СБЕР

gRPC client-side load balancing: зачем, что, как и WTF?!

Дмитрий Бугайченко

Немного о себе:

2020 SberRecSys, CDS РБ

2019 Облачная платформа рекомендации

2018 Автоматические A/B тесты

2017 Рекомендации контента для витрин

2016 Рекомендации друзей

2015 Умная лента

2014 Универсальная аналитическая платформа

2013 Рекомендации Групп

2012 Рекомендации музыки

2011 Работа в Mail.ru

2009 Преподаватель СПбГУ

2008 Кандидат физ-мат наук

Почему CDS рассказывает про gRPC?

Почему CDS рассказывает про gRPC?

SLA SberRecSys: 20 000 тпс, <100 ms для 99.99% запросов

Почему CDS рассказывает про gRPC?

SLA SberRecSys: 20 000 тпс, <100 ms для 99.99% запросов

Включая:

1. Запрос от бизнес-логики к SberRecSys
2. Запрос за кандидатами для рекомендаций от SberRecSys к Feature store
3. Запрос за признаками для кандидатов от SberRecSys к Feature store
4. Запрос на скоринг от SberRecSys к сервису моделей

Почему CDS рассказывает про gRPC?

SLA SberRecSys: 20 000 тпс, <100 ms для 99.99% запросов

Включая:

1. Запрос от бизнес-логики к SberRecSys
2. Запрос за кандидатами для рекомендаций от SberRecSys к Feature store
3. Запрос за признаками для кандидатов от SberRecSys к Feature store
4. Запрос на скоринг от SberRecSys к сервису моделей

Feature store

- Быстрый доступ к признакам объектов и клиентов
  - В том числе по большому количеству ключей за раз
- Наполнение пакетными и near real time процессами
- Крупные, не облачные узлы
  - Запрос за 10000+ ключами разбивается на <=4 части
  - Меньше вероятность «черного лебедя»
- Доступен по gRPC ☺

Почему gRPC?

- Protobuf – компактно, быстро, управляемо
- HTTP2 – минимум накладных расходов, масштабируется «многие ко многим»
- TLS 1.2+ - одобрено кибербезой
- JVM + Python – интеграция с моделями
- Совместимо с service mesh Istio

Три компонента успеха

Горизонтальная масштабируемость

Высокая доступность

Низкая задержка

Три компонента успеха

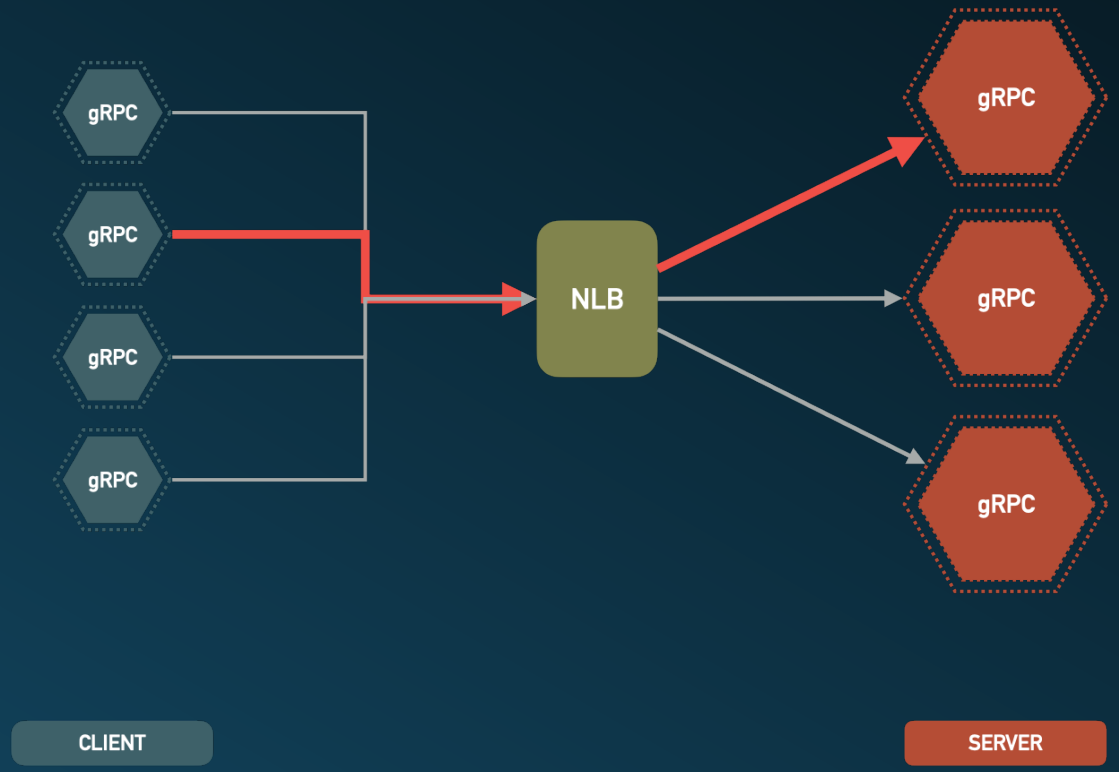
Горизонтальная масштабируемость

Высокая доступность

Низкая задержка


EXCELLENT...
LET THE LOAD BALANCING BEGIN
memegenerator.net

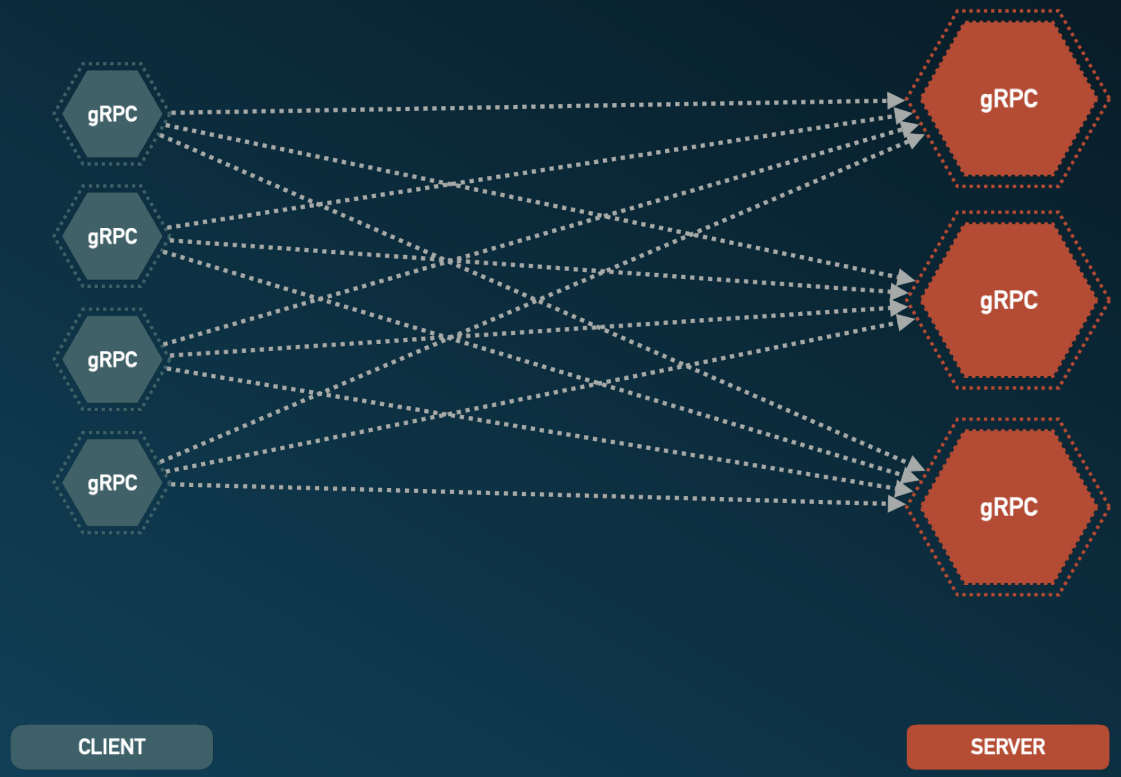
Server-Side load balancing



- ✓ Прозрачно для клиента и сервера
- ✗ Дополнительное плечо
- ✗ Потенциальная точка отказа
- ✗ Дисбаланс при наличии крупных клиентов

Источник: https://majidfn.com/blog/grpc-load-balancing

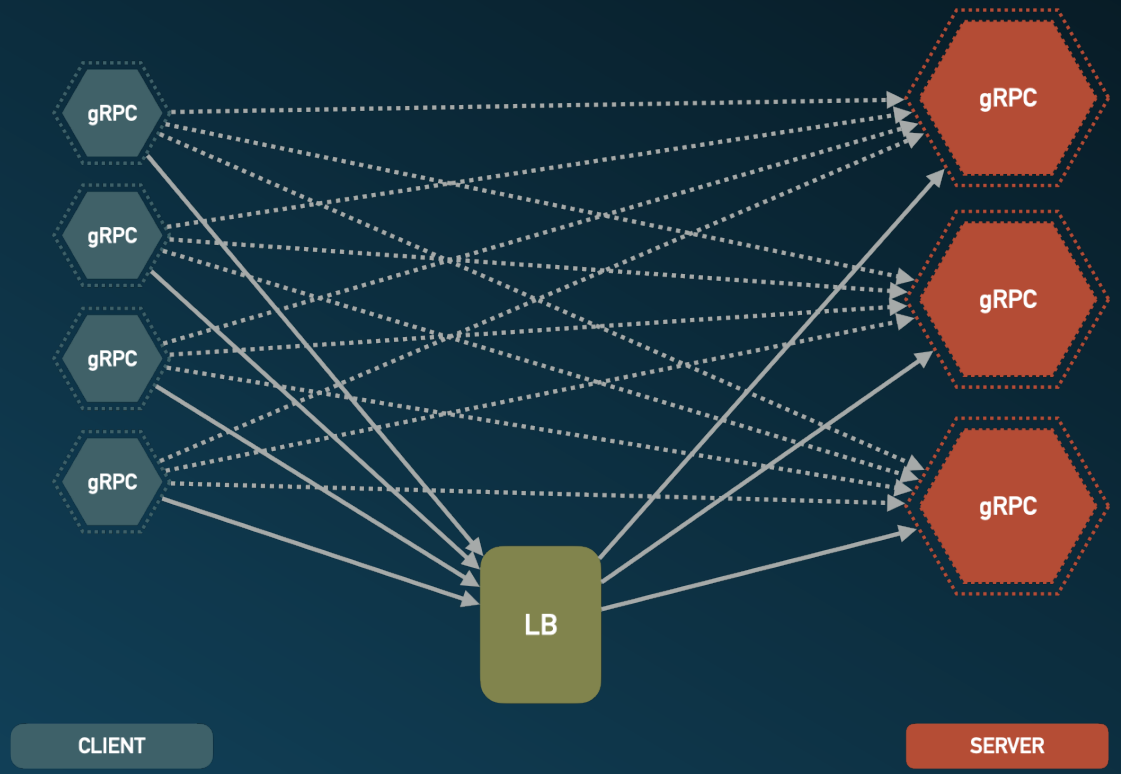
Client-Side load balancing



- ✓ Минимум накладных расходов
- ✓ Нет бутылочного горлышка
- ✗ Нужна поддержка на клиент

Источник: https://majidfn.com/blog/grpc-load-balancing

Look-Aside load balancing



- ✓ Минимум накладных расходов
- ✓ Простое дискавери
- ✗ Нужен еще один компонент в систему

Источник: https://majidfn.com/blog/grpc-load-balancing

Client-side load balancing в gRPC

Ожидания

- Указываешь список серверов в настройках клиента
- Клиент распределяет запросы между указанными серверами

Реальность

- Указывается одно имя
- По имени через DNS получаются адреса
- ✗ Применяется стратегия pick_first

Источник: https://github.com/grpc/grpc/blob/master/doc/load-balancing.md

Как настроить round_robin?

Как настроить round_robin?

В DNS txt record!

Источник: https://github.com/grpc/proposal/blob/master/A2-service-configs-in-dns.md

Идея решения

- Переопределить логику получения адресов и конфигурации по имени
- Подставить нужные значения конфигурации в результат

Смотрим в код

- NameResolverProvider
- ResolutionResult
- ServiceConfigParser
- BalancingNameResolver

Источник: https://github.com/DmitryBugaychenko/jocker-grpc-loadbalancing

Три компонента успеха

Горизонтальная масштабируемость

Высокая доступность

Низкая задержка


REQUEST IT
ONE MORE TIME
risovach.ru

Настройки повторов в gRPC

```
"retryPolicy": {
  "maxAttempts": 4,
  "initialBackoff": "0.1s",
  "maxBackoff": "1s",
  "backoffMultiplier": 2,
  "retryableStatusCodes": [
    "UNAVAILABLE"
  ]
}
```

- Количество попыток
- Задержки
- Допустимые коды
- Backoff
- Вроде все на месте

Источник: https://github.com/grpc/proposal/blob/master/A6-client-retries.md

Где прописать настройки повторов для gRPC?

Где прописать настройки повторов для gRPC?

В DNS txt record!

Источник: https://github.com/grpc/proposal/blob/master/A2-service-configs-in-dns.md

Защита от «шторма повторами»

```
"retryThrottling": {
  "maxTokens": 10,
  "tokenRatio": 0.1
}
```

- Every failed RPC will decrement the token_count by 1.
- Every successful RPC will increment the token_count by tokenRatio
- If token_count <= (maxTokens / 2), then RPCs will not be retried

Источник: https://github.com/grpc/proposal/blob/master/A6-client-retries.md

Защита от «шторма повторами»

```
"retryThrottling": {
  "maxTokens": 10,
  "tokenRatio": 0.1
}
```

Throttling активируется при ~10% ошибок

- Every failed RPC will decrement the token_count by 1.
- Every successful RPC will increment the token_count by tokenRatio
- If token_count <= (maxTokens / 2), then RPCs will not be retried

Источник: https://github.com/grpc/proposal/blob/master/A6-client-retries.md

Смотрим в код

- Статусы ошибок
- Обработка таймаутов

Источник: https://github.com/DmitryBugaychenko/jocker-grpc-loadbalancing

Три компонента успеха

Горизонтальная масштабируемость

Высокая доступность

Низкая задержка

Простая арифметика

- Ваш сервис должен отработать за 100мс
- Вы хотите иметь второй шанс

Простая арифметика

- Ваш сервис должен отработать за 100мс
- Вы хотите иметь второй шанс
- Ваш сервис должен отработать за 45мс


gRPC
клиент
Спекулятивная
попытка
Хеджированный
запрос

Настройки хэджирования в gRPC

```
"hedgingPolicy": {
  "maxAttempts": 4,
  "hedgingDelay": "0.5s",
  "nonFatalStatusCodes": [
    "UNAVAILABLE",
    "INTERNAL",
    "ABORTED"
  ]
}
```

- Количество попыток
- Задержки
- Допустимые коды

Источник: https://github.com/grpc/proposal/blob/master/A6-client-retries.md

gRPC клиент

Запрос на
нездоровый сервер

@#$%^& сервер!

Health check в gRPC

- Дополнительный сервис Health
- Два режима опроса
  - Check – разовая проверка
  - Watch – стриминг обновлений
- Сервис либо SERVING, либо NOT_SERVING
- Если все NOT_SERVING – даунтайм
- Для активации имя сервиса надо добавить в конфиг

Источник: https://github.com/grpc/grpc/blob/master/doc/health-checking.md

Смотрим в код

- Сервис Health
- HealthStatusManager
- Health checking config

Источник: https://github.com/DmitryBugaychenko/jocker-grpc-loadbalancing

Вместо послесловия

- gRPC это офигительная штука
- С поддержкой из коробки, в том числе
  - балансировки, повторов, хеджирования, проверок здоровья
- Но без возможности все это нормально настроить
- Поэтому - BalancingNameResolver

Вместо послесловия

- gRPC это офигительная штука
- С поддержкой из коробки, в том числе
  - балансировки, повторов, хеджирования, проверок здоровья
- Но без возможности все это нормально настроить
- Поэтому - BalancingNameResolver
  - Или grpc-xds

Источник: https://github.com/grpc/proposal/blob/master/A27-xds-global-load-balancing.md

СБЕР

Создавайте будущее вместе с нами!

https://sberbank-talents.ru