gRPC-based Dependency Recovery

I. Background

Dependency analysis is essentially important during the code review process for program understanding. Tremendous efforts have been made to to extract semantic dependencies within individual programming languages. For example, both LSIF implementations [6] and Scitools Understand [5] can index code bases of mainstream languages like C/C++ or Java, and recover dependency relations like `definition`, `reference`, `function call`, etc. However, there is little, if any, static dependency analysis on **R**emote **P**rocedure **C**alls (RPCs for short). In a typical RPC scenario, one procedure can invoke another procedure in a different address space, just like a `regular function call`. gRPC [3] is a high-performance, open-source universal RPC framework (Figure. 1) widely-used in our company. It allows clients and servers communicate in a microservice style architecture. Usually, the communication messages are serialized and deserialized in Protocol Buffers [2] format (protobuf for short) and RPCs are processed with the help of gRPC method stubs. To better understand how such a software system work, we look for a technique that can *statically* recover the dependency relations induced by gRPC-based remote calls.

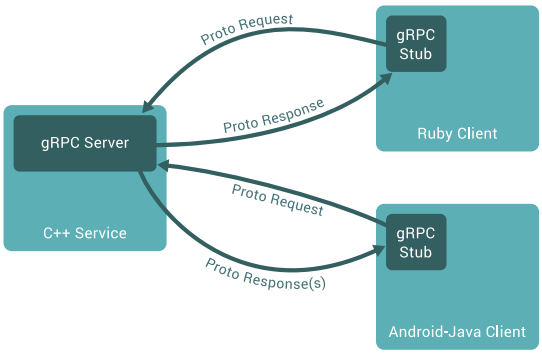

Figure 1. Clients and server communicate via gRPC

II. Technical Challenges

The challenges lie in the modeling of the communication channel operations and the gRPC message specifications.

gRPC uses HTTP/2 to transport. IP addresses and ports are used to establish and shutdown the channel. This requires the dependency recovery technique to be aware of the semantics of gRPC libraries as implemented in C++, Java, Golang.

Meanwhile, gRPC by default uses protobuf for serializing structured messages. It defines `service` and `message` and code will be generated in specific languages like C/C++, Java, Golang, etc. As an example, Listing 1 is a protobuf file which contains a `service` named `HelloWorld` and several kinds of `messages` like `Point`, `Rectangle`. Inside `HelloWorld` there are four kinds of RPCs, unary RPC (`GetFeature`), Server streaming RPC (`ListFeatures`), client streaming RPC (`RRoute`), and bidirectional streaming RPC (`RChat`). The signatures of method stubs and the protobuf compiler `protoc` determine the generated code, which depicts how the RPCs are processed, together with corresponding gRPC libraries.

```
service HelloWorld {
  rpc GetFeature(Point) returns (Feature) {}
  rpc ListFeatures(Rectangle) returns (stream
    Feature) {}
  rpc RRoute(stream Point) returns (RSummary) {}
  rpc RChat(stream RNote) returns (stream RNote) {}
}
message Point {
  int32 x = 1;
  int32 y = 2;
}
...
```

Listing 1. A simple protobuf code snippet with four different RPCs

III. Related Work

We are not aware of researches on statically recovering gRPC dependencies, but some related work may be inspiring. For language-specific semantic dependencies, LSIF [6] specification can be extended for such purpose but existing LSIF open-source implementations only resolve semantic dependencies within individual languages. [1] [4] dynamically track dependencies between microservices, and some of the techniques may also be used.

IV. Technical Requirements

The dependency recover technique should answer 1) whether two gRPC-based software components, written in C++, Java, or Golang, may ever invoke remote calls; 2) if so, show the involved method stubs in both original protobuf files and generated code. The deliverables include the technical report and the corresponding implementation. It should cover 80% use cases of gRPC and the time cost should be no more than 5x of LSIF-based [6] semantic dependency analyzers lsif-clang, lsif-java, lsif-go.

References

[1] S. Esparrachiari, T. Reilly, and A. Rentz. Tracking and controlling microservice dependencies: Dependency management is a crucial part of system and software design. *Queue*, 16(4):44–65, aug 2018.

[2] G. Inc. Protocol buffers documentation. https://protobuf.dev/, 2008.

[3] G. Inc. grpc: A high performance, open-source universal rpc framework. https://grpc.io/, 8 2016.

[4] S. Luo, H. Xu, C. Lu, K. Ye, G. Xu, L. Zhang, Y. Ding, J. He, and C. Xu. Characterizing microservice dependency and performance: Alibaba trace analysis. In *SoCC '21*, page 412–426, New York, NY, USA, 2021.

[5] I. Scientific Toolworks. Understand: An ide and static code analysis tool by scitools. https://www.scitools.com/.

[6] Sourcegraph. Lsif.dev. https://lsif.dev/, 2020.