Yes, No, Maybe?

Error handling with gRPC examples
Agenda

Hello world with Protocol buffers and gRPC
   What's done by "magic"?

Error codes
   Did it work? yes, no, and maybe?
   Should I Retry?

TL;DR guidelines
Protocol buffers and gRPC

In 5-ish mins...

Protocol Buffers
The Greeter Service

1. define data structure schemas and programming interfaces
2. implementation code
3. remote procedure call (RPC) for the distributed client and server

```
SayHello("Fred", "de_DE")

"Guten Tag Fred!"

Translate("Hello", "de_DE")

"Guten Tag"
```
1. The service definition .proto

Protocol Buffers is a simple language-neutral and platform-neutral Interface Definition Language (IDL)

// Hello world
service Greeter {
  rpc SayHello (HelloRequest) returns (HelloReply) {}  
}

// Who to greet?
message HelloRequest {
  string name = 1;
  string locale = 2;
}

// The greeting.
message HelloReply {
  string greeting = 1;
}
1. The generated helper code

The protocol buffer compiler generates codes that have:

- remote interface *stub* for Client to call with the methods
- abstract interface for Server code to implement

Protocol buffer code will populate, serialize, and retrieve our request and response message types.
2. The implementation code

greeter_client.cc

#include "greeter.grpc.pb.h" // generated by protoc

GreeterClient(std::shared_ptr<Channel> channel)
    : stub_(Greeter::NewStub(channel)) {}  
std::string SayHello(const std::string& user) {
    HelloRequest request;
    request.set_name(user);
    HelloReply reply;
    ClientContext context;
    stub_->SayHello(&context, request, &reply);
    return reply.message();

greeter_server.cc

#include "greeter.grpc.pb.h" // generated by protoc

class GreeterServiceImpl final : public Greeter::Service {
    grpc::Status SayHello(ServerContext* context, const HelloRequest* request, HelloReply* reply) override {
        std::string prefix("Hello ");
        reply->set_message(prefix + request->name());
        return Status::OK;
    }
};
3. ...and the gRPC core

- Exposes core api to language api
- Filters
  - Implements RPC deadlines
  - Performs authentication
- Reconnect automatically with exponential backoff
- Takes care of socket creation, timers etc.
Status OK
Yes

It's done, ship it!
Status OK

But..

what happens if something fails?
Status Error: yes, no, maybe

- OK
  - It worked (as implemented)

gRPC Core Status Codes
- OK
- CANCELLED
- UNKNOWN
- INVALID_ARGUMENT
- DEADLINE_EXCEEDED
- NOT_FOUND
- ALREADY_EXISTS
- PERMISSION_DENIED
- UNAUTHENTICATED
- RESOURCE_EXHAUSTED
- FAILED_PRECONDITION
- ABORTED
- OUT_OF_RANGE
- UNIMPLEMENTED
- INTERNAL
- UNAVAILABLE
- DATA LOSS
Status Error : yes, no, maybe

greeter_client.cc

stub_->SayHello(&context, request, &reply);
return reply.message();

Status status = stub_->SayHello(&context, request, &reply);
if (status.ok()) {
  return reply.message();
} else {  // do something useful and cheap
  LOG_EVERY_N(ERROR, 10)
    << "Error " << google::COUNTER << " with status "
    << status.error_code() << status.error_message();
}
return "no hello available";
Status Error: yes, no, maybe

**OK** - It worked (as implemented)

It *wasn't* gRPC library.

---

**gRPC Core Status Codes**

- **OK**
- CANCELLED
- UNKNOWN
- INVALID_ARGUMENT
- DEADLINE_EXCEEDED
- NOT_FOUND
- ALREADY_EXISTS
- PERMISSION_DENIED
- UNAUTHENTICATED
- RESOURCE_EXHAUSTED
- FAILED_PRECONDITION
- ABORTED
- OUT_OF_RANGE
- UNIMPLEMENTED
- INTERNAL
- UNAVAILABLE
- DATA_LOSS
Maybe

Deadline exceeded... or did it?
Status Error: yes, no, maybe

- DEADLINE_EXCEEDED
- CANCELLED
- DEADLINE_EXCEEDED
- OK

Client

Server

gRPC Core Status Codes
- OK
- CANCELLED
- UNKNOWN
- INVALID_ARGUMENT
- DEADLINE_EXCEEDED
- NOT_FOUND
- ALREADY_EXISTS
- PERMISSION_DENIED
- UNAUTHENTICATED
- RESOURCE_EXHAUSTED
- FAILED_PRECONDITION
- ABORTED
- OUT_OF_RANGE
- UNIMPLEMENTED
- INTERNAL
- UNAVAILABLE
- UNAVAILABLE
- DATA_LOSS
Client is slow, the deadline expires on the internal queues.

Do the client and server agree?

no
DEADLINE_EXCEEDDED

Client

SayHello

Network

gRPC

Server

Network

gRPC

☠
Transport flaps

Stubs re-connects automatically.

Like magic but not actually magic!
DEADLINE_EXCEEDED

Client

SayHello

Server

gRPC

Network

☠

set

D_E

gRPC
Do the client and server agree?

no
Client's deadline reached before the response from the server.

Do the client and server agree?

- Server did wasted work.
- Client had already received deadline_exceeded from gRPC
DEADLINE_EXCEEDED

- Client's deadline reached before response from the server.
- Do client and server agree?
  - Server did wasted work.
  - Client had already received deadline_exceeded from gRPC

DEADLINE_EXCEEDED CAN MEAN IT WORKED!
DEADLINE_EXCEEDED

Servers. Cascading outage happen when servers spend resources handling requests that will exceed their deadlines on the client.

Clients. Think carefully about whether your request is idempotent before considering retries.

Servers can succeed and clients could still be retrying the requests!
// Check whether the client deadline has expired before processing. 
if (context->IsCancelled()) {
  LOG(INFO) << "Deadline exceeded or Client cancelled, abandoning.";
  return Status::CANCELLED;
}
Client's deadline reached before the response from the server.
- Server did no wasted work.
- Client receives `deadline_exceeded`

Do the client and server agree? No
// Avoid expensive backend calls for client who won't wait for results.
if (time_left < FLAGS_too_little_time_ms) {
  LOG(INFO) << "Don't call the backends, and set deadline exceeded.";
  return Status(grpc::DEADLINE_EXCEEDED, "Greeter service needs more time.");
}
The proto

// Hello world
service Greeter {
    // If request deadline < FLAGS_too_little_time_ms remains,
    // returns DEADLINE_EXCEEDED.
    rpc SayHello (HelloRequest) returns (HelloReply) {}()
}

message HelloRequest {
    string name = 1;
    string locale = 2;
}

message HelloReply {
    string greeting = 1;
}
Do the client and server agree?
maybe!
Do the client and server agree?
Yes, for the same reason.
Do the client and server agree?
Yes, but for different reasons.
Status Error: yes, no, maybe

gRPC Core Status Codes
- OK
- CANCELLED
- UNKNOWN
- INVALID_ARGUMENT
- DEADLINE_EXCEEDED
- NOT_FOUND
- ALREADY_EXISTS
- PERMISSION_DENIED
- UNAUTHENTICATED
- RESOURCE_EXHAUSTED
- FAILED_PRECONDITION
- ABORTED
- OUT_OF_RANGE
- UNIMPLEMENTED
- INTERNAL
- UNAVAILABLE
- DATA_LOSS

CLIENT
- CANCELLED
- UNKNOWN
- DEADLINE_EXCEEDED
- UNAUTHENTICATED
- RESOURCE_EXHAUSTED
- UNIMPLEMENTED
- INTERNAL
- UNAVAILABLE

SERVER
- CANCELLED
- UNKNOWN
- DEADLINE_EXCEEDED
- UNAUTHENTICATED
- RESOURCE_EXHAUSTED
- UNIMPLEMENTED
- INTERNAL
- UNAVAILABLE
Status Error: yes, no, maybe

**OK** - It worked (as implemented)

**MAYBE** - It _might_ have WORKED

**NO** - It _probably_ didn't WORK.

gRPC Core Status Codes
- **OK**
- CANCELLED
- UNKNOWN
- INVALID_ARGUMENT
- DEADLINE_EXCEEDED
- NOT_FOUND
- ALREADY_EXISTS
- PERMISSION_DENIED
- UNAUTHENTICATED
- RESOURCE_EXHAUSTED
- FAILED_PRECONDITION
- ABORTED
- OUT_OF_RANGE
- UNIMPLEMENTED
- INTERNAL
- UNAVAILABLE
- DATA_LOSS
Status Error: yes, no, maybe

**OK** - It worked (as implemented)

**MAYBE** - It might have worked

**NO** - It probably didn’t work.

Error codes are conventions not rules!

Is there any source of "truth"?

### gRPC Core Status Codes

- **OK**
- **CANCELLED**
- **UNKNOWN**
- **INVALID_ARGUMENT**
- **DEADLINE_EXCEEDED**
- **NOT_FOUND**
- **ALREADY_EXISTS**
- **PERMISSION_DENIED**
- **UNAUTHENTICATED**
- **RESOURCE_EXHAUSTED**
- **FAILED_PRECONDITION**
- **ABORTED**
- **OUT_OF_RANGE**
- **UNIMPLEMENTED**
- **INTERNAL**
- **UNAVAILABLE**
- **DATA_LOSS**

Error codes are conventions not rules!
No

But services can set expectations
Status Error: yes, no, maybe

google.rpc.Codes

Client errors
- INVALID_ARGUMENT
- FAILED_PRECONDITION
- OUT_OF_RANGE
- UNAUTHENTICATED
- PERMISSION_DENIED
- NOT_FOUND
- ABORTED
- ALREADY_EXISTS
- RESOURCE_EXHAUSTED
- CANCELLED

Server errors
- DATA_LOSS
- UNKNOWN
- INTERNAL
- NOT_IMPLEMENTED
- UNAVAILABLE
- DEADLINE_EXCEEDED

https://cloud.google.com/apis/design/errors#error_retries
Status Error: yes, no, maybe

google.rpc.Codes

Client errors
- INVALID_ARGUMENT
- FAILED_PRECONDITION
- OUT_OF_RANGE
- UNAUTHENTICATED
- PERMISSION_DENIED
- NOT_FOUND
- ABORTED
- ALREADY_EXISTS
- RESOURCE_EXHAUSTED
- CANCELLED

Server errors
- DATA_LOSS
- UNKNOWN
- INTERNAL
- NOT_IMPLEMENTED
- UNAVAILABLE
- DEADLINE_EXCEEDED

Clients should retry on UNKNOWN and UNAVAILABLE errors with exponential backoff.

The minimum delay should be 1s unless it is documented otherwise.

For RESOURCE_EXHAUSTED errors, the client may retry with minimum 30s delay.

https://cloud.google.com/apis/design/errors#error_retries
Status Error : yes, no, maybe

google.rpc.Codes

Client errors
- INVALID_ARGUMENT
- FAILED_PRECONDITION
- OUT_OF_RANGE
- UNAUTHENTICATED
- PERMISSION_DENIED
- NOT_FOUND
- ABORTED
- ALREADY_EXISTS
-
- CANCELLED

Server errors
- DATA_LOSS
- INTERNAL
- NOT_IMPLEMENTED
- DEADLINE_EXCEEDED

For all other errors:
Retry may not be applicable - first ensure your request is idempotent, and see the error message for guidance.
The translator

// Hello world in most languages
service Translator {
  // If translate returns INTERNAL, this serious error is not retryable.
  // If translate returns UNAVAILABLE, is retryable after 1s & exponential backoff.
  rpc Translate (TranslationRequest) returns (TranslationReply) {}
The greeter

// Hello world server
service Greeter {
    // If < FLAGS_too_little_time_ms remains, returns DEADLINE_EXCEEDED.
    // If locale not set, returns INVALID_ARGUEMENT. Not retryable.
    // INTERNAL is not retryable.
    // UNAVAILBLE, is retryable.
    rpc SayHello (HelloRequest) returns (HelloReply) {} 
}

message HelloRequest {
    string name = 1;
    string locale = 2;
}

message HelloReply {
    string greeting = 1;
}
// If locale not set, returns INVALID_ARGUMENT. Not retryable.
DEFINE_string(user, "world", "Who to greet.");
DEFINE_string(locale, "gd_IE", "Locale for greeting, default to Irish.");

// We want to know how often the service is broken.
if (status.error_code() == grpc::Status::INTERNAL) {
  ++num_broken;
...

// We want to know how often we're retrying.
if (status.error_code() == grpc::Status::UNAVAILABLE) {
  ++num_retry;
...

TL;DR

There is no definitive answer but maybe some guidelines?
Status Error: yes, no, maybe

Tell clients which are temporary and which are permanent errors.

Invalid_argument will never work regardless of the state of the server. Unavailable might work later if the server was down.

If more than one error code applies return the most specific.

Out of range versus failed precondition. Permission denied < unauthenticated < resources exhausted.

Hide implementation, unless you want client decisions to depend on it.

Don't blindly propagate errors. They can contain confidential data.
More?

@sre_grain

Or find me in the hallway