gRPC

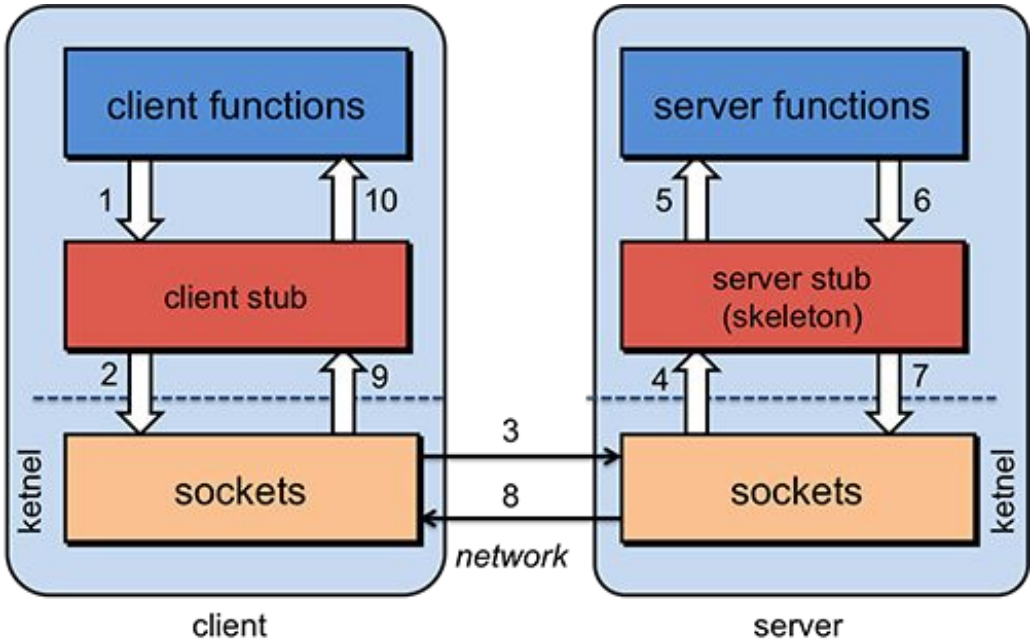
Lab posted online

http://cs.brown.edu/courses/cs138/s18/content/labs/grpc.pdf

Motivation: PuddleStore!

- Every project progressively contained less and less RPC support code
- In Puddlestore you will be creating your own RPC methods and data from scratch

Remote Procedure Call



Protocol Buffers

- “Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data”
- Similar to XML but more efficient
- Starting point for RPC implemenation

Definitions

When to use RPC calls?

- Need to change the state of a remote node
- Need to retrieve information from the remote node

Protobuf file

- Filename ends with “.proto”
- Includes syntax and package
- Contains messages and service

File:

```
1  syntax = “proto3”;
2
3  package myPackage;
4
5  // define service
6
7  // define messages
...
```

Protobuf messages

- Protobuf's version of "struct"
- Types include: bool, int32, float, double, and string
- Nested object definitions
- Enum values
- Repeated fields
- Numbers represent unique tags for each field
  - = 1; = 2; ... etc

```
message Person {
  string name = 1;
  int32 id = 2;
  string email = 3;

  enum PhoneType {
    MOBILE = 0;
    HOME = 1;
    WORK = 2;
  }

  message PhoneNumber {
    string number = 1;
    PhoneType type = 2;
  }

  repeated PhoneNumber number = 4;
}
```

Protobuf services

- Protobuf's version of "methods"
- Must contain one input and one output
  - can use "empty" data types if field is not needed
- Inputs and outputs are predefined messages
- Requires the "rpc" keyword

```
service RaftRPC {
  rpc JoinCaller(RemoteNode) returns (Ok) {}
  rpc GetIdCaller(Empty) returns (IdReply) {}
 }
```

Compiling protobuf

- Compile

```
$ protoc -I=$SRC_DIR $SRC_DIR/example_rpc.proto --go_out=$DST_DIR

$ /course/cs1380/bin/cs138_protoc -I . ./example_rpc.proto --go_out=plugins=grpc:.
```

- Creates `example_rpc.pb.go` with Go implementation of defined objects and interface
- Messages -> structs
- Service -> interface

Implementing gRPC methods

- Implement the service interface (e.g. RaftNode implementing `Caller` functions)
- Service RPC methods will compile to the form

```
JoinCaller(context.Context, *RemoteNode) returns (*Ok, error)
```

- Note:
  - Contains context object as an argument
  - RPC object arguments are pointers
  - Additional error return value
- (Recommended) RPC methods that implements client side of call... Calls Caller methods.

Communication

Client vs Server

Client:

- Connects to server through address (includes port)
- Converts local data to RPC messages and receives RPC messages back
- (optional) methods end in “RPC”

Server:

- listens for clients on port
- Implements methods in service interface
- (optional) methods end in “Caller”
- Converts RPC messages to local data and replies with RPC messages

Server/Client communication

- Without gRPC:
  - Server listens on port, Client connects to server’s address and port
- With gRPC:
  - Similar but wrapped with grpc objects and functions

Receiving RPCs (Server)

- Requires:
  - An object that implements the services interface in *.pb.go
  - A listening connection object (using port)
  - A grpc Server object
- Steps:
  - Register object as grpc server
  - Listen for RPCs

```
// Create conn
conn := Listen(...)
// Create grpc server
s := grpc.NewServer()
// Create receiving object
myNode := Node {
  // data fields
}
// register and serve
RegisterMyServiceServer(s,
&myNode)
go s.Serve(conn)

// implement Caller methods
```

Sending RPCs (Client)

- Requires:
  - A grpc Client object (using address + port)
- Steps:
  - Use client object to call methods on server
- DialOptions examples:
  - grpc.WithInsecure()
  - grpc.FailOnNonTempDialError(true)

```
// Create grpc client
conn, err := grpc.Dial(addr,...
dialOptions)
// Register client
cc = NewMyServiceClient(conn)

// user “Caller” methods on client
obj
cc.JoinCaller(...)
```

Raft

- RaftNodes is both a client and a server:
  - Each RaftNode object contains `*grpc.Server` to receive RPC calls
  - Calls RPC on other nodes by using RPC methods with extesnsive ConnClient Managment

Additional resources

- Protobuf:
  - https://developers.google.com/protocol-buffers/
- gRPC:
  - https://grpc.io/
- Lab exercise:
  - http://cs.brown.edu/courses/cs138/s18/content/labs/grpc.pdf

Questions?

Goodluck