gRPC: Remote Procedure Call

Vojtěch Horký          Petr Tůma

2010 – 2022

This work is licensed under a “CC BY-NC-SA 3.0” license. Created to support the Charles University Performance Evaluation lecture. See http://d3s.mff.cuni.cz/teaching/introduction-to-middleware for details.

Contents

1 Technology Overview 1
2 Assignment Part I 2
3 Server Implementation 2
4 Client Implementation 3
5 Assignment Part II 4

1 Technology Overview

Technology Overview

Goals
Provide platform independent remote procedure call mechanism.

Features
- Protocol buffers as interface description language.
- Stub code generation for multiple languages (C++, Java, Python, Go, Ruby, JavaScript, PHP, C# ...).
- Binary transport format with compact data representation.
- Supports streaming arguments during remote call.
- Synchronous and asynchronous invocation code.
- Compression support at transport level.
- Security support at transport level.

Examples To Begin With ...

> git clone http://github.com/d-iii-s/teaching-introduction-middleware.git

C
> cd teaching-introduction-middleware/src/grpc-basic-server/c
> cat README.md

Java
Service Specification Example

syntax = "proto3";

message AnExampleRequest { ... }
message AnExampleResponse { ... }

service AnExampleService {
  rpc OneToOneCall (AnExampleRequest) returns (AnExampleResponse) {}  
  rpc OneToStreamCall (AnExampleRequest) 
    returns (stream AnExampleResponse) {}  
  rpc StreamToStreamCall (stream AnExampleRequest)  
    returns (stream AnExampleResponse) {}  
}

2 Assignment Part I

Assignment

Server
Implement a server that will provide information on current time.
- The server should accept a spec of what fields to return.
- Fields should be standard YYYY-MM-DD HH:MM:SS.

Client
Implement a client that will query server time:
- Pick a random combination of fields.
- Query information on current time.
- Print the time.

Interoperability
Implement compatible clients and servers in two languages.

3 Server Implementation

C++ Service Basics

Implementation
class MyService : public AnExampleService::Service {
  grpc.Status OneToOne (grpc.ServerContext *context,  
    const AnExampleRequest *request, AnExampleResponse *response) {
    // Method implementation goes here ...
    return (grpc.Status::OK);
  }
  ...
Execution

MyService service;
grpc.ServerBuilder builder;
builder.AddListeningPort("localhost:8888", grpc.InsecureServerCredentials ());
builder.RegisterService (&service);
std::unique_ptr<grpc.Server> server (builder.BuildAndStart ());
server->Wait ();

Java Service Basics

Implementation
class MyService extends AnExampleServiceGrpc.AnExampleServiceImplBase {
    @Override
    public void OneToOne (
        AnExampleRequest request,
        io.grpc.stub.StreamObserver<AnExampleResponse> responseObserver) {
        // Method implementation goes here ...
        responseObserver.onNext (response);
        responseObserver.onCompleted ();
    }
}

Execution

io.grpc.Server server = io.grpc.ServerBuilder
    .forPort (8888).addService (new MyService ()).build ().start ();
server.awaitTermination ();

Python Service Basics

Implementation
class MyServicer (AnExampleServiceServicer):
    def OneToOne (self, request, context):
        # Method implementation goes here ...
        return response

Execution

server = grpc.server (futures.ThreadPoolExecutor (max_workers = SERVER_THREAD_COUNT))
add_AnExampleServiceServicer_to_server (MyServicer (), server)
server.add_insecure_port ("localhost:8888")
server.start ()

4 Client Implementation

C++ Client Basics

Connection

std::shared_ptr<grpc.Channel> channel = grpc.CreateChannel ("localhost:8888", grpc.InsecureChannelCredentials ());

Invocation
grpc::ClientContext context;
AnExampleResponse response;
std::shared_ptr<AnExampleService::Stub> stub = AnExampleService::NewStub (channel);
grpc::Status status = stub->OneToOne (&context, request, &response);
if (status.ok ()) {
  // Response available here ...
}

### Java Client Basics

**Connection**

```java
io.grpc.ManagedChannel channel = io.grpc.ManagedChannelBuilder
    .forAddress ("localhost", 8888)
    .usePlaintext ()
    .build ();
```

**Invocation**

```java
AnExampleServiceGrpc.AnExampleServiceBlockingStub stub =
    AnExampleServiceGrpc.newBlockingStub (channel);
AnExampleResponse response = stub.OneToOne (request);
// Response available here ...
```

### Python Client Basics

**Connection**

```python
with grpc.insecure_channel ("localhost:8888") as channel:
```

**Invocation**

```python
stub = AnExampleServiceStub (channel)
response = stub.OneToOne (request)
# Response available here ...
```

### Show Your Code ...

**Query Host Name**

```
> hostname
u1-22
```

**Run Screen Sharing**

```
> x11vnc -viewonly
```

### 5 Assignment Part II

**Assignment**

**Performance**

Measure the performance of your implementation.

**Experiment Design**

Stick to the following, or provide arguments for why not:
- Random field mix, each field with probability 1/2.
- Measure at least two minutes long traffic.
- Report average invocation throughput.
- No printing during measurement.
- Compare with past assignments.

**Submission**

**GitLab**

**Requirements**
- Use the assignment subdirectory.
- Write brief report in SOLUTION.md.
- Include build scripts with instructions.
- Do not commit binaries or temporary build artifacts.
- Tag your solution with `task-03` and push the tag.