Hash-Map оптимизорована на быстрые look-up операции, однако она проигрывает другим реализациям в скорости обхода контейнера и расходе оперативной памяти.

Сервер шардирует ключи на 32 партиции, с хеш-таблицей под кажду партицию. Это позволяет увеличить степень паралелизма системы.

Обработка одного запроса выглядит следующим образом:

- Берется глобальная блокировка на состояние веб-сервера.
- Запрос разбивается на 32 части. В качестве ключа шардирования используется модуль от ключей.
- Партиции обновляются в 32 потока.
- Снимается блокировка
- Результирующие статистики возвращаются в ответе.

Несмотря на простоту, данное решение успешно выдерживает нагрузку Яндекс.Метрики.

Однако, внимательный читатель может задаться вопросом: разве оперативная память это надежное хранилище данных? Действительно, данные из оперативной памяти теряются при завершении процесса. Поэтому, сервер периодически складывает свое состояние на жесктий диск и восстанавливает его при перезапуске.

В целях отказоустойчивости используется модель репликации Master-Slave [3] из трех нод севера, расположеных в разных датацентрах.

У описанного решения есть проблемы:

- **Нехватка памяти**. Нагрузка, с которой должна справляться Метрика, постоянно растет. Если в 2011 году 200 гигабайт оперативной памяти хватало для хранения всех необходимых статистик, то в 2023 году этого обьема памяти стало недостаточно и сервер сталкивается с Out Of Memory падениями.

- **Масштабируемость**. Данное решение поддерживает только вертикальное масштабирование. То есть добавить больше оперативной памяти. Естественно, у такого подхода существует разумный предел.

1.3. YDB

В качестве нового хранилища статистик было решено использовать базу данных YDB.

YDB — это распределённая отказоустойчивая Distributed SQL база данных, которая сочетает в себе высокую доступность и масштабируемость со строгой согласованностью и транзакциями ACID.

Масштабируемость базы данных достигается засчет разделения таблицы на множество шардов. Один шард таблицы отвечает за свой диапазон первичных ключей. Диапазоны ключей, обслуживаемых разными шардами, не пересекаются.

При этом YDB поддерживает автоматическое партиционирование данных по нагрузке. Если один из шардов таблицы перегружен по CPU, то он встает в очередь на разделение.

Также YDB поддерживает транзакционность на уровне различных шардов. Однако, такой вид запросов обладает большей задержкой. Так, на рис. 3 для каждого класса транзакций представлена его стомость обработки.

Класс транзакции	Стоимость (Задержка)
Read Only, 1 shard	1 RTT
Write Only, 1 shard	2 RTT + 1 disk IO
Read only/Write Only, muli-shard	6 RTT + 3 disk IO
Read-Write, multi-shard	8 RTT + 4 disk IO

Рис. 3: Задержка для каждого класса транзакции YDB

При работе с YDB важно определить требуемый уровень консистентности прикладного программного обеспечения для выбора необходимого класса транзакций и избежания бесцельных накладных расходов. Так, в реализации нового антифрод сервера было решено пожертвовать консистентностью во избежание распределенных многошардовых транзакций.

Также YDB позволяет отказаться от транзакционности на запись в рамках одного шарда. Для этого необходимо использовать BulkUpsert [6] вставку.

В Метрике реализована собственная библиотека по работе с C++ SDK YDB.

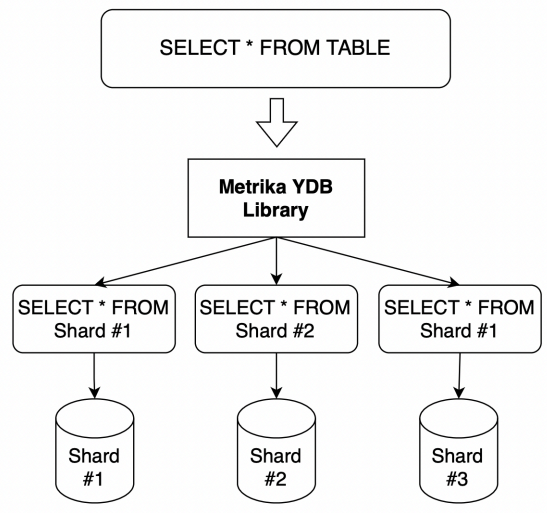

Рис. 4: Библиотека Метрики для работы с C++ SDK YDB

Данная библиотека разбивает распределенный запрос в YDB на множество одношардовых запросов.

Библиотека предоставлят широкий простор в конфигурации исполнения запроса: она позволяет ограничить размер и число одновременно исполняющихся подзапросов.

Так, YDB лучше всего справляется с большим количеством маленьких запросов. Однако, разбиение запроса на однострочные подзапросы требует большого количества CPU ресурсов.

Конфигурирование этих параметров позволяет достичь баланса между потреблением ресурсов на стороне клиента базы данных и скоростью обработки всего запроса.

1.4. Оригинальное решение и YDB

Начнем с наивной реализации нового антифрод сервера с использованием базы данных YDB. Для этого воспользуемся подходом обработки запросов оригинального решения, добавив чтение и запись в базу данных. Для каждого запроса:

- Глобальная блокировка.
- Чтение статистик из YDB.
- Применение бизнес логики (обновление статистик).
- Запись данных в YDB.
- Возвращаем ответ.

Однако, реализация данного решения показала свою несостоятельность из-за низкой пропускной способности системы.

Действительно, заменив чтение и запись из RAM на чтение и запись из базы данных, мы получили гораздо большую задержу.

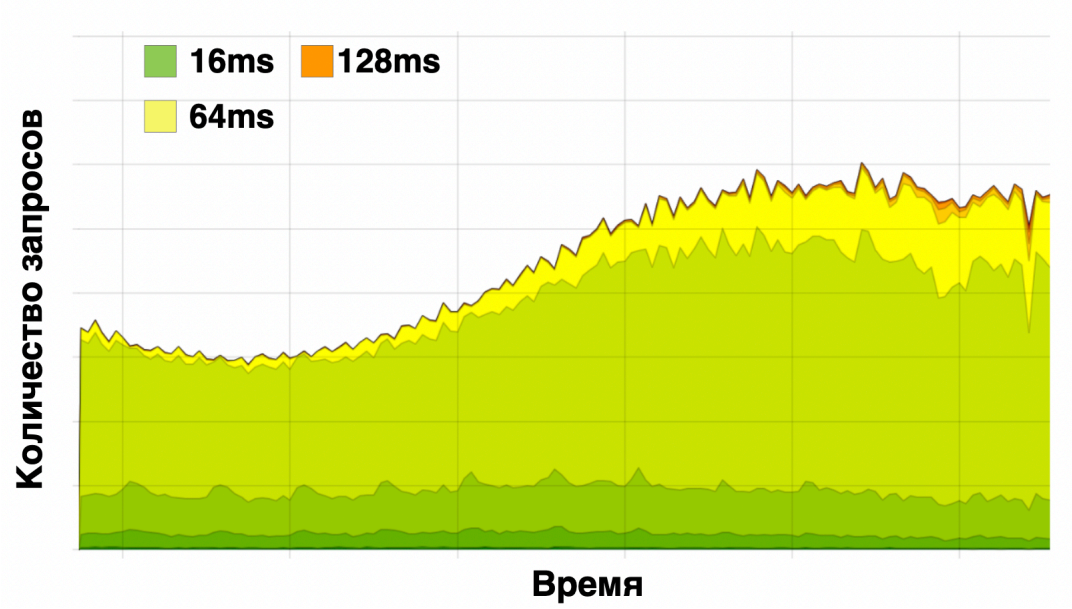

Рис. 5: Время чтения статистик из базы данных при наивной реализации

Чтение статистик из базы данных для одного запроса может занимать до 128 миллисекунд. При этом сервер способен обрабатывать лишь один запрос одномоментно (глобальный мьютекс).

Как результат, такая реализация антифрод сервера способна обрабатывать лишь 80 тысяч событий в секунду. Этой пропускной способности недостаточно для нагрузки Яндекс.Метрики (1 миллион событий в секунду).

Появляется вопрос: в чем необходимость глобальной блокировки? Причиной этому – само предназначение антифрод сервера. Сервер агрегирует статистики ключей из разных шардов logprocessd (клиента). То есть в разных запросах к антифрод серверу могут быть события с пересекающимися ключами. Паралельная обработка запросов может привести к гонке данных: одновременному обновлению статистик одного и того же ключа.

Во избежание гонки данных оригинальный антифрод сервер обрабатывает запросы последовательно.

Необходимо разработать новое решение.

1.5. Батчинг обработки запросов

Описанное в прошлой главе решение неоптимально задействовало ресурсы YDB, обрабатывая последовательно по одному запросу (5000 событий).

Да, запросы в базу данных требуют большего времени, чем чтение из оперативной памяти. Причиной этому транспортировка данных по сети и чтения с жесткого диска. Однако, эту задержку можно компенсировать повышением нагрузки. Последовательная обработка запросов по одному этой нагрузки на базу данных не предоставляло.

Попробуем обрабатывать запросы пачками:

- Чтение статистик для N новых запросов.
- Последовательное обновление статистик.
- Запись статистик в базу данных.

Это позволит увеличить размер запроса в YDB и увеличить нагрузку на шарды базы данных.

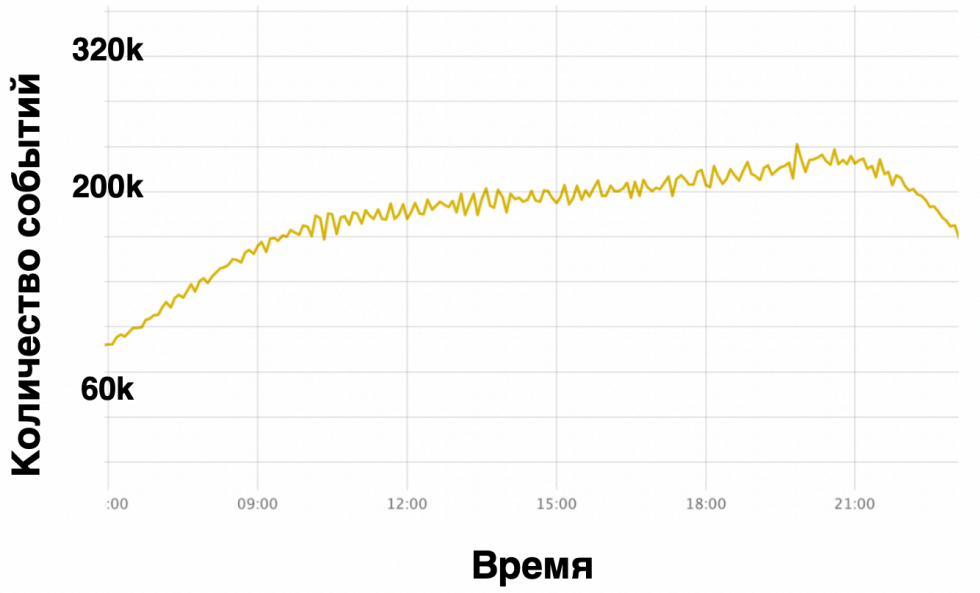

Рис. 6: Количество обрабатываемых событий в секунду с батчингом запросов

Новый подход к обработке запросов позволил увеличить пропускную способность сервера с 80 тысяч событий в секунду до 220 тысяч.

1.6. Параллелизация и обработчики

Однако, 220 тысяч событий в секунду все еще недостаточно. Следующим шагом для увеличения пропускной способности сервера стала параллелизация вычислений. А именно, выделение сущности "Обработчик".

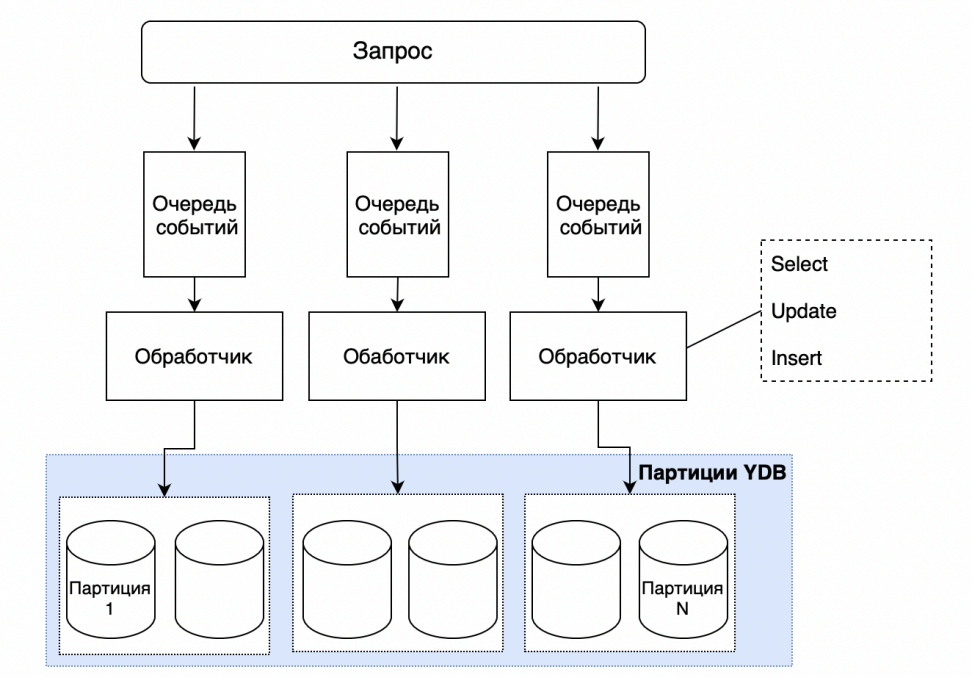

Рис. 7: Архитектура сервера с "Обработчиками"

Обработчик – сущность, ответственная за последовательное обновление ключей конкретного типа (напомню, что всего существует 12 ключей и для каждого ключа подсчитывается свой набор статистик). Одна итерация обработчика выглядит следующим образом:

- Чтение статистик из базы данных
- Применение бизнес логики (обновление статистик)
- Запись статистик в базу данных.

События для очередной итерации обработчик берет из своей очереди. Модель обработки запросов сервером описывается следующими шагами:

- **Распределение событий по очередям обработчиков**. Для определения очереди используется модуль ключа от количества очередей. Таким образом, гонка данных не возникает: ключ всегда обрабатывается одним и тем же обработчиком.
- **Ожидание результатов**. При добавлении событий в очередь обработчик возвращает Future (результат асинхронной операции). Запрос дожидается выполнения каждой полученной Future.
- **Извлечение ответа**. В качестве результата асинхроной операции обработчик возвращает статистики ключей. Запрос выставляет для каждого события из запроса результирующие значения ключей.

Такой подход к обработке запросов:

- Увеличивает скорость обновления статистик засчет парализации.
- Лучше утилизирует ресурсы YDB. YDB не простаивает, пока сервер применяет бизнес логику на своей стороне.

В результате удалось достичь пропускной способности сервера в 1 миллион событий в секунду.

Рис. 8: Количество обратываемых событий в секунду с "Обработчиками"

1.7. Результаты

В данной главе описаны основные архитектуры решения принятые при проектировании нового веб-сервера. Они позволили увеличить пропускную способность сервера с 80 тысяч до 1 миллиона событий в секунду. Среди них:

- Отказ от распределенных транзакций.
- Батчинг запросов.
- Параллелизация вычислений при помощи абстракции "Обработчик".

2. Оптимизация потребления вычислительных ресурсов

2.1. Высокое потребление CPU YDB

Однако, тестирование нового антифрод сервера на реальной продовой нагрузке выявило его существенный недостаток. Этот недостаток – высокое потребление CPU на стороне YDB.

Если оригинальному решению было достаточно 96 ядер, то прототип использует 370 (из них 50 на веб сервер и 320 ядер на стороне YDB).

Рис. 9: Потребление CPU YDB

2.2. Уменьшение нагрузки на YDB

Для уменьшения нагрузки нагрузки на базу данных было имлементировано две оптимизации:

- Кэширование ”свежих” статистик.
- Асинхронное обновление статистик.

Кэширование статистик на стороне веб-сервера. Сервер стал хранить в оперативной памяти те статистики, который обновлялись в

последний час. Данная оптимизация уменьшила количество SELECT запросов с 250 тысяч в секунду до 5 тысяч.

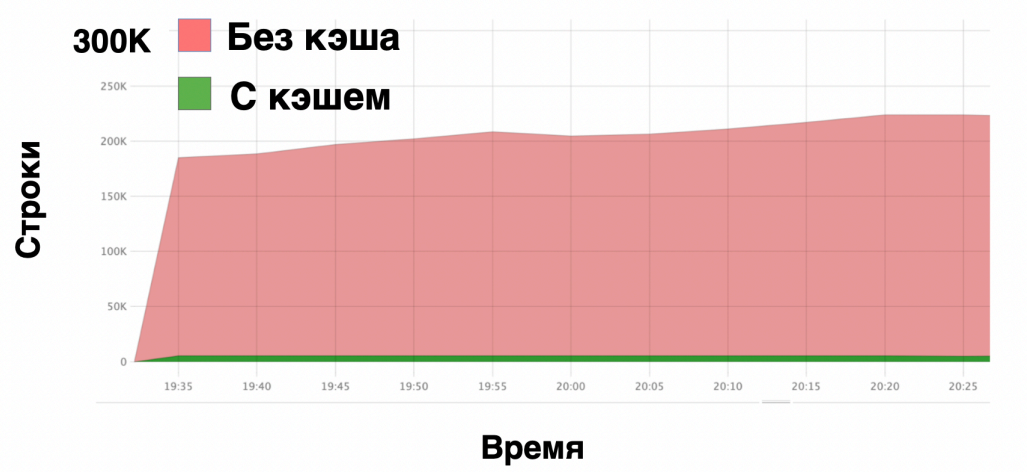

Рис. 10: Сравнение количества прочитанных строк из YDB с кэшированием и без

Однако, кэширование требует большого количества выделенной оперативной памяти на стороне веб-сервера. Так, для хранения в кэше 200 миллионов статистик требуется порядка 90 гигабайт оперативной памяти.

На рис. 11 представлен график отображающий количество статистик, хранящихся в кэше антифрод сервера за 2 дня.

Рис. 11: Количество статистик в кэше

Асинхронное обновлнение статистик. Обновления в YDB отправляются не сразу, а группируются в большой батч и асинхроно сбрасываются в базу данных. Один BulkUpsert требует гораздо меньших ресурсов CPU, в сравнений с множеством небольщих запросов.

Однако, асинхронность приводит к ”потерянному обновлению”. В

случае падения сервера обновление статистик может не успеть отправиться в YDB и обновленные данные будут потеряны.

Потеря данных в случае отказа сервера не является критичной в контексте разработки антифрод системы. Действительно, задача антифрода – успешно идентифицировать спам трафик. Утрата ограниченного количества событий не должна сказаться на выявлении трафика вцелом.

Результат оптимизаций: удалось уменьшить потребление CPU на строне YDB со 100 до 30 процентов.

2.3. Фильтр Блума

Целью данного проекта является распознавание фродового трафика. При этом для CatBoost модели интересны статистики не для всех ключей, а только для тех, которые встречались достаточно часто.

Значит, антифрод сервер может не хранить статистики для редко встречающихся событий. Для реализаций этого подхода нам понадобится структура данных "Фильтр Блума".

Определение. Фильтр Блума [2], – это вероятностная структура данных, позволяющая проверять принадлежность элемента к множеству за $\mathcal{O}(1)$.

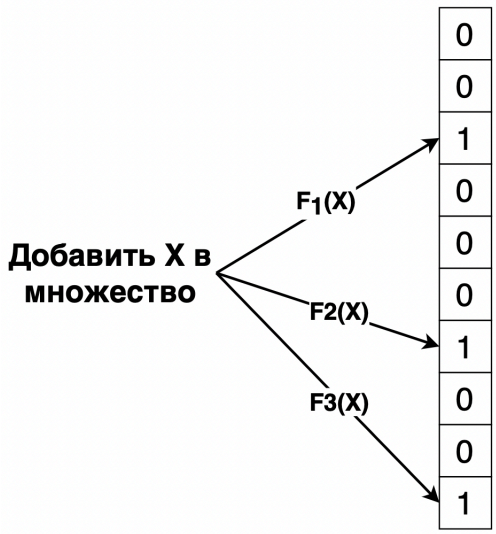

Рис. 12: Добавление элемента в фильтр Блума

Фильтр Блума представляет собой массив из M счетчиков и набора хеш функций. Хеш функции отображают элемент множества в соответствующие ему счетчики и при добавлении элемента эти счетчики увеличиваются.

Фильтр Блума позволяет вероятностно оценить количество вхождений элемента в множества. При этом возможны ложноположительные срабатывания, но не ложноотрицательные.

Идея: храним и обновляем только те события, которые встретились хотя бы 16 раз. Проверять это условие будем при помощи фильтра Блума.

При этом, во избежание колизий число счетчиков должно быть достаточно большим. Опытным путем было выявлено, что оптимальное число счетчиков – 10 миллиардов.

В целях отказоустойчивости, состояние Блум фильтра также асинхронно записывается в базу данных. При перезапуске сервер вычитывает счетчики из базы данных.

Фильтр необходимо периодически очищать. В противном случае счетчики фильтра переполнятся и фильтр перестанет выполнять свою функцию.

Для этой цели антифрод сервер использует не один фильтр, а два. Первый фильтр считается активным и фильтрация событий происходит при его помощи, а второй поддерживается на замену первому. Один жизненый цикл фильтров выглядит следующим образом:

- Шаг 0 (начало работы): оба фильтра пустые.
- Шаг 1: Первый фильтр обрабатывает запросы.
- Шаг 2 (спустя X дней): элементы добавляемые в первый фильтр, начинают добавляться и во второй.
- Шаг 3 (спустя 2X дней): первый фильтр удаляется и его место занимает второй фильтр. Второй фильтр инициализируется пустым. Возвращаемся к шагу 1.

Использование Блум фильтра позволило:

- Фильтровать 10% событий.
- Сократить количество статистик, хранимых в кэше, в 2 раза.
- Уменьшить потребление памяти YDB в 2.5 раза.

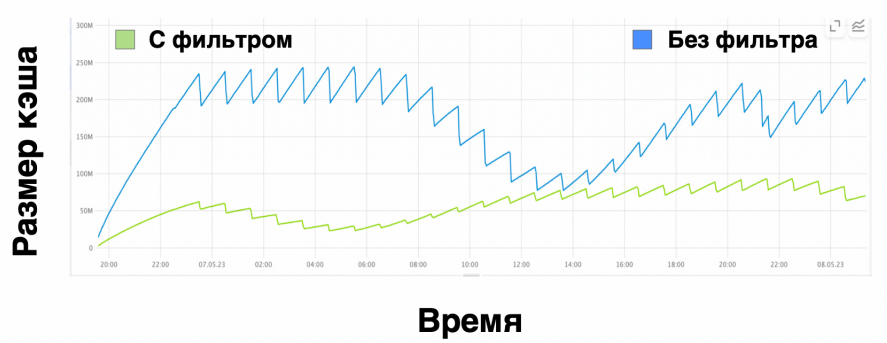

Рис. 13: Количество статистик в кэше сервера до/после применения фильтра Блума

2.4. Результаты

В данной главе описаны основные оптимизации, позволившие уменьшить расход вычислительных ресурсов антифрод сервера.

- Кэширование и батчинг запросов позволили уменьшить CPU нагрузку на базу данных со 100% до 30%.
- Фильтр Блума уменьшил обьем данных, хранимый на стороне YDB в 2.5 раза, а также сократил размер кэша в оперативной памяти веб-сервера в 2 раза.

3. Сравнение с оригинальным сервером

Подводя итог работе, хотелось бы сравнить полученное решение с оригинальным антифрод сервером.

Преимущества нового антифрод сервера:

- **Использование базы данных**. Использование YDB позволяет переложить задачи масштабирования и отказоустойчивости на базу данных.
- **Уменьшение расхода оперативной памяти**. Если оригинальный сервер расходовал по 200GB на каждый из 3х веб-серверов, то новому решению требуется лишь 40GB оперативной памяти для поддержки кэша статистик.
- **Эластичность**. Антифрод сервер способен горизонтально масштабироваться под необходимую нагрузку. Оригинальный сервер этим свойством не обладал, поскольку он также отвечал за хранение статистик.
- **Легкость администрирования**. Оригинальный антифрод сервер использовал самописную Master-Slave систему с хранением данных на жестком диске. Любое изменение логики антифрод сервера приводило к рискам повреждения данных, а также требовало нетривиальной выкатки обновлений. Хранение статистик в базе данных позволило избавиться от этих проблем.

Однако, у нового решения есть и недостаток:

- **Увеличение времени обработки запроса**. Среднее время обработки запросов увеличилось с 128 до 1024 милисекунд. Запросы в базу данных обладают большей задержкой, чем обращение к оперативной памяти. К тому же, батичнг запросов приводит к большим задержкам обработки вцелом: время обработки запроса равняется времени обработки всей пачки запросов.

Для Яндекс.Метрики не так критична скорость обработки запросов, как общая пропускная способность системы. Поэтому, увеличение времени обработки запроса новым антифрод сервером является разумным компромисом ко всем тем преимуществам, описанным выше.

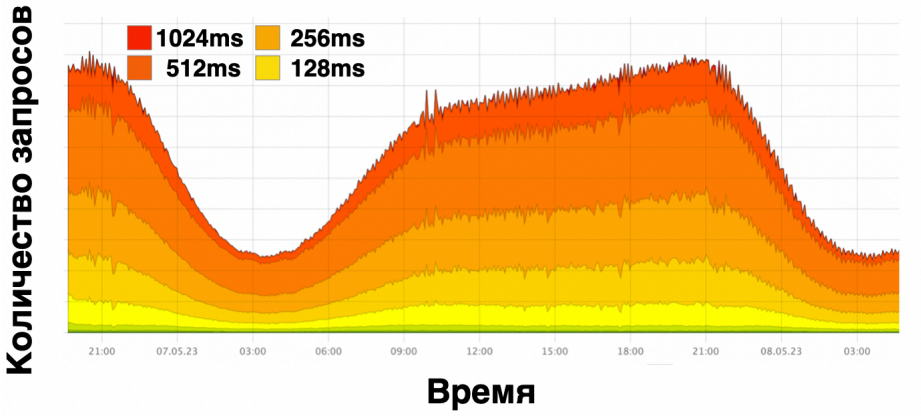

Рис. 14: Гистограмма времени обработки запросов новым антифрод сервером

Заключение

В результате данной работы был реализован новый антифрод сервер с использованием базы данных YDB. Разработанное решение способно выдерживать нагрузку в 1 миллион событий в секунду.

Использование YDB позволило переложить задачи горизонтального масштабирования и отказоустойчивости на базу данных. К тому же, новое решение требует меньшего обьема оперативной памяти и обладает способностью горизонтального масштабирования.

Однако, среднее время обработки запроса сервером увеличилось с 128 милисекунд до 1 секунды (без потери в пропускной способности).

Было оптимизировано потребления вычислительных ресурсов новой системы при помощи следующих методик:

- Кэширование позволило уменьшить количество SELECT запросов в базу данных с 250 тысяч запросов в секунду до 5 тысяч.
- Фильтр Блума сократил наполовину количество хранимых статистик в оперативной памяти, а также позволил уменьшить потребление памяти YDB в 2.5 раза.

Новый антифрод сервер успешно интегрирован в инфраструктуру Яндекс.Метрики.

Список литературы

[1] A Prokhorenkova L Gusev G Vorobev A Dorogush V Gulin. CatBoost: unbiased boosting with categorical features. — 2018. — Access mode: https://proceedings.neurips.cc/paper_files/paper/2018/file/14491b756b3a51daac41c24863285549-Paper.pdf.

[2] H. Bloom. Space/time trade-offs in hash coding with allowable errors. — 1970. — Access mode: https://dl.acm.org/doi/pdf/10.1145/362686.362692.

[3] Wikipedia. Master/Slave. — online; accessed: https://ru.wikipedia.org/wiki/Ведущий_-_ведомый.

[4] Wikipedia. Яндекс.Метрика. — online; accessed: https://ru.wikipedia.org/wiki/Яндекс.Метрика.

[5] Yandex. YDB. — online; accessed: https://cloud.yandex.ru/services/ydb.

[6] Yandex. YDB BulkUpsert. — online; accessed: https://ydb.tech/ru/docs/reference/ydb-sdk/recipes/bulk-upsert.