10 DE JULHO DE 2022

GRPC & PROTOBUF COMO SOLUÇÃO DE MENSAGERIA

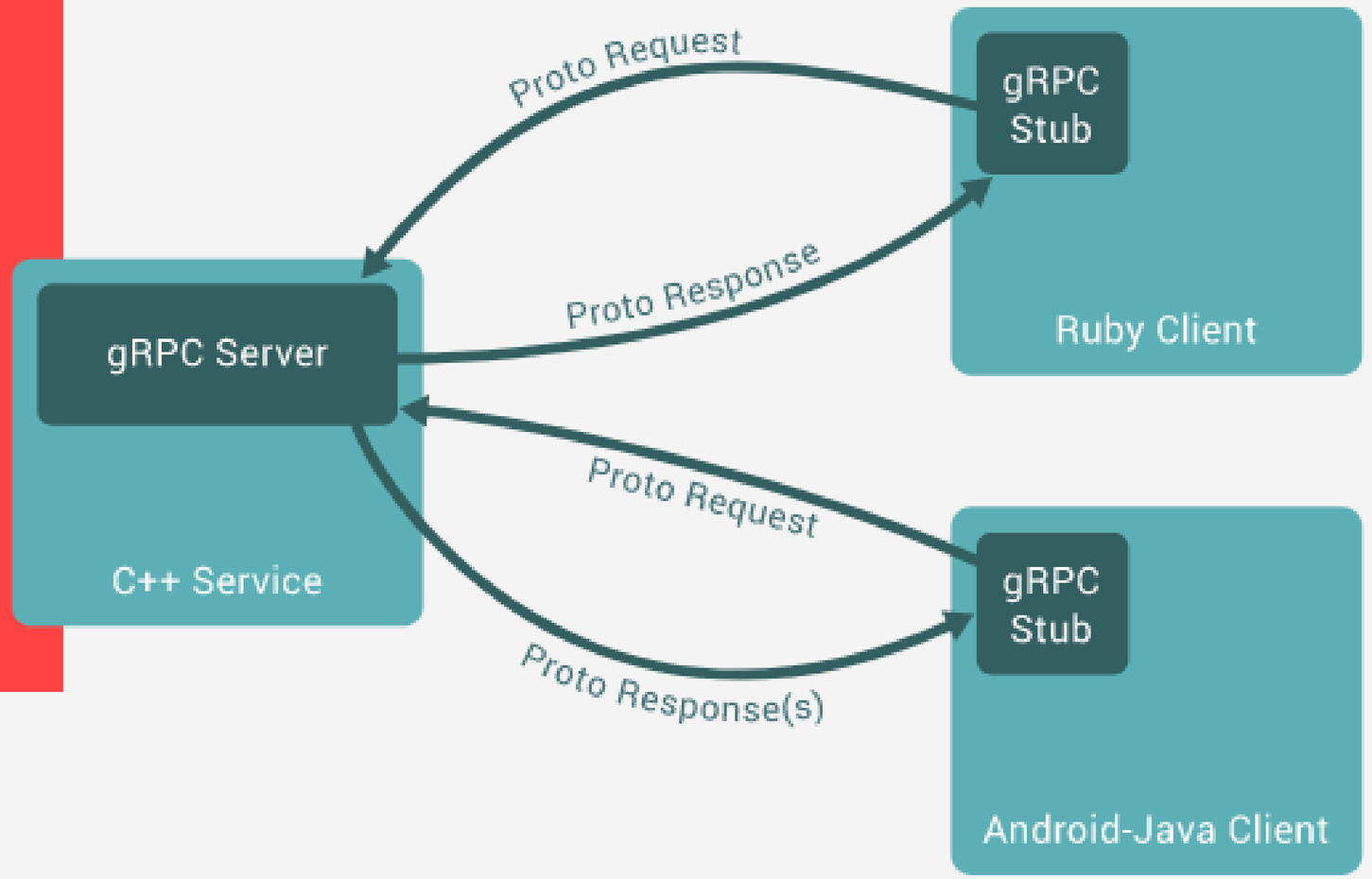
Laboratório 02

Equipe

Thiago Paiva

190020377

gRPC


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

gRPC- Tipos de Serviços

- Unary RPC
  - Cliente envia uma requisição e o servidor retorna uma resposta
- Server streaming RPCs
  - Cliente envia uma requisição e o servidor retorna uma stream para o cliente ler uma quantidade variada de mensagens como resposta
- Client streaming RPCs
  - Cliente envia uma stream de mensagem como requisição e o servidor lê as mensagem e retorna uma resposta
- Bidirectional streaming RPCs
  - Cliente envia uma stream de mensagem como requisição e o servidor devolve uma stream de mensagem como resposta.

Protobuf

```
message  Pessoa {
    string nome = 1;
    Date dataNasc = 2;
    repeated Date faltas = 3;
    optional Money renda = 4;
}
```

- Serialização de dados estruturados
- Binário
- Compacto
- Eficiente
- Suporta diversos tipos de dados:
  - Escalares (double, float, int32, int64, uint32, bytes, ...)
  - Duration
  - Timestamp
  - Money
  - Date
  - ...

Protobuf - Serviços

- Unary RPC
  - **rpc** SayHello(HelloRequest) **returns** (HelloResponse);
- Server streaming RPCs
  - **rpc** LotsOfReplies(HelloRequest) **returns** (**stream** HelloResponse);
- Client streaming RPCs
  - **rpc** LotsOfReplies(HelloRequest) **returns** (**stream** HelloResponse);
- Bidirectional streaming RPCs
  - **rpc** BidiHello(**stream** HelloRequest) **returns** (**stream** HelloResponse);

HTTP/2

- Binary framing layer
- One connection per origin
- Streams, messages, and frames
- Request and response multiplexing
- Stream prioritization
- Server push
- Header compression

SOLUÇÃO

ENCONTRAR O MENOR E MAIOR VALOR

Definição do Serviço

```
syntax = "proto3";
package minmax;

service MinMax {
    rpc Find (FindRequest) returns (FindResponse) {}
}

message FindRequest {
    repeated float numbers = 1;
}

message FindResponse {
    float min = 1;
    float max = 2;
}
```

Definição do Serviço

```
syntax = "proto3";
package minmax;

service MinMax {
    rpc Find (FindRequest) returns (FindResponse) {}
}

message FindRequest {
    repeated float numbers = 1;
}

message FindResponse {
    float min = 1;
    float max = 2;
}
```

Diagrama da solução

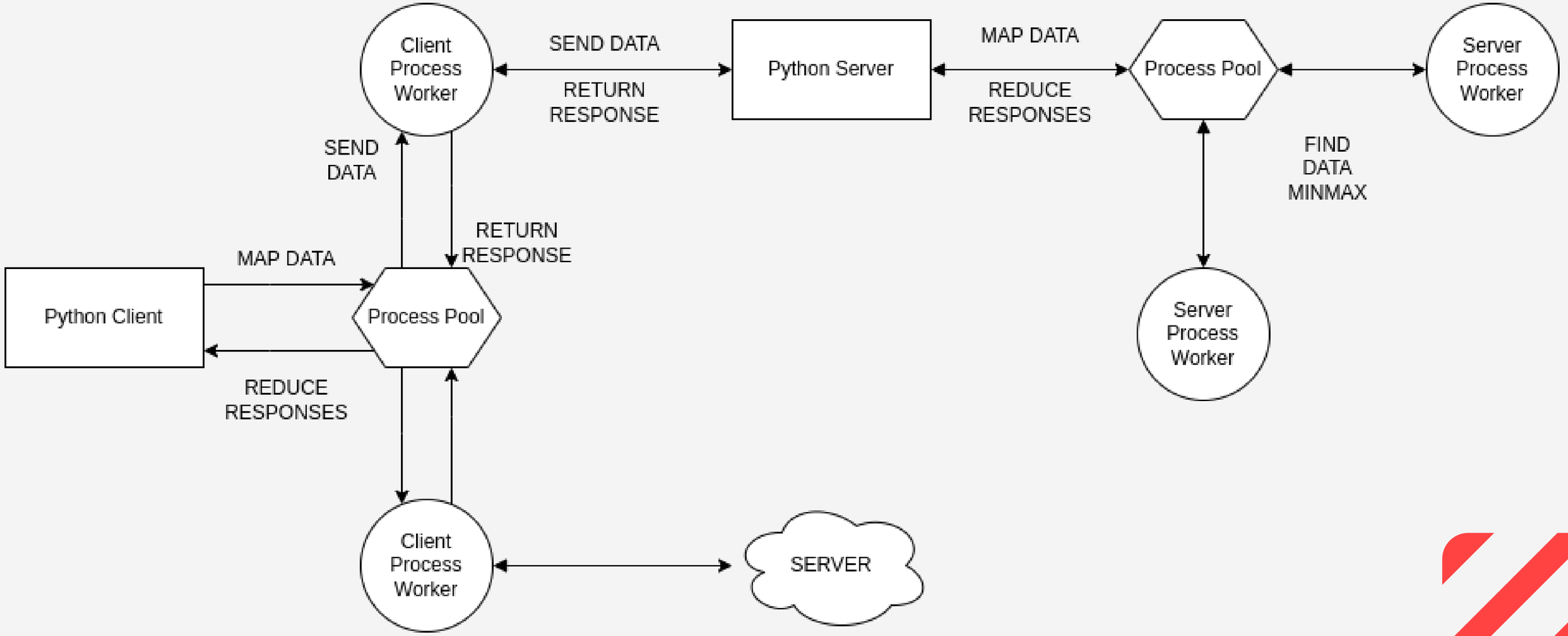
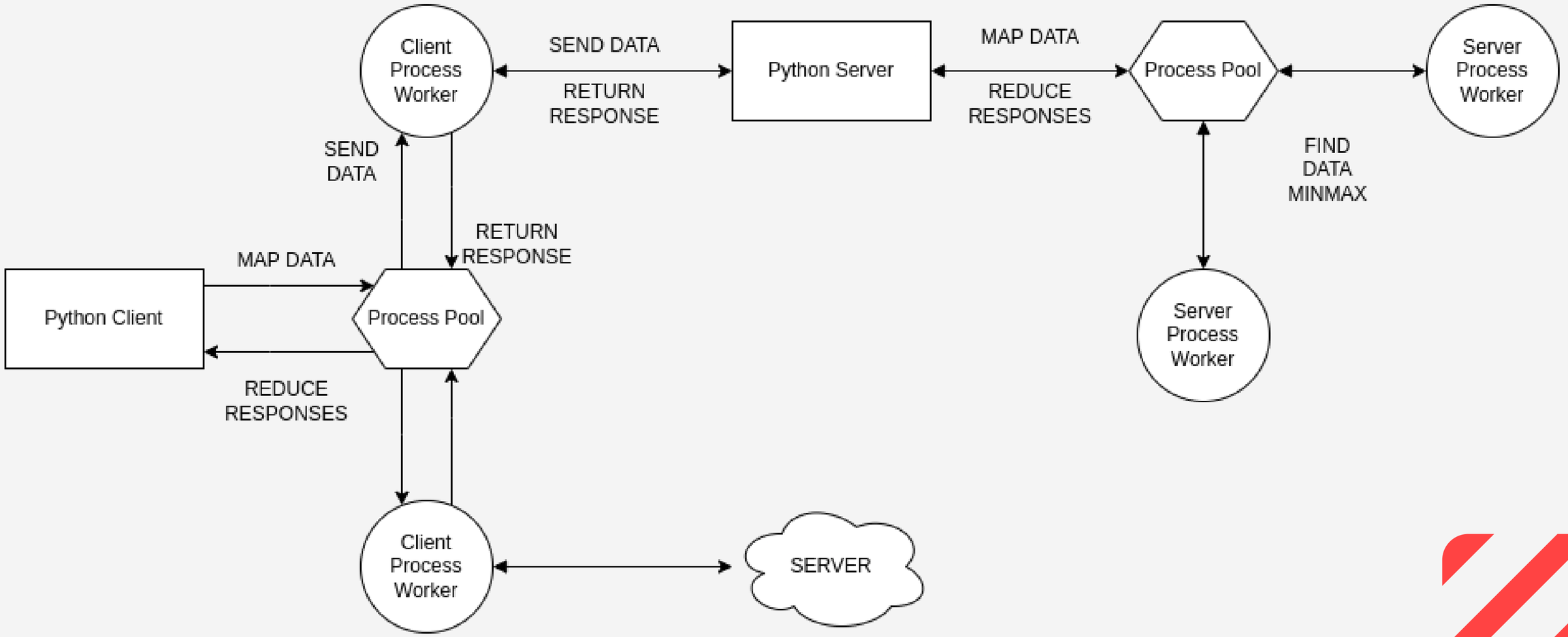


Diagrama da solução



Cliente

minmax_client.py

```python
def main():
    MAX = 500_000
    numbers = [sqrt((i - uniform(0, MAX)/2)**2) for i in range(MAX)]

    n_workers = len(sys.argv) - 1
    hosts = sys.argv[1:]
    offset = ceil(MAX/n_workers)
    numbers = [numbers[i*offset:(i+1)*offset] for i in range(n_workers)]

    with futures.ProcessPoolExecutor(max_workers=n_workers) as executor, timeit():
        workers_args = zip(numbers, hosts)
        responses = executor.map(run, workers_args)

        final_response = reduce(merge_responses, responses, minmax_pb2.FindResponse(min=inf, max=-inf))
        print(f"MIN = {final_response.min}\nMAX = {final_response.max}")


def merge_responses(response, parcial):
    response.min = min(parcial.min, response.min)
    response.max = max(parcial.max, response.max)
    return response
```

Cliente

minmax_client.py

```
19  def run(args):
20      numbers, target = args
21
22      with grpc.insecure_channel(target) as channel:
23          stub = minmax_pb2_grpc.MinMaxStub(channel)
24          request = minmax_pb2.FindRequest(numbers=numbers)
25          return stub.Find(request)
```

Servidor

minmax_server.py

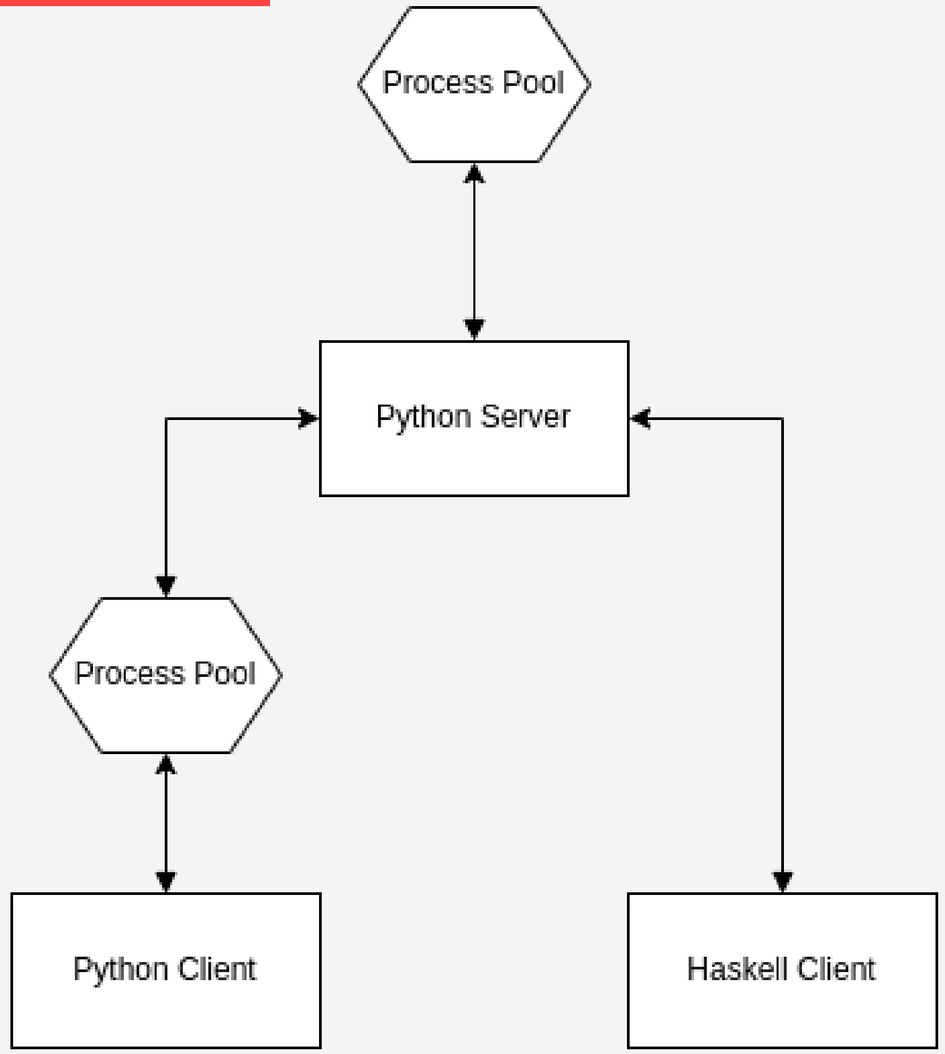
```python
def serve():
    n_workers = int(sys.argv[1])
    workers = futures.ProcessPoolExecutor(max_workers=n_workers)

    server = grpc.server(futures.ThreadPoolExecutor())
    minmax_pb2_grpc.add_MinMaxServicer_to_server(MinMax(workers, n_workers), server)
    server.add_insecure_port(f'[::]:{sys.argv[2]}')
    server.start()
    server.wait_for_termination()

    workers.shutdown()
```

Servidor

minmax_server.py

```python
class MinMax(minmax_pb2_grpc.MinMaxServicer):

    def __init__(self, workers: futures.ProcessPoolExecutor, n_workers: int):
        self.workers = workers
        self.n_workers = n_workers

    def Find(self, request, context):
        length = len(request.numbers)

        print(f"INFO: {length} foram recebidos")

        offset = ceil(length/self.n_workers)
        numbers = (request.numbers[i*offset:(i+1)*offset] for i in range(self.n_workers))

        responses = self.workers.map(find_minmax, numbers)
        return reduce(merge_responses, responses, minmax_pb2.FindResponse(min=inf, max=-inf))

def find_minmax(numbers: Iterable[float]):
    numbers = sorted(numbers)
    return minmax_pb2.FindResponse(min=numbers[0], max=numbers[-1])

def merge_responses(response, parcial):
    response.min = min(parcial.min, response.min)
    response.max = max(parcial.max, response.max)
    return response
```

Diagrama da solução

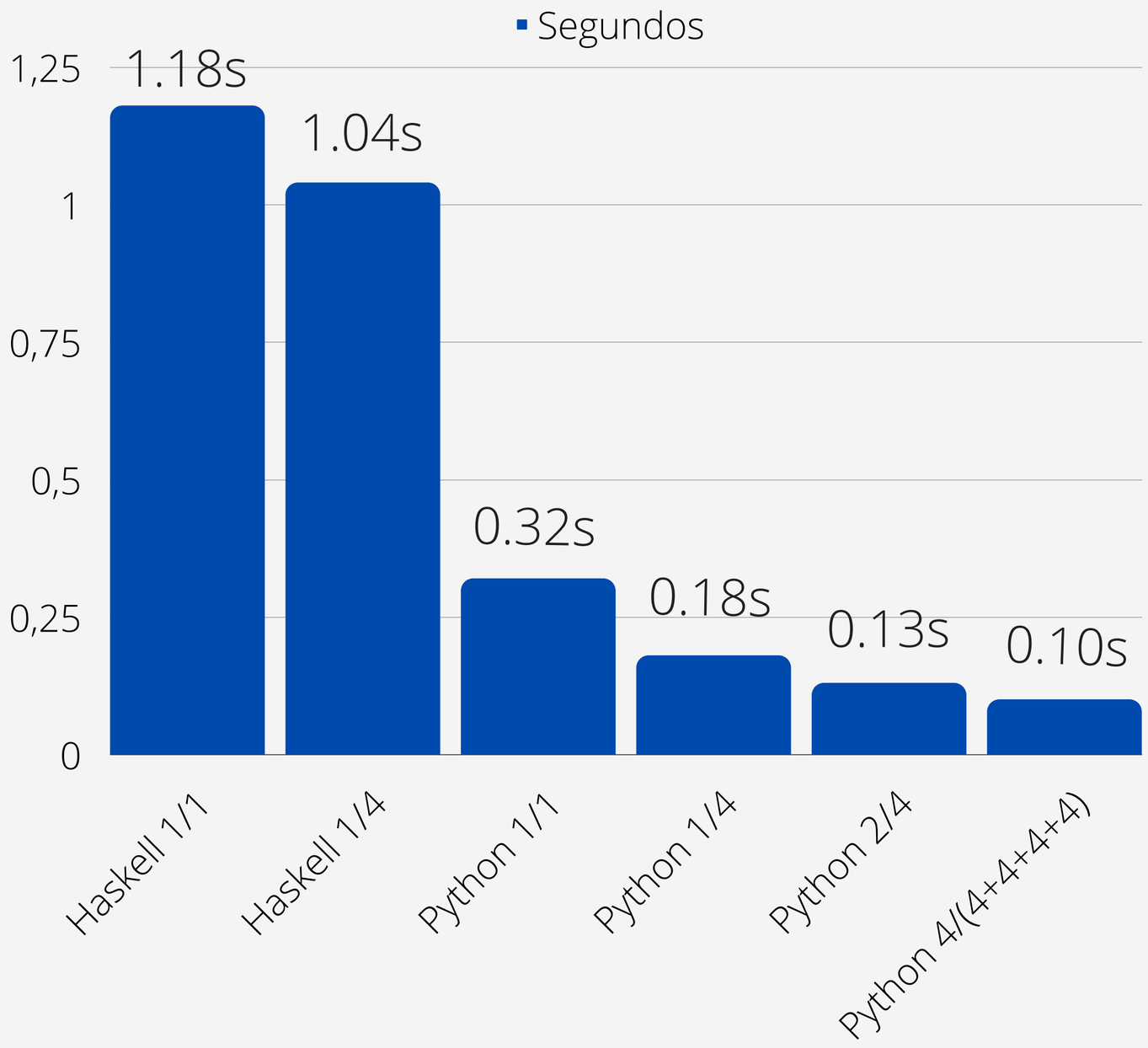


Cliente Haskell

Main.hs

```haskell
clientConfig :: ClientConfig
clientConfig = ClientConfig { clientServerHost = "127.0.0.1"
                            , clientServerPort = 50051
                            , clientArgs = []
                            , clientSSLConfig = Nothing
                            , clientAuthority = Nothing
                            }

run :: Vector Float -> IO ()
run numbers = withGRPCClient clientConfig $ \client -> do
  MinMax{..} <- minMaxClient client

  let req = FindRequest numbers
  res <-  minMaxFind (ClientNormalRequest req 60 mempty)

  case res of
      ClientErrorResponse err -> putStrLn $ show err
      ClientNormalResponse (FindResponse minNumber maxNumber) _ _ _ _
        -> putStrLn $ "MIN = " ++ show minNumber ++ " MAX = " ++ show maxNumber

  return ()
```

Benchmark


Segundos
1,25
1
0,75
0,5
0,25
0
1.18s
1.04s
0.32s
0.18s
0.13s
0.10s
Haskell 1/1
Haskell 1/4
Python 1/1
Python 1/4
Python 2/4
Python 4/(4+4+4+4)

Referências

gRPC. What is gRPC?. Acesso em 10/07/2022

Google. Protocol Buffers Overview. Acesso em 10/07/2022

Grigorik e Surma. Introduction to HTTP/2. Acesso em 10/07/2022