**Повторите попытку**

gRPC избавляет от стресса, связанного с ошибками! Получите детальный контроль повторных попыток и подробную аналитику с поддержкой OpenCensus и OpenTelemetry.

Обзор

Повторные попытки — ключевой метод повышения надёжности сервисов. Повторяя неудачные операции, приложения могут устранять временные проблемы, такие как сбои в работе сети или сервера. Это важно для современных облачных приложений, чтобы справляться с неизбежными кратковременными сбоями.

Для оптимальной работы приложениям следует понимать, какие неудачные операции можно повторить, определять параметры экспоненциальной отсрочки для повторных попыток, устанавливать количество повторных попыток, а также отслеживать показатели повторных попыток.

Как работает повторная попытка gRPC-клиента

Встроенная в gRPC логика повторных попыток сохраняет историю вызовов для возможных повторных попыток и отслеживает события RPC. Даже если политика повторных попыток не настроена, gRPC всё равно сохраняет историю вызовов на случай, если потребуется выполнить прозрачную повторную попытку (обсуждается в следующем разделе). Обратите внимание, что «повторная попытка» означает замену неудачного вызова новым вызовом и воспроизведение истории вызовов для этого нового вызова.

Если выполняются определённые условия — RPC-запрос завершается с кодом состояния ошибки, соответствующим допустимым для повторных попыток кодам состояния, и остаётся в пределах лимита повторных попыток, — gRPC создаст новый поток повторных попыток после экспоненциальной задержки.

gRPC также поддерживает другие функции, такие как ограничение повторных попыток и откат на стороне сервера. Дополнительные сведения см. в gRFC для повторных попыток на стороне клиента.

Как только получен заголовок ответа, RPC завершается. Дальнейшие попытки не предпринимаются, и gRPC передаёт RPC приложению.

На графике ниже показан архитектурный обзор gRPC retry internal.

ApplicationgRPC ClientServerConfigure retry policy.Send request to dns:///my-serviceSave messageCreate initial attemptRPC closed with errorCreate retry attempt 1SuccessfulNo more retry. Proceed.ApplicationgRPC ClientServer

Retry configuration

Retries are enabled by default, but there is no default retry policy. Without a retry policy, gRPC cannot safely retry RPCs in most cases. Only RPCs that failed due to low-level races are retried, and only if gRPC is certain the RPCs have not been processed by a server. This is known as “transparent retry.” You can configure a retry policy to allow gRPC to retry RPCs in more circumstances and more aggressively. You can also disable retries entirely when creating a channel, which disables transparent retries and any configured retry policies.

Transparent Retry

Failure can occur in different stages. Even without an explicit retry policy, gRPC may perform transparent retries. The extent of these retries depends on when the failure happens:

* gRPC may do unlimited transparent retry when RPC never leaves the client.
* gRPC performs a single transparent retry when RPC reaches the gRPC server library, but has never been seen by the server application logic. Be aware of this type of retry, as it adds load to the network.

You can optimize your application’s retry functionality by focusing on key steps and configurations that gRPC supports.

* Max number of retry attempts
* Exponential backoff
* Set of retryable status codes

Retry is configurable via gRPC Service Config, at a per-method granularity. The configuration contains the following knobs:

"retryPolicy": {

"maxAttempts": 4,

"initialBackoff": "0.1s",

"maxBackoff": "1s",

"backoffMultiplier": 2,

"retryableStatusCodes": [

"UNAVAILABLE"

]

}

Jitter of plus or minus 20% is applied to the backoff delay to avoid hammering servers at the same time from a large number of clients. In the example configuration above, initialBackoff is set to 100ms, so the actual backoff delay after the first attempt will be for a random time period within the range [80ms, 120ms].

gRPC supports throttle limit that prevents server overload due to retries. Below is an examples of retry throttle configuration:

"retryThrottling": {

"maxTokens": 10,

"tokenRatio": 0.1

}

Для каждого сервера клиент gRPC отслеживает token\_count (первоначально установлено значение maxTokens). При неудачных RPC счётчик уменьшается на 1, при успешных RPC — увеличивается на tokenRatio. Если token\_count становится меньше половины maxTokens, повторные попытки приостанавливаются до восстановления счётчика.

Кроме того, хеджирование является дополнительной функцией к повторным попыткам и может быть настроено аналогичным образом. Подробнее см. в руководстве по хеджированию.

Повторите попытку Наблюдения

gRPC поддерживает предоставление показателей OpenCensus и OpenTelemetry, если включена функция повторных попыток. Вот пример доступной статистики повторных попыток OpenTelemetry:

* grpc.client.attempt.started
* grpc.client.attempt.duration
* grpc.client.attempt.sent\_total\_compressed\_message\_size
* grpc.client.attempt.rcvd\_total\_compressed\_message\_size

Показатели на уровне per cal:

* grpc.client.call.duration

И показатели на стороне сервера:

* grpc.server.call.started
* grpc.server.call.sent\_total\_compressed\_message\_size
* grpc.server.call.rcvd\_total\_compressed\_message\_size
* grpc.server.call.duration

Подробные сведения о метриках и трассировке, а также инструкции по настройке можно найти в gRFC для метрик Otel, gRFC для статуса повторных попыток.

Языковые руководства и примеры

ЯзыкПримерДокументацияC++ВпередПерейти к ПримеруJavaПример JavaДокументация JavaПитонПример Python

Дополнительные ресурсы

* gRFC для повторной попытки на стороне клиента
* gRFC для получения статуса повторной попытки
* Руководство по хеджированию
* Конфигурация службы gRPC
* gRFC для показателей отелей

Последнее изменение: 10 октября 2024 г.: Обновление схемы повторных попыток (#1371) (f13bf5b)