Exercício 02

Cliente/servidor utilizando middlewares RPC e MOM

Grupo: Matheus Viana
Natália Soares

Roteiro

- Aplicação
  - com middleware baseado em RPC (MOO)
  - com middleware orientado a mensagem (MOM)
- Avaliação comparativa de desempenho
  - metodologia
  - resultados
- Conclusão

Aplicação

- Cliente envia o número do request e o servidor retorna uma string com a resposta para o request;
- Middlewares utilizados:
  - RPC: **RPC-TCP**
  - MOM: **RabbitMQ**

Aplicação :: Servidor RPC (MOO)

```go
12  type RequestBank struct {}
13
14  type Request struct {
15      Header string
16      RequestNumber int
17  }
18
19  func (t *RequestBank) ReceiveMessage(req *Request, reply *string) (error) {
20      *reply = "Here is your answer for the request " + strconv.Itoa(req.RequestNumber) + "!"
21      return nil
22  }
```

```go
31  func main() {
32      // cria instância do banco
33      bank := new(RequestBank)
34
35      // cria um novo servidor rpc e registra o banco
36      server := rpc.NewServer()
37      server.RegisterName("Bank", bank)
38
39      // Cria um listen rpc-sender
40      l, err := net.Listen("tcp", ":"+strconv.Itoa(SERVER_PORT))
41      checkError(err, "Couldn't create the server")
42
43      // Aguarda por chamadas
44      log.Println("Server is ready (RPC TCP) ...")
45      server.Accept(l)
46  }
```

- A instância de “RequestBank” é registrada no serviço de nomes como “Bank”.

Aplicação :: Client RPC (MOO)

```go
func main() {
    var reply string

    // Conectando ao servidor
    client, err := rpc.Dial("tcp", ":"+ strconv.Itoa(SERVER_PORT))
    checkError(err, "The server isn't ready")

    defer client.Close()

    // Invoca request
    for i := 0; i < SAMPLE_SIZE; i++ {

        // Prepara request
        msgRequest := Request{Header:"Request", RequestNumber: i}

        // Invoca request
        err := client.Call("Bank.ReceiveMessage", msgRequest, &reply)
        checkError(err, "Error communicating with server")

        fmt.Println(reply)
        time.Sleep(10 * time.Millisecond)
    }
}
```

```go
type Request struct {
    Header string
    RequestNumber int
}
```

- O cliente chama remotamente o método “ReceiveMessage” do objeto remoto Bank.

Aplicação :: Servidor MOM

```go
func main() {
    // Conecta ao servidor de mensageria
    connection, err := amqp.Dial("amqp://guest:guest@localhost:5672/")
    checkError(err,"Não foi possível se conectar ao servidor de mensageria")
    defer connection.Close()

    // Cria o canal
    ch, err := connection.Channel()
    checkError(err,"Não foi possível estabelecer um canal de comunicação com o servidor de mensageria")
    defer ch.Close()

    // Declaração de filas
    requestQueue, err := ch.QueueDeclare("request", false, false, false,
        false, nil, )
    checkError(err,"Não foi possível criar a fila no servidor de mensageria")

    replyQueue, err := ch.QueueDeclare("response", false, false, false,
        false, nil, )
    checkError(err,"Não foi possível criar a fila no servidor de mensageria")

    // Prepara o recebimento de mensagens do cliente
    msgsFromClient, err := ch.Consume(requestQueue.Name, "", true, false,
        false, false, nil, )
    checkError(err,"Falha ao registrar o consumidor do servidor de mensageria")

    log.Println("Servidor pronto...")
```

Inicialização do servidor

```go
    for d := range msgsFromClient {
        // Recebe request
        msgRequest := Request{}
        err := json.Unmarshal(d.Body, &msgRequest)
        checkError(err,"Falha ao desserializar a mensagem")

        // Processa request
        r := "Here is your answer for the request " + strconv.Itoa(msgRequest.RequestNumber) + "!"

        // Prepara resposta
        replyMsg := r
        replyMsgBytes, err := json.Marshal(replyMsg)
        checkError(err,"Não foi possível criar a fila no servidor de mensageria")
        if err != nil {
            log.Fatalf("%s: %s", "Falha ao serializar mensagem", err)
        }

        // Publica resposta
        err = ch.Publish("", replyQueue.Name, false, false,
            amqp.Publishing{ContentType: "text/plain", Body: []byte(replyMsgBytes),})
        checkError(err,"Falha ao enviar a mensagem para o servidor de mensageria")
    }
```

Tratamento da requisição

Aplicação :: Client MOM

```go
func main() {
    // Conecta ao servidor de mensageria
    connection, err := amqp.Dial("amqp://guest:guest@localhost:5672/")
    checkError(err,"Não foi possível se conectar ao servidor de mensageria")
    defer connection.Close()

    // Cria o canal
    ch, err := connection.Channel()
    checkError(err,"Não foi possível estabelecer um canal de comunicação com o servidor de mensageria")
    defer ch.Close()

    // Declara as filas
    requestQueue, err := ch.QueueDeclare(
        "request", false, false, false, false, nil, )
    checkError(err,"Não foi possível criar a fila no servidor de mensageria")

    replyQueue, err := ch.QueueDeclare(
        "response", false, false, false, false, nil, )
    checkError(err,"Não foi possível criar a fila no servidor de mensageria")

    // Cria consumidor
    msgsFromServer, err := ch.Consume(replyQueue.Name, "", true, false,
        false, false, nil, )
    checkError(err,"Falha ao registrar o consumidor servidor de mensageria")
```

Inicialização do cliente

```go
    for i := 0; i < SAMPLE_SIZE; i++{
        // prepara request
        msgRequest := Request{Header:"Request",RequestNumber:i}
        msgRequestBytes,err := json.Marshal(msgRequest)
        checkError(err,"Falha ao serializar a mensagem")

        // publica request
        err = ch.Publish("", requestQueue.Name, false, false,
            amqp.Publishing{ContentType: "text/plain", Body: msgRequestBytes,})
        checkError(err,"Falha ao enviar a mensagem para o servidor de mensageria")

        // recebe resposta
        // msgRet <- msgsFromServer
        <-msgsFromServer

        // fmt.Println(string(msgRet.Body))

        time.Sleep(10 * time.Millisecond)
    }
```

Envio da requisição e recebimento da resposta

Análise Comparativa :: Metodologia

- **Objetivo:** comparar os Round-Trip Time (RTT) da implementação com o middleware baseado em RPC e um MOM;
- **Técnica:** Medição;
- **Métrica:** Tempo do cliente enviar uma requisição e receber a resposta do servidor (RTT de uma invocação medida em 1 cliente);

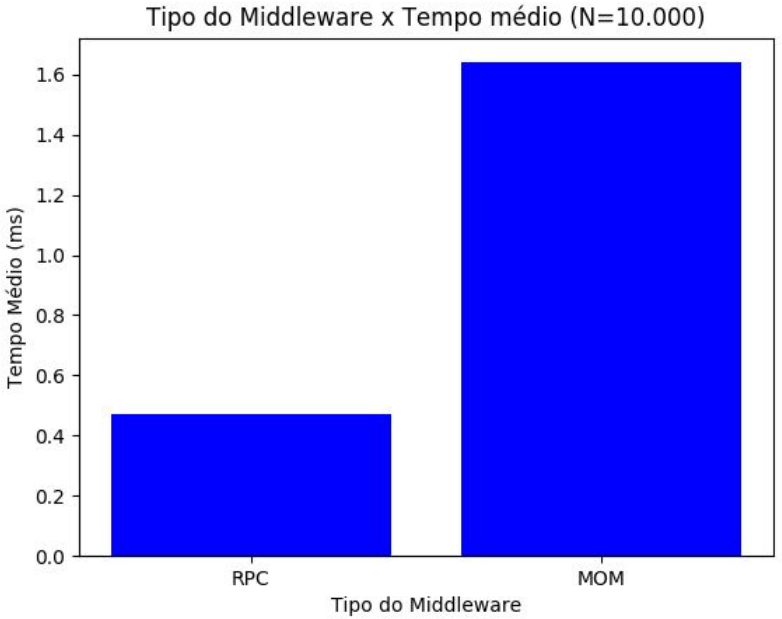
Análise Comparativa :: Metodologia

- **Experimento:**
  - 10.000 requisições sucessivas ao servidor
  - tempo entre requisições de 10ms
  - com 1,2,3,4 e 5 clientes (Parâmetro de Carga)
- **Teste Estatístico:** Mann-Whitney U
  - distribuição não normal
  - variável independente (tipo do middleware)

Análise Comparativa :: Resultados

1 Cliente

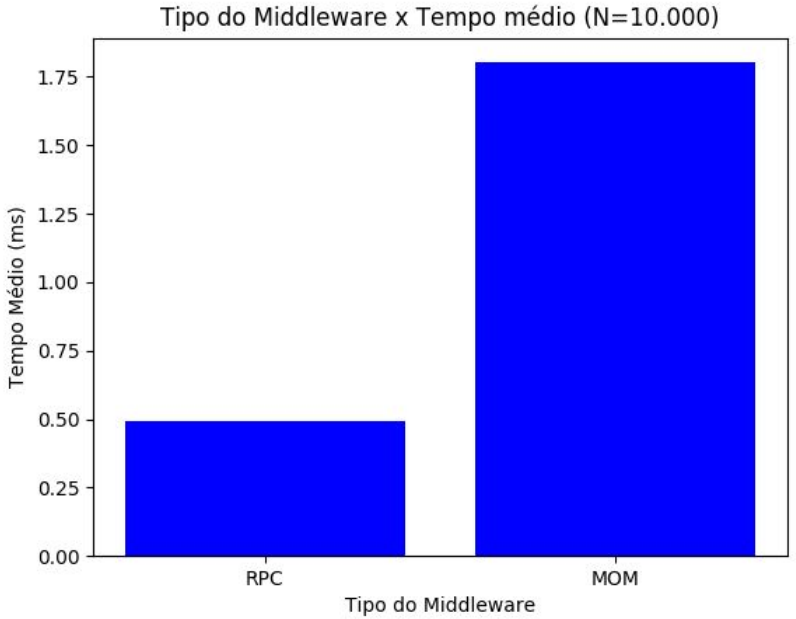
- p-Value: 0.0
- RPC:
  - Média: 0.47 ms
  - Desvio Padrão: 0.15 ms
- MOM:
  - Média: 1.64 ms
  - Desvio Padrão: 0.32 ms



Análise Comparativa :: Resultados

2 Clientes

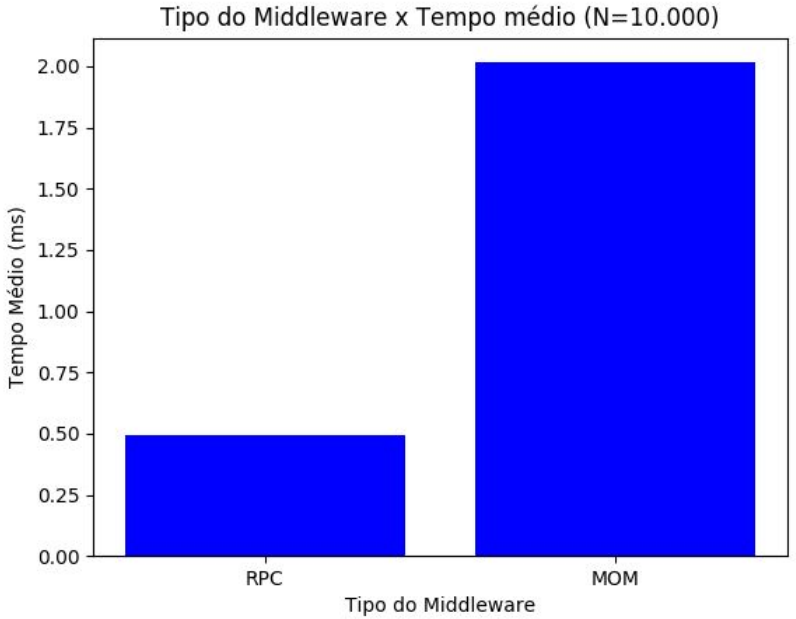
- p-Value: 0.0
- RPC:
  - Média: 0.49 ms
  - Desvio Padrão: 0.11 ms
- MOM:
  - Média: 1.80 ms
  - Desvio Padrão: 0.34 ms



Análise Comparativa :: Resultados

3 Clientes

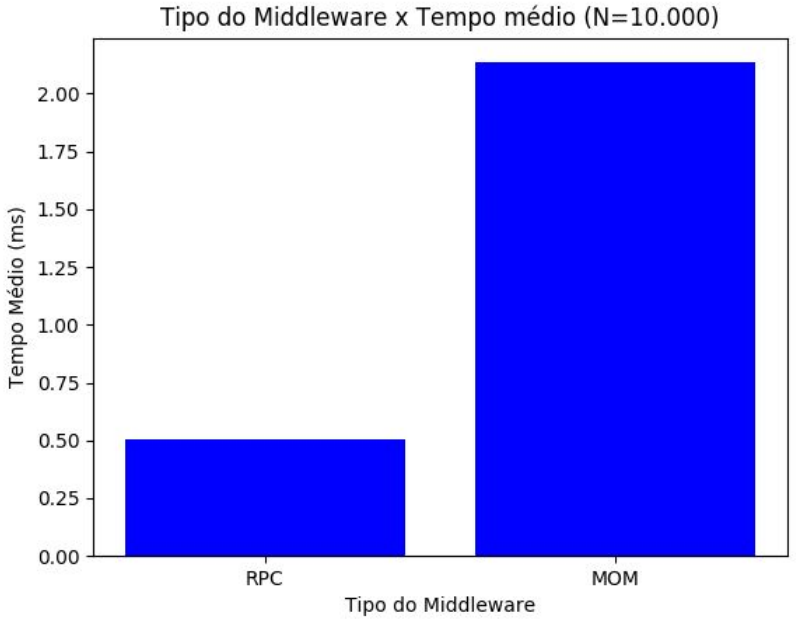
- p-Value: 0.0
- RPC:
  - Média: 0.48 ms
  - Desvio Padrão: 0.15 ms
- MOM:
  - Média: 2.02 ms
  - Desvio Padrão: 0.45 ms



Análise Comparativa :: Resultados

4 Clientes

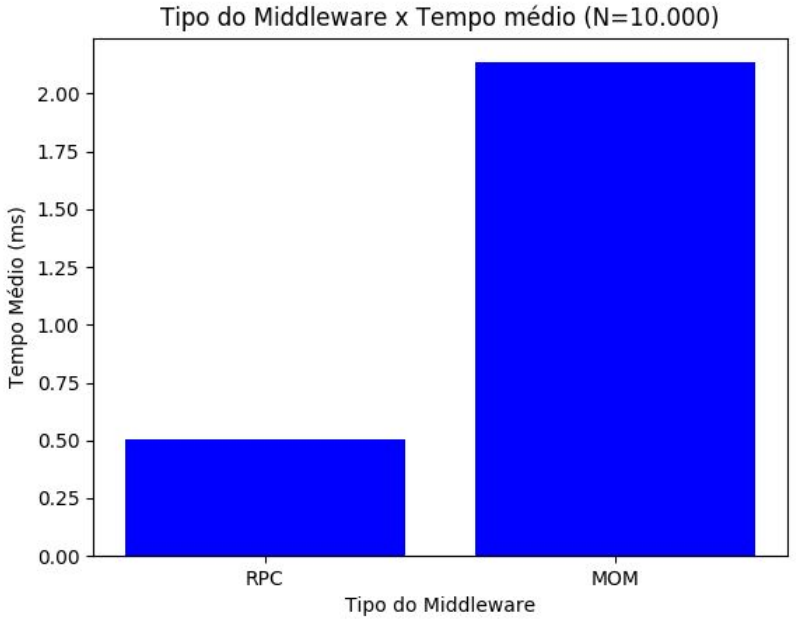
- p-Value: 0.0
- RPC:
  - Média: 0.51 ms
  - Desvio Padrão: 0.16 ms
- MOM:
  - Média: 2.14 ms
  - Desvio Padrão: 0.63 ms



Análise Comparativa :: Resultados

5 Clientes

- p-Value: 0.0
- RPC:
  - Média: 0.48 ms
  - Desvio Padrão: 0.20 ms
- MOM:
  - Média: 2.11 ms
  - Desvio Padrão: 0.99 ms



Conclusão

- A hipótese nula do teste foi rejeitada em todos os casos, pois $p < 0.05$, logo, existe diferença significativa entre os dados do RPC e do MOM;
- O middleware baseado em RPC apresentou um melhor desempenho para nas condições analisadas.

-