# Introduction to gRPC

@Microservice ID Meetup

# $ whoami

Prakash Divy

Software Engineer @katadotai

prakash@kata.ai

https://github.com/prakashdivyy

https://www.linkedin.com/in/prakash-divy/

# gRPC

# gRPC Remote Procedure Calls

# Remote Procedure Call?

ACM DL DIGITAL LIBRARY

Check out a preview of the *next* ACM DL

**Implementing remote procedure calls**

Full Text: PDF Get this Article

1984 Article


Authors: Andrew D. Birrell Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA

Bruce Jay Nelson Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA


Bibliometrics

- Citation Count: 445
- Downloads (cumulative): 13,543
- Downloads (12 Months): 848
- Downloads (6 Weeks): 48


Published in:

- Journal

ACM Transactions on Computer Systems (TOCS) TOCS Homepage archive

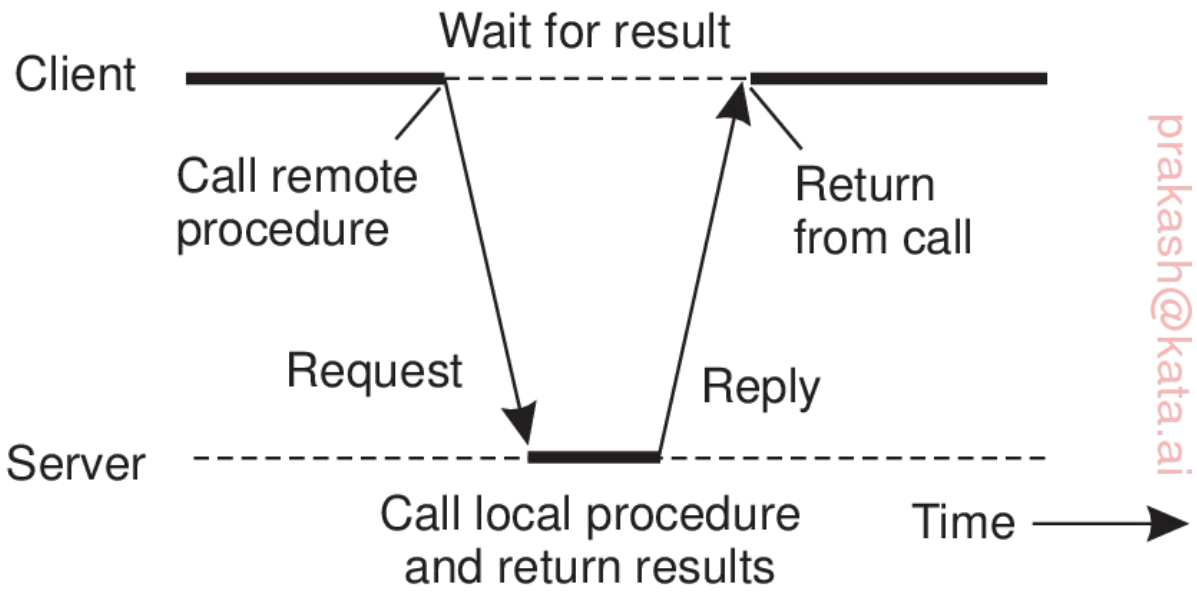
Volume 2 Issue 1, February 1984

Pages 39-59

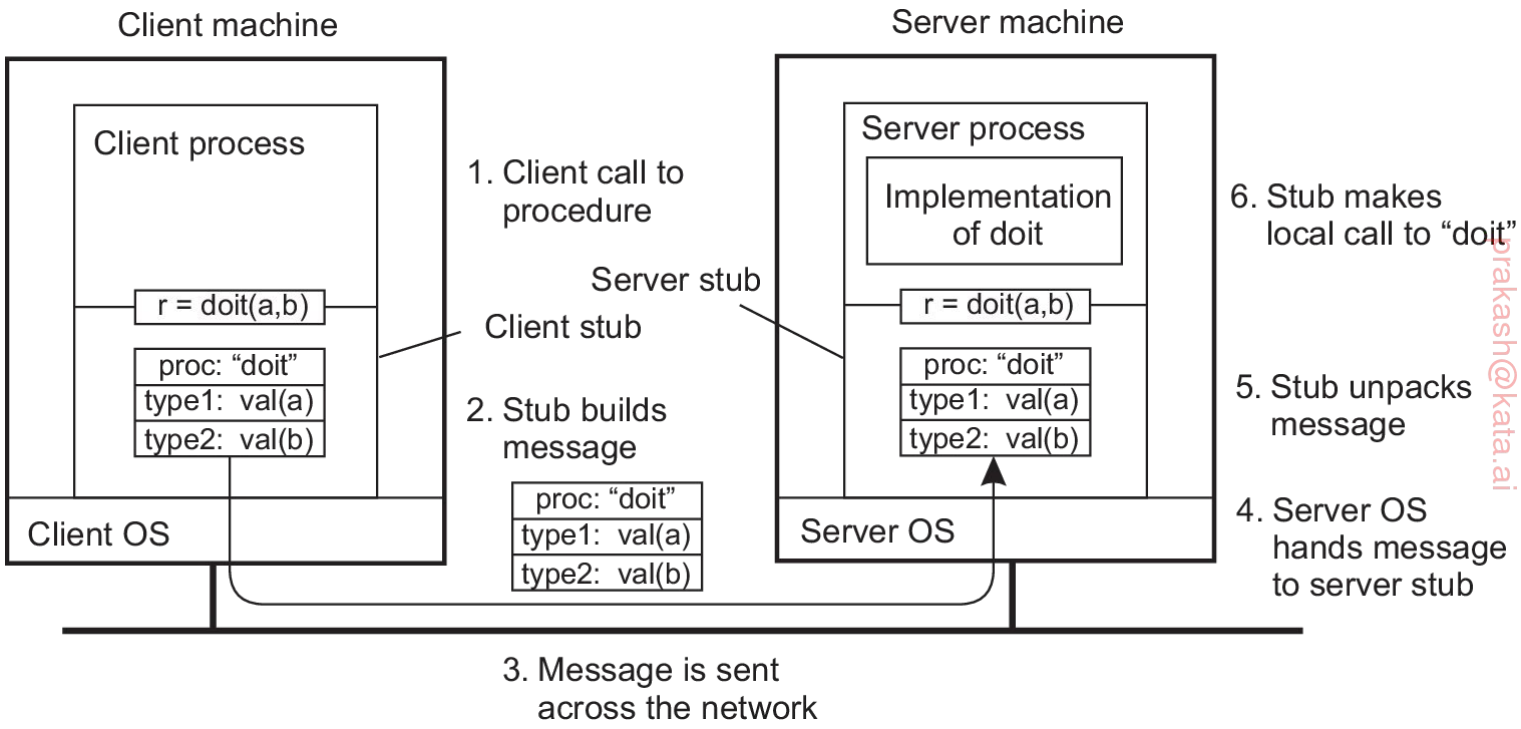
ACM New York, NY, USA

table of contents doi>10.1145/2080.357392

# In a Nutshell



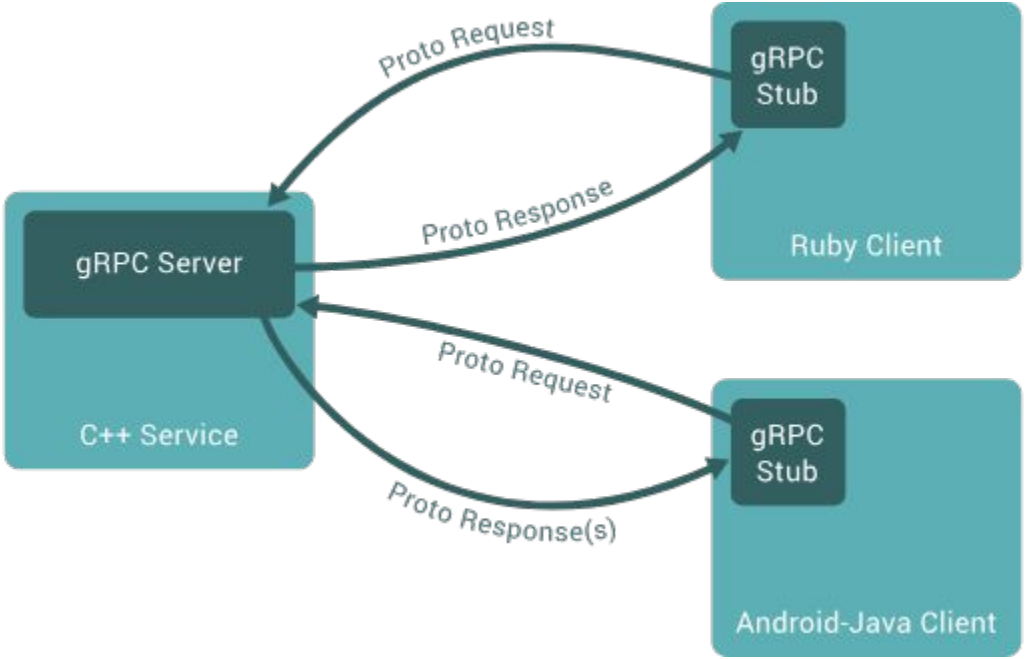
# More Complex



# About gRPC

- Stubby (Google's Internal RPC)
- March 2015, build the next version of Stubby
- Become part of Cloud Native Computing Foundation (CNCF) since 1 March 2017
- More info: https://cloud.google.com/blog/products/gcp/grpc-a-true-internet-scale-rpc-framework-is-now-1-and-ready-for-production-deployments?m=0

# How It Works?


Proto Request
gRPC Stub
Ruby Client
Proto Response
gRPC Server
C++ Service
Proto Request
gRPC Stub
Proto Response(s)
Android-Java Client

# Benefits

- Low Latency Using HTTP/2
- Efficient Serialization Using Protocol Buffer
- Multi-language Support
- Connection Options (unary, server streaming, client streaming, bi-directional streaming)

# HTTP/2

HTTP/2 vs HTTP/1.1 Demo:
http://www.http2demo.io/

Further reading:
https://www.cncf.io/blog/2018/08/31/grpc-on-http-2-engineering-a-robust-high-performance-protocol/

gRPC vs REST over HTTP/2:
https://stackoverflow.com/questions/44877606/is-grpchttp-2-faster-than-rest-with-http-2

# Protocol Buffers

- Google's mature open source mechanism for serializing structured data.
- Data structured as messages.
- Steps:
  a. Define the structure in a proto file.
  b. Using the protocol buffer compiler (protoc) to generate data access classes.
  c. With plugin can generated gRPC client and server.

```
syntax = "proto3";

package example;

service ExampleService {
    rpc SayHello(HelloRequest) returns (HelloResponse);
    rpc DoAddition(NumberRequest) returns (NumberResponse);
    rpc DoSubtraction(NumberRequest) returns (NumberResponse);
}

message HelloRequest {
    string greeting = 1;
}

message HelloResponse {
    string reply = 1;
}

message NumberRequest {
    int32 first_number = 1;
    int32 second_number = 2;
}

message NumberResponse {
    int32 result = 1;
}
```

# Protocol Buffers vs XML

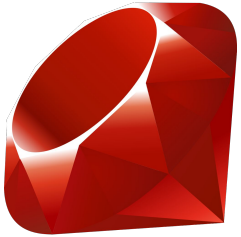
- Simpler
- 3 to 10 times smaller
- 20 to 100 times faster
- Less ambiguous
- Generate data access classes that are easier to use programmatically

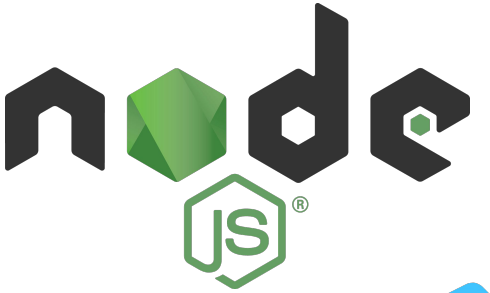
```
# Textual representation of a protocol buffer.
# This is *not* the binary format used on the wire.
person {
  name: "John Doe"
  email: "jdoe@example.com"
}
```

```
<person>
  <name>John Doe</name>
  <email>jdoe@example.com</email>
</person>
```

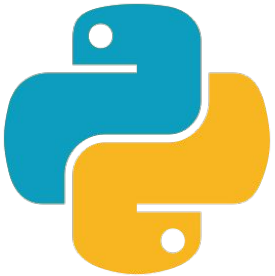
# Supported Language

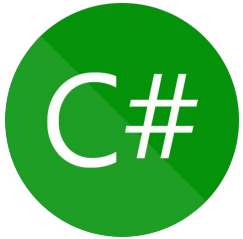










LET'S GET THIS PARTY STARTED!
imgflip.com

# Step 1: Create Proto

```
syntax = "proto3";

package example;

service ExampleService {
    rpc SayHello(HelloRequest) returns (HelloResponse);
    rpc DoAddition(NumberRequest) returns (NumberResponse);
    rpc DoSubtraction(NumberRequest) returns (NumberResponse);
}

message HelloRequest {
    string greeting = 1;
}

message HelloResponse {
    string reply = 1;
}

message NumberRequest {
    int32 first_number = 1;
    int32 second_number = 2;
}

message NumberResponse {
    int32 result = 1;
}
```

# Step 2: Generate Server and Client Stubs

```
protoc --proto_path=example --proto_path=../ --go_out=plugins=grpc:example example.proto
```

# Step 3: Create Server or Client

```go
package main

import (
  "fmt"
  "log"
    "net"

    "golang.org/x/net/context"
    "google.golang.org/grpc"
    "grpc-example/go/example"

)

type Server struct{}

// SayHello function implementation
func (s *Server) SayHello(context context.Context, in *example.HelloRequest) (*example.HelloResponse,
error) {
    return &example.HelloResponse{Reply: "Hello " + in.Greeting + "!"}, nil
}

// DoAddition function implementation
func (s *Server) DoAddition(context context.Context, in *example.NumberRequest)
(*example.NumberResponse, error) {
    return &example.NumberResponse{Result: in.FirstNumber + in.SecondNumber}, nil
}

// DoSubtraction function implementation
func (s *Server) DoSubtraction(context context.Context, in *example.NumberRequest)
(*example.NumberResponse, error) {
    return &example.NumberResponse{Result: in.FirstNumber - in.SecondNumber}, nil
}

func main() {
    // create a listener on TCP port 50050
    lis, err := net.Listen("tcp", fmt.Sprintf(":%d", 50050))
    if err != nil {
      log.Fatalf("failed to listen: %v", err)
    }

    // create a gRPC server object
    grpcServer := grpc.NewServer()
    // attach the Ping service to the server
    example.RegisterExampleServiceServer(grpcServer, &Server{})

    // start the server
    if err := grpcServer.Serve(lis); err != nil {
      log.Fatalf("failed to serve: %s", err)
    }
}
```

```go
package main

import (
    "log"

    "golang.org/x/net/context"
    "google.golang.org/grpc"
    "grpc-example/go/example"

)

func main() {
    conn, err := grpc.Dial(":50050", grpc.WithInsecure())
    if err != nil {
        log.Fatalf("did not connect: %s", err)
    }
    defer conn.Close()

    c:= example.NewExampleServiceClient(conn)

    responseHello, err := c.SayHello(context.Background(), &example.HelloRequest{Greeting: "Prakash"})

    if err != nil {
        log.Fatalf("Error when calling SayHello: %s", err)
    }

    log.Printf("Response from server: %s", responseHello.Reply)

    responseAddition, err := c.DoAddition(context.Background(), &example.NumberRequest{FirstNumber: 1,
SecondNumber: 2})

    if err != nil {
        log.Fatalf("Error when calling DoAddition: %s", err)
    }

    log.Printf("Response from server: %d", responseAddition.Result)

    responseSubtraction, err := c.DoSubtraction(context.Background(),
&example.NumberRequest{FirstNumber: 1, SecondNumber: 2})

    if err != nil {
        log.Fatalf("Error when calling DoSubtraction: %s", err)
    }

    log.Printf("Response from server: %d", responseSubtraction.Result)
}
```

# Step 4: Running Server or Client

# Demo

https://github.com/prakashdivyy/grpc-example

# Further Reading

- https://grpc.io/docs/guides/
- https://developers.google.com/protocol-buffers/docs/overview
- https://grpc.io/docs/guides/concepts/
- https://www.slideshare.net/Codemotion/introduction-to-grpc-a-general-rpc-framework-that-puts-mobile-and-http2-first-mete-atamel-codemotion-amsterdam-2017

FIN!