УДК 004.27
ГРНТИ 50.05.15
DOI 10.24412/2409-3203-2021-27-32-37

ИНТЕГРАЦИЯ МИКРОСЕРВИСОВ НА ОСНОВЕ RPC

Пацей Наталья Владимировна
к.т.н., заведующая кафедрой программной инженерии
Шитько Андрей Михайлович
аспирант
Белорусский государственный технологический университет
Республика Беларусь, г. Минск

Аннотация: Статья посвящена анализу вопроса интеграции микросервисов с помощью технологии RPC (Remote Procedure Call – удаленный вызов процедур). Существует большое количество технологий для определения способа общения микросервисов, поэтому здесь было представлено два свойства, которыми должна обладать выбранная технология. Выделены несколько преимуществ RPC технологии в сравнении с REST, где критериями оценки выступали скорость и производительность. Для интеграции микросервисов использовался gRPC – высокопроизводительный RPC фреймворк, который работает поверх HTTP/2 протокола и используют бинарный формат Protocol Buffers для сериализации и десериализации данных. Представлена базовая схема интеграции микросервисов, дающая представление о том, как происходит взаимодействие. При построении распределенной системы на основе микросервисов приходится решать достаточное количество проблем, которые возникают при интеграции микросервисов. Выявлены и проанализированы такие проблемы, как наличие сбоев, сетевых задержек, бесполезная нагрузка на сервисы из-за долгих запросов, контроль запросов, горизонтальное масштабирование. Представлены решения устранения представленных проблем с помощью технологии RPC.

Ключевые слова: микросервисы, микросервисная архитектура, интеграция, RPC.

MICROSERVICES INTEGRATION BASED ON RPC

Patsei Natallia Vladimirovna
PhD, Associate Professor of software engineering
Shytska Andrei Mikhaylovich
graduate student
Belarusian State Technological University
Republic of Belarus, Minsk

Abstract: The article is devoted to the analysis of the issue of joint integration of microservices using RPC (Remote Procedure Call) technology. There are a lot of technologies for determining the way to communicate microservices, so here were presented two properties that the chosen technology should possess. There are several advantages of RPC technology in comparison with REST, where the speed and performance were the criteria for evaluation. To integrate microservices, gRPC was used, a high-performance RPC framework that runs over an HTTP / 2 protocol and uses the binary Protocol Buffers format to serialize and deserialize data. A basic scheme for integrating microservices is also presented, giving an idea of how the services interact. When building a distributed system based on microservices, it is necessary to solve a sufficient number of problems that arise when integrating microservices. Identified and analyzed such problems as the presence of failures, network delays, useless load on services due to long requests, query control, horizontal scaling. Solutions for solving the presented problems

using RPC technology are presented.

Keywords: microservices, microservice architecture, integration, RPC.

Корректная интеграция является наиболее важным этапом проектирования микросервисов. При правильном проетировании микросервисы сохранят свою автономность, в то же время можно будет вносить в них изменения и выпускать их новые версии независимо от всей остальной системы. При некорректном исполнении система на основе микросервисов будет функционировать со сбоями и низкой производительностью, что недопустимо при построении микросервисной архитектуры.

Для определения способа взаимодействия одного микросервиса с другим имеется следующий выбор: SOAP, REST, JSON-RPC, Protocol Buffers. Поэтому, чтобы решить, какую технологию использовать, важно, что нужно получить.

Можно выделить два существенных свойства, которыми должна обладать выбранная технология.

1. *Стойкость к изменениям*. Как правило, правки неизбежны. Задача заключается в том, чтобы те правки, которые вносятся в бизнес-логике микросервиса (например, добавление новых полей), не должны касаться уже имеющихся клиентов.
2. *Сохранение технологической связанности*. Важным является сохранение технологической независимости API, используемых для обмена данными между микросервисами. Нужно избегать применения технологий, которые навязывают, какие технологические наборы следует применять при реализации микросервисов.

Известно, что REST технология может решить данные проблемы, однако технология RPC бладает некоторыми преимуществами, по сравнению с REST.

Protocol Buffers (Protobuf). Protobuf – это формат сериализации данных, используемый по умолчанию для передачи данных между клиентом и сервером. Строгая типизация полей и бинарный формат для передачи структурированных данных потребляет меньше ресурсов. Время выполнения процесса сериализации/десериализации значительно меньше, как и размер сообщений в отличии от JSON/XML [1].

Для написания protobuf файлов используют язык описания интерфейсов (IDL). Например, чтобы описать структуру данных сообщения, нужно добавить message, имя структуры, а внутри тип, название и номер поля. Номера полей нужны для обратной совместимости, поэтому не стоит менять их последовательность при добавлении или удалении полей. Образец сообщения представлен в рисунке 1.

```
message Profile {
  int32 id = 1;
  string name = 2;
  int32 age = 3;
  string email = 4;
  enum PhoneType {
    MOBILE = 0;
    HOME = 1;
    WORK = 2;
  }
  message PhoneNumber {
    string number = 1;
    PhoneType type = 2;
  }
  repeated PhoneNumber phones = 5;
}
```

Рисунок 1 - Образец protobuf сообщения

Производительность. В Protocol Buffers данные представлены и передаются в бинарном формате, что существенно уменьшает время сериализации/десериализации

данных. Также с применением механизма Protocol Buffers размер передаваемых данных в разы меньше [2].

В настоящее время наиболее популярным RPC-решением для интеграции микросервисов является gRPC. Это высокопроизводительный фреймворк, разработанный компанией Google для вызовов удаленных процедур, работающий поверх протокола HTTP/2. Как и во многих RPC-системах, gRPC основан на идее определения сервиса, указывающий методы, которые можно вызвать удаленно с их параметрами и возвращаемыми типами.

На рисунке 2 представлена базовая схема интеграции микросервисов с помощью gRPC.

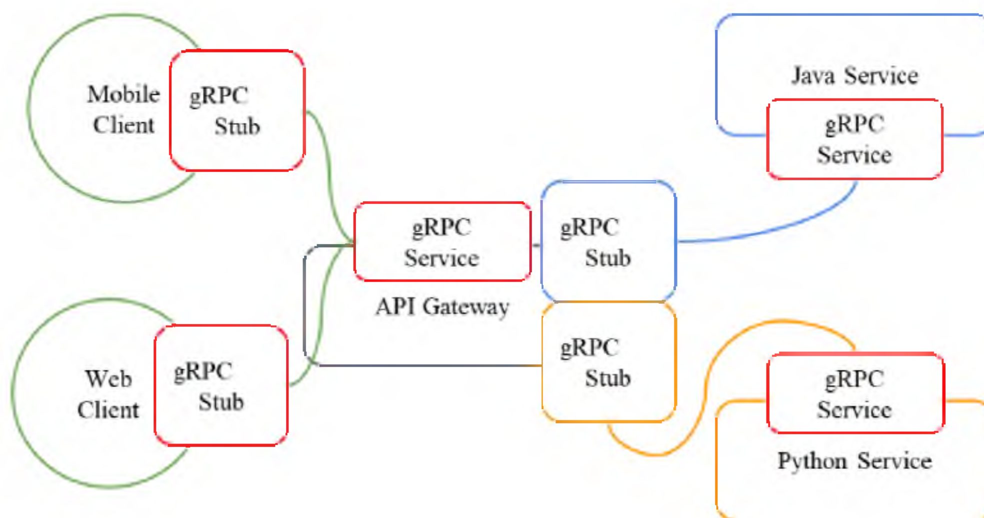

Рисунок 2 – Схема gRPC интеграции микросервисов

На стороне сервера реализуются методы, которые предоставляются для клиентов, и запускается gRPC-сервер для обработки клиентских запросов. На стороне клиента используется заглушка, которая предоставляет те же методы, что и сервер. Его высокая производительность достигается за счет использования протокола HTTP/2 и Protocol Buffers.

При микросервисной архитектуре имеется немалое количество интеграционных точек. Соответственно какие-либо сервисы рано или поздно будут содержать задержки и сбои. Поэтому стоит задача суметь справиться с этими проблемами без вывода из строя всей системы целиком. В архитектуре микросервисов в этом случае следует использовать тайм-ауты, а именно адаптивные тайм-ауты (рис. 3).

Алгоритм работает следующим образом: например, тайм-аут на запрос составляет 300 мс. Пусть обработка на шлюзе заняла 60 мс, соответственно необходимо, чтобы он обратился к микросервису 1 с тайм-аутам 300 – 60 = 240 мс. Если сетевой вызов занял 110 мс, а время обработки заняло 30 мс, соответственно на следующий микросервис 2 выделяется 240 – 110 – 30 = 100 мс. Если сетевой вызов занял 30 мс, а время обработки 80 мс, то когда придет время вызывать микросервис 3 тайм-аут уже будет превышен. В итоге не было необходимости выполнять запрос к микросервису 3. Это концепция называется тайм-аут распространение, где используется относительное значение во времени.

В gRPC фреймворке, к сожалению, тайм-аутов нет, но есть альтернативная концепция – дедлайны. Это абсолютное значение времени, по истечению которого запрос будет прерван. На рис. 4 представлена схема работы дедлайнов.

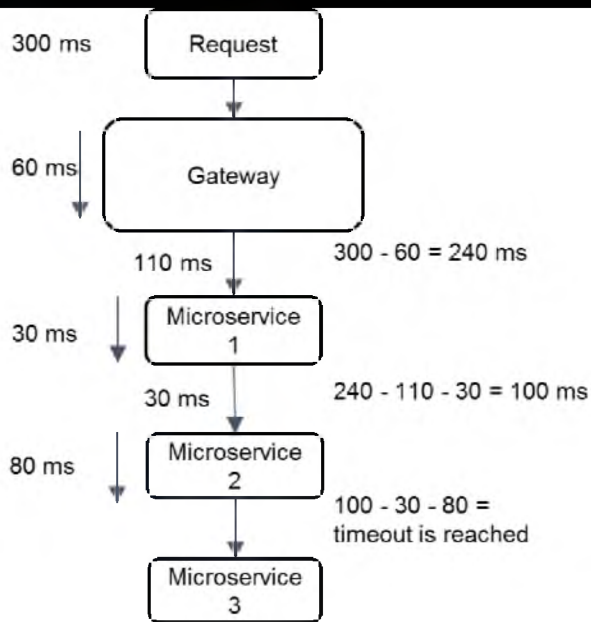

Рисунок 3 – Схема работы механизма «Тайм-аут распространение»

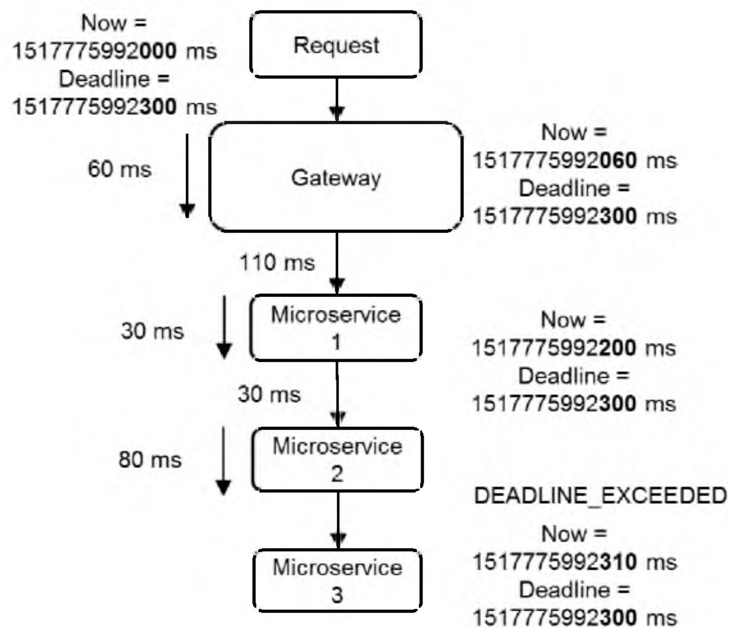

Рисунок 4 – Схема работы механизма дедлайнов

Алгоритм работает следующим образом: например, тайм-аут на запрос составляет 300 мс и начальное время выполнения запроса составляет 1517775992000 мс, тогда финальное время, до которого может протекать запрос, составляет 1517775992300. Пусть обработка на шлюзе заняла 60 мс, соответственно текущее время уже составляет 1517775992060 мс. Если сетевой вызов микросервиса 1 занял 110 мс, а время обработки запроса заняла 30 мс, то соответственно, текущее время после обработки запроса микросервисом 1 составляет 1517775992200 мс. Если сетевой вызов микросервиса 2 занял 30 мс, а время обработки заняло 80 мс, то когда придет время обрабатывать запрос микросервисом 3, текущее время уже будет составлять 1517775992310 мс, что превысит доступное абсолютное значение. Итого, как только время будет превышено, запросу

устанавливается статус код DEADLINE_EXCEEDED, по которому можно отслеживать, что данный запрос был отклонен, и уведомлять клиента о неудачной попытке выполнить запрос.

Следующую проблему, которую приходится решать – это неожиданная отмена запроса (отмена по инициативе пользователя, пропало интернет-соединение и т.д.). Соответственно, необходимо, чтобы при отмене входящего запроса отменялись все последующие. Данная концепция называется «отмена распространения». На рис. 5 представлена схема решения проблемы с помощью RPC.

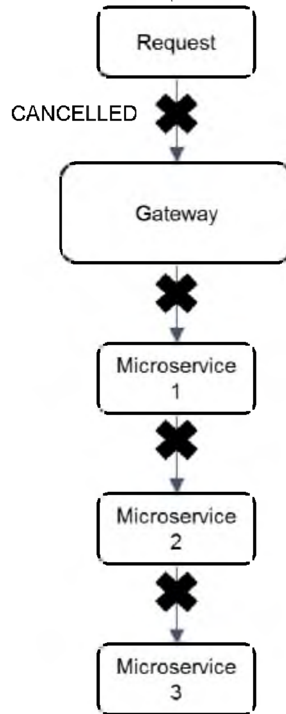

Рисунок 5 – Схема работы механизма «Отмена распространения»

Как только произошла отмена запроса все последующие запросы отменяются автоматически и получателю возвращается CANCELLED статус код.

Еще одна проблема, которую также хотелось бы отметить, это контроль запроса. В данном случае имеется в виду наличие механизма контроля запроса до его обработки. В RPC данная проблема решается с помощью интерсепторов. Интерсептор – это компонент, служащий для перехвата вызовов и встраивания необходимых условий. Схема работы данного механизма представлена на рис. 6.

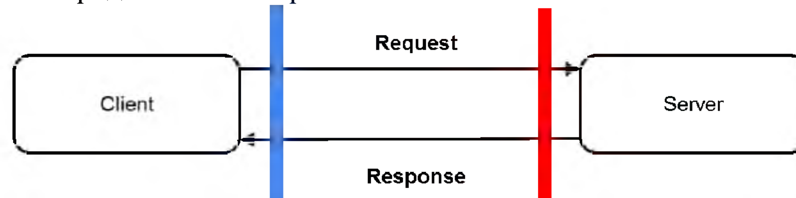

Рисунок 6 – Схема работы интерсептора

Данный механизм работает как фильтр запросов и полезен тогда, когда необходимо осуществить логирование, сбор метрик, аутентификацию и т.д.

Еще с одной проблемой, с которой рано или поздно сталкиваешься при построении микросервисной архитектуры, это горизонтальное масштабирование. На рис. 7 представлена схема решения указанной проблемы с помощью RPC.

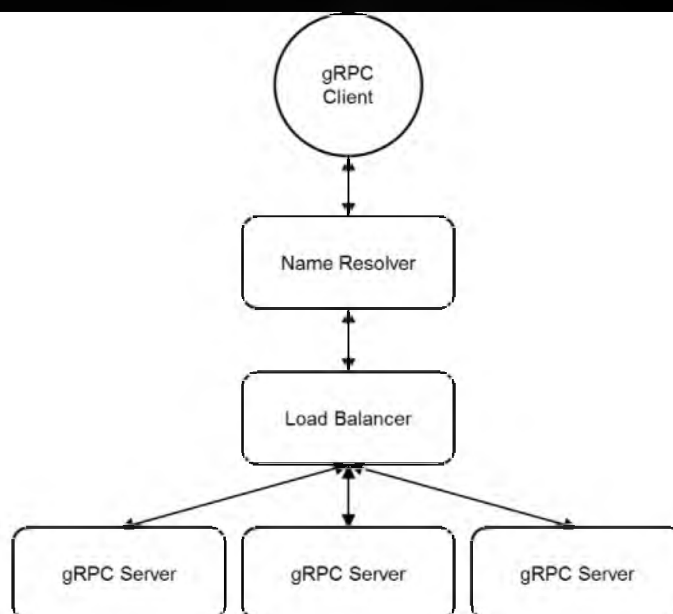

Рисунок 7 – Схема горизонтального масштабирования микросервисов

В систему добавляются два дополнительных компонента: обнаружитель сервисов и балансировщик нагрузки. Обнаружитель сервисов – это компонент, который, получает имя сервиса из запроса RPC-клиента и осуществляет поиск необходимого сервиса. Далее передает его адрес балансировщику нагрузки. Балансировщик нагрузки, в свою очередь, по полученному адресу, применяя различные алгоритмы, направляет запрос целевому менее нагруженному сервису.

Таким образом, при построении распределенной системы на основе микросервисов, как выяснилось, приходится решать достаточное количество проблем, которые возникают при интеграции микросервисов. Были выявлены и проанализированы такие проблемы, как наличие сбоев, сетевых задержек, бесполезная нагрузка на сервисы из-за долгих запросов, контроль запросов, горизонтальное масштабирование. Были представлены решения описанных выше проблем, которые возникают при интеграции, а также было представлено решение построения отказоустойчивой системы и интеграции микросервисов с помощью технологии RPC. К сожалению, при использовании RPC присутствует связанность клиентов (в виде заглушек) с реализацией сервиса. Однако, удалось добиться гибкого добавления изменений в существующие бизнес-модели, не нарушая работу существующих клиентов, а также повышения производительности за счет передачи данных по протоколу HTTP/2 в бинарном формате Protocol Buffers.

Список литературы:

1. Google Developers [Электронный ресурс]. Режим доступа: https://developers.google.com/protocol-buffers. Дата доступа: 10.09.2021.
2. DZone / Performance Zone [Электронный ресурс]. Режим доступа: https://dzone.com/articles/is-protobuf-5x-faster-than-json. Дата доступа: 30.08.2021.