Exercício 2
RPC e MOM

Aplicação e Avaliação de Desempenho

Implementação RPC usando gRPC

Código - Serviço

- ProtoBuffs e Plugins em Go

```
1   syntax = "proto3";
2
3   package fibonacci;
4
5   service Fibonacci {
6       rpc getFibo(FibRequest) returns (FibResponse) {}
7   }
8
9   // Mensagem de Request
10  message FibRequest {
11      int32 number = 1;
12  }
13
14  // Mensagem de Response
15  message FibResponse {
16      int32 number = 1;
17  }
```

Código - Servidor gRPC

```go
package main

import (…
)

type fibonacciServer struct{}

func (s *fibonacciServer) GetFibo(ctx context.Context, req *fibonacci.FibRequest) (*fibonacci.FibResponse, error) {
    return &fibonacci.FibResponse{ Number: application.CalcFibonacci(req.Number) }, nil
}

func main() {
    conn, err := net.Listen("tcp", ":"+strconv.Itoa(shared.GRPC_PORT))
    shared.CheckError(err)

    servidor := grpc.NewServer()
    fibonacci.RegisterFibonacciServer(servidor, &fibonacciServer{})

    fmt.Println("Servidor pronto ...")

    // Register reflection service on gRPC servidor.
    reflection.Register(servidor)

    err = servidor.Serve(conn);
    shared.CheckError(err)
}
```

Código - Cliente gRPC

- Estabelecendo conexão
- Criando contexto com informações sobre Request

```
25      conn, err := grpc.Dial(ipContainer + ":" +
26          strconv.Itoa(shared.GRPC_PORT), grpc.WithInsecure())
27      shared.CheckError(err)
28
29      defer conn.Close()
30
31      fib := fibonacci.NewFibonacciClient(conn)
32
33      // Contacta o servidor
34      ctx, cancel := context.WithTimeout(context.Background(),
35          time.Minute) // havia um problema com o time.Second . 1s -> 1m
36      defer cancel()
37
```

Código - Cliente gRPC

- Fazendo requisições
- Invocando operação remota

```
38      number, _ := strconv.Atoi(os.Args[2])
39
40      fmt.Println("Fibonacci,Answer,Time")
41      for i = 0; i < shared.SAMPLE_SIZE; i++ {
42          t1 := time.Now()
43
44          // Invoca operação remota
45          msgReply, err := fib.GetFibo(ctx, &fibonacci.FibRequest{ Number: int32(number)})
46          shared.CheckError(err)
47
48          t2 := time.Now()
49          x := float64(t2.Sub(t1).Nanoseconds()) / 1000000
50
51          s := fmt.Sprintf("%d,%d,%f", number, msgReply.Number, x)
52          fmt.Println(s)
53      }
```

Implementação MOM usando RabbitMQ

Código - Servidor MOM

- Criando conexão com RabbitMQ
- Criando canal

```
24      conn, err := amqp.Dial("amqp://guest:guest@" +
25          ipContainer + ":" +
26          strconv.Itoa(shared.RABBITMQ_PORT) +"/")
27      shared.CheckError(err)
28      defer conn.Close()
29
30      ch, err := conn.Channel()
31      shared.CheckError(err)
32      defer ch.Close()
```

Código - Servidor MOM

- Criando filas de comunicação

```
34      // declaração de filas , cria se não existir
35      requestQueue, err := ch.QueueDeclare( // mesma fila de envio
36          "request", false, false, false, false, nil, )
37      shared.CheckError(err)
38
39      replyQueue, err := ch.QueueDeclare( // mesma fila de respostas
40          "response", false, false, false, false, nil, )
41      shared.CheckError(err)
42
43      // prepara o recebimento de mensagens do cliente
44      msgsFromClient, err := ch.Consume(requestQueue.Name, "", true, false,
45          false, false, nil, )
46      shared.CheckError(err)
47
48      fmt.Println("Servidor pronto...")
```

Código - Servidor MOM

```
50      forever := make(chan bool) // travar
51      go func(){
52          for d := range msgsFromClient {
53
54              // recebe request
55              msgRequest := shared.Request{}
56              err := json.Unmarshal(d.Body, &msgRequest)
57              shared.CheckError(err)
58
59              // processa request
60              r := application.CalcFibonacci(msgRequest.Req)
61
62              // prepara resposta
63              replyMsg := shared.Response{Res: r}
64              replyMsgBytes, err := json.Marshal(replyMsg)
65              shared.CheckError(err)
66
67              // publica resposta
68              err = ch.Publish("", replyQueue.Name, false, false,
69                  amqp.Publishing{ContentType: "text/plain", Body: replyMsgBytes,})
70              shared.CheckError(err)
71          }
72      }()
73
74      <- forever
```

Código - Cliente MOM

```
23      // conecta ao servidor de mensageria
24      conn, err := amqp.Dial("amqp://guest:guest@" +
25          ipContainer + ":" +
26          strconv.Itoa(shared.RABBITMQ_PORT) +"/")
27      shared.CheckError(err)
28      defer conn.Close()
29
30      // cria o canal
31      ch, err := conn.Channel()
32      shared.CheckError(err)
33      defer ch.Close()
34
35      // declara  filas, cria se não existir
36      requestQueue, err := ch.QueueDeclare( // fila de envio
37          "request", false,   false,  false,  false,  nil, )
38      shared.CheckError(err)
39
40      replyQueue, err := ch.QueueDeclare( // fila de respostas
41          "response", false, false, false, false, nil, )
42      shared.CheckError(err)
43
44      // cria consumidor <-> fila de respostas -> async
45      msgsFromServer, err := ch.Consume(replyQueue.Name, "", true, false,
46          false, false, nil, )
47      shared.CheckError(err)
48
```

Código - Cliente MOM

```
51      for i := 0; i<shared.SAMPLE_SIZE; i++{
52
53          t1 := time.Now()
54
55          // prepara request
56          msgRequest := shared.Request{Req: int32(number)} // Fibonacci 5
57          msgRequestBytes,err := json.Marshal(msgRequest)
58          shared.CheckError(err)
59
60          // publica request <-> fila de envio
61          err = ch.Publish("", requestQueue.Name, false, false,
62              amqp.Publishing{ContentType: "text/plain", Body: msgRequestBytes,})
63          shared.CheckError(err)
64
65          // recebe resposta em bytes
66          x := <- msgsFromServer
67
68          // deserializa
69          msgReply := shared.Response{}
70          err = json.Unmarshal([]byte(x.Body), &msgReply)
71          shared.CheckError(err)
72
73          t2 := time.Now()
74          xtime := float64(t2.Sub(t1).Nanoseconds()) / 1000000
75          s := fmt.Sprintf("%d,%d,%f", number, msgReply.Res, xtime)
76          fmt.Println(s)
77      }
```

Avaliação de Desempenho

Máquina:

- Memória: 7,7 GiB
- Desktop
- OS: Manjaro xfce
- Processador: Intel® Core™ i7-3770 CPU @ 3.40GHz × 8

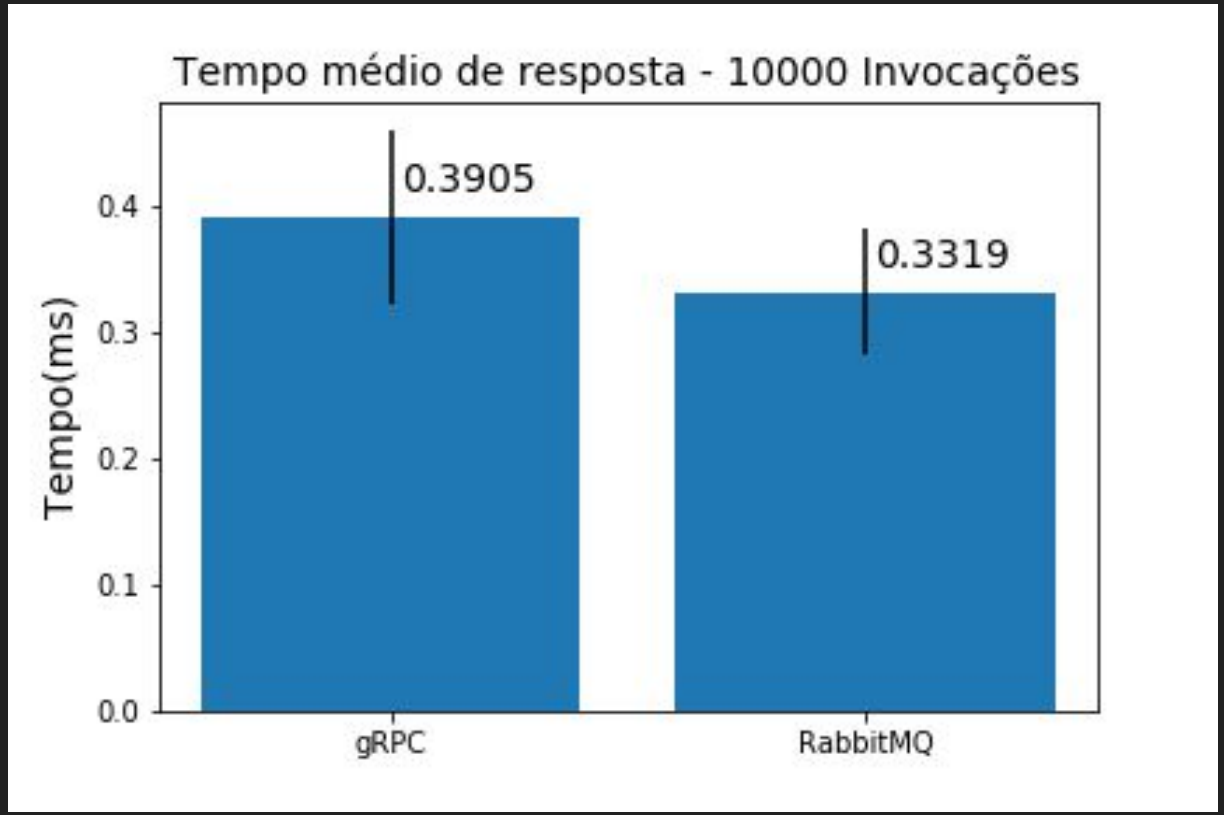
Preparação de Ambiente

- Máquina recém inicializada
- 1 Docker (Imagem: golang + repositório)
- 1 Docker (Imagem: rabbitmq)
- Sem alocação especial de recursos para os containers
- Experimento realizado 30 vezes

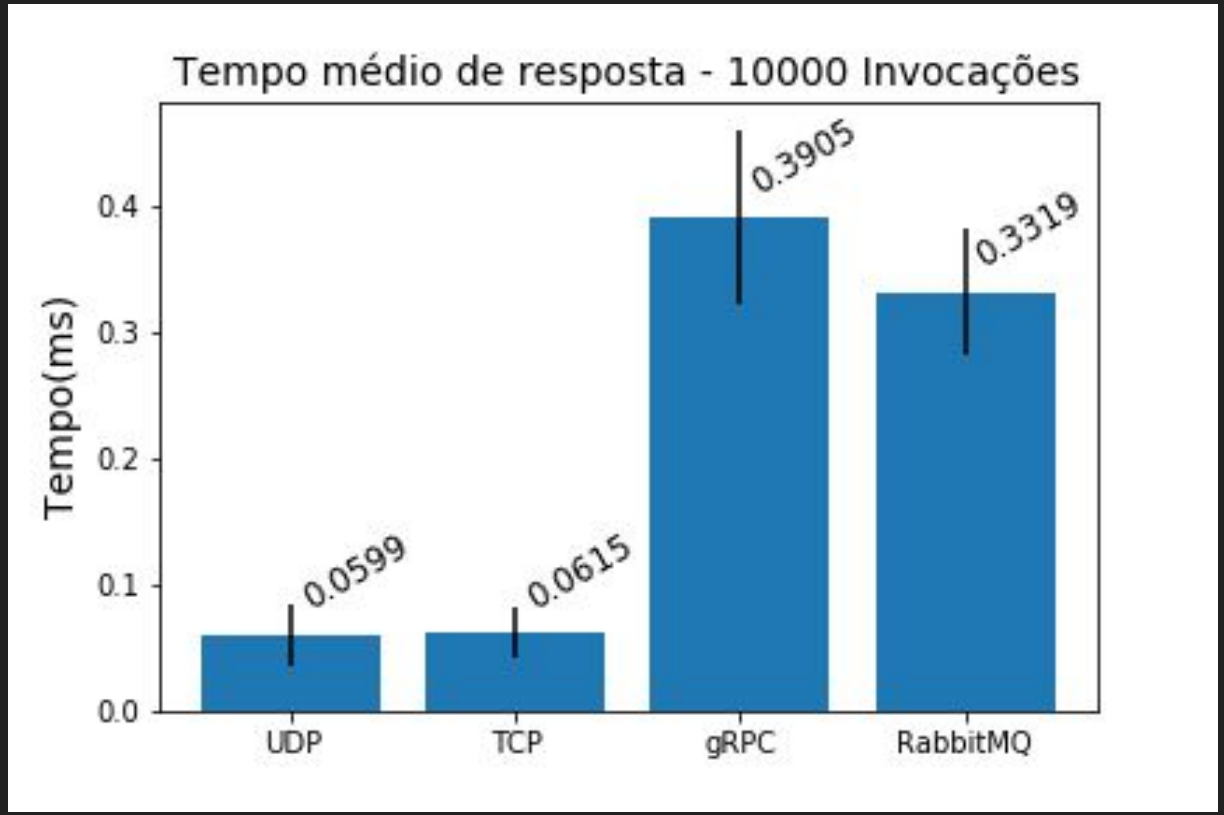
Script de Teste

```
11  # Remove files before starts
12  (find ~/Output/$grpc -type f -delete)
13  (find ~/Output/$rab -type f -delete)
14
15  # Update/Create docker image before starts
16  (docker build --rm --tag=perf-image:v2 .)
17
18  #GRPC x30
19  current='grpc/docker-compose.yml'
20  for i in {1..30}
21  do
22      (docker-compose -f $current up -d)
23      (sleep 10s) #wait writes everything in file
24      (docker-compose -f $current down)
25  done
26  echo "GRPC Done"
27
28  #RABBIT x30
29  current='rabbitmq/docker-compose.yml'
30  for i in {1..30}
31  do
32      (docker-compose -f $current up -d)
33      (sleep 10s) #wait writes everything in file
34      (docker-compose -f $current down)
35  done
36  echo "RABBITMQ Done"
```

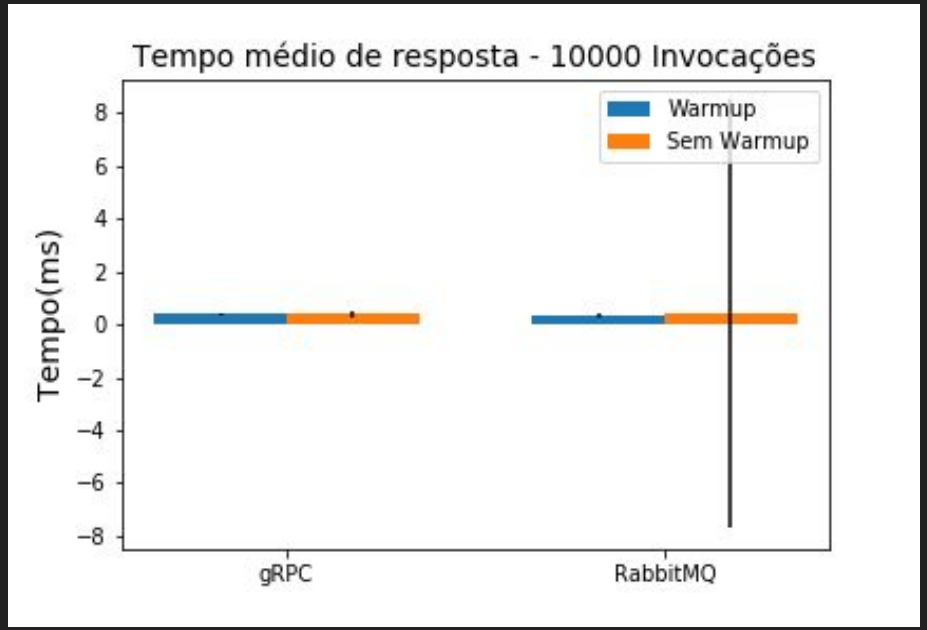
Resultados

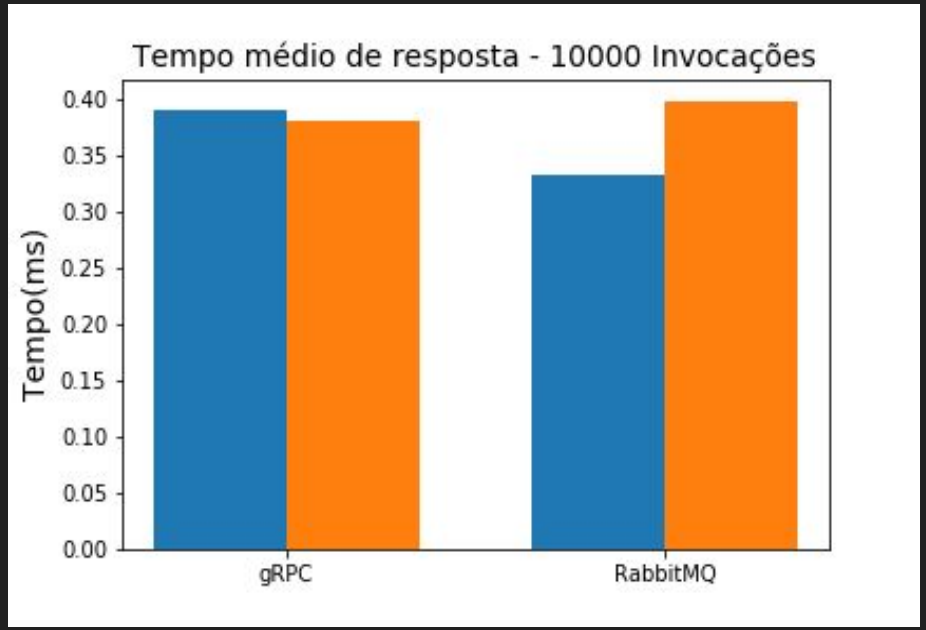

Tempo médio de resposta - 10000 Invocações
Tempo(ms)
0.0
0.1
0.2
0.3
0.4
0.3905
0.3319
gRPC
RabbitMQ

Resultados


Tempo médio de resposta - 10000 Invocações
Tempo(ms)
0.0
0.1
0.2
0.3
0.4
0.0599
0.0615
0.3905
0.3319
UDP
TCP
gRPC
RabbitMQ

Resultados


Tempo médio de resposta - 10000 Invocações
Warmup
Sem Warmup
Tempo(ms)
8
6
4
2
0
−2
−4
−6
−8
gRPC
RabbitMQ


Tempo médio de resposta - 10000 Invocações
Tempo(ms)
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
gRPC
RabbitMQ