Exercício 05 - Prog. Concorrente e Distribuída (IF711)

Luís Eduardo Martins Alves
Pedro Nogueira Coutinho
Zenio Angelo

Centro de Informática UFPE
Universidade Federal de Pernambuco

Implementação Fibonacci - MQTT

```
func fibo(strNum string) string {
        n, err := strconv.Atoi(strNu
        if err != nil {
                panic(err)
        }
        ans := 1
        prev := 0
        for i := 1; i < n; i++ {
                temp := ans
                ans = ans + prev
                prev = temp
        }
        return strconv.Itoa(ans)
```

Subscriber.go - MQTT

```go
const MQTTHost = "mqtt://172.17.0.5:1883"
const MQTTTopic = "Fibonacci"
const QoS = 0

type Message struct {
        Msg string `json:"msg"`
        Pid int    `json:"pid"`
}

func main() {
        // Configurar cliente
        opts := MQTT.NewClientOptions()
        opts.AddBroker(MQTTHost)
        opts.SetClientID("subscriber 1")

        // Criar cliente
        client := MQTT.NewClient(opts)
```

```go
// Conectar ao Broker
token := client.Connect()
token.Wait()
if token.Error() != nil {
        panic(token.Error())
}
```

Subscriber.go - MQTT

```
//Subscreve ao tópico e definir handler
token = client.Subscribe(MQTTTopic, QoS, func(c MQTT.Client, m MQTT.Message) {
        var msg Message
        json.Unmarshal(m.Payload(), &msg)
        ans := fmt.Sprintf("Mensagem recebida, o fibo de %s eh %s\n", msg.Msg, fibo(msg.Msg))
        fmt.Printf("Mensagem recebida, o fibo de %s eh %s\n", msg.Msg, fibo(msg.Msg))
```

```
 token := client.Publish(MQTTTopic+"/publisher_"+fmt.Sprint(msg.Pid), QoS, false, ans)
 token.Wait()
 if token.Error() != nil {
         panic(token.Error())
 }
```

Publisher.go - MQTT

```
const MQTTHost = "mqtt://172.17.0.5:1883"
const MQTTTopic = "Fibonacci"
const QoS = 0

type Message struct {
        Msg string `json:"msg"`
        Pid int    `json:"pid"`
}
```

```
var receiveHandler MQTT.MessageHandler = func(c MQTT.Client, m MQTT.Message) {
        total_time = time.Since(start)
        fmt.Println(total_time)
        signal = true
        fmt.Printf("Mensagem recebida, o fibo eh %s\n", m.Payload())
}
```

Publisher.go - MQTT

```go
func main() {
        // config
        clientID := "publisher_" + fmt.Sprint(os.Getpid())
        opts := MQTT.NewClientOptions()
        opts.AddBroker(MQTTHost)
        opts.SetClientID(clientID)

        // criar cliente
        client := MQTT.NewClient(opts)
```

```go
// conectar ao broker
if token := client.Connect(); token.Wait() && token.Error() != nil {
        fmt.Println(token.Error())
        os.Exit(1)
}
```

Publisher.go - MQTT

```
// Subscreve para a resposta
if token := client.Subscribe(MQTTTopic+"/"+clientID, QoS, receiveHandler); token.Wait() && token.Error() != nil {
        fmt.Println(token.Error())
        os.Exit(1)
}
```

```
for i := 0; i < 10000; i++ {
        signal = false
        msg := Message{
                Msg: fmt.Sprint((i + 1) % 50),
                Pid: os.Getpid(),
        }
        jmsg, err := json.Marshal(msg)
```

```
// Publicar a mensagem
token := client.Publish(MQTTTopic, QoS, false, jmsg)
start = time.Now()
token.Wait()
if token.Error() != nil {
        panic(token.Error())
}
```

Implementação Fibonacci - RabbitMQ

```
func fibo(n int) int {
        ans := 1
        prev := 0
        for i := 1; i < n; i++ {
                temp := ans
                ans = ans + prev
                prev = temp
        }
        return ans
}
```

Consumer.go - RabbitMQ

```go
// configurations
const RequestQueue = "request_queue"
const ResponseQueue = "response_queue"

type Request struct {
        Num int
}
```

```go
func main() {
        //connect to broker
        conn, err := amqp.Dial("amqp://guest:guest@172.17.0.3:5672/")
        failOnError(err, "Failed to connect to RabbitMQ")
        defer conn.Close()

        //create channel
        ch, err := conn.Channel()
        failOnError(err, "Failed to open a channel")
        defer ch.Close()
```

Consumer.go - RabbitMQ

```
//create request queue
q, err := ch.QueueDeclare(
        RequestQueue, //routing key(queue's name)
        false,        //durable
        false,        //autodelete
        false,        //exclusive
        false,        //nowait
        nil,          //args
)
```

```
//create request queue's consumer
msgs, err := ch.Consume(
        q.Name, //routing key(queue's name)
        "",     //consumer
        true,   //autoACK
        false,  //exclusive
        false,  //noLocal
        false,  //nowait
        nil,    //args
)
```

Consumer.go - RabbitMQ

```go
//receive and process the messages
for d := range msgs {

        //receive request
        msg := Request{}
        err := json.Unmarshal(d.Body, &msg)
        failOnError(err, "Failed to deserialize the message")

        //process request
        replymsgBytes, err := json.Marshal(fibo(msg.Num))
        fmt.Println(msg.Num, fibo(msg.Num))
        failOnError(err, "Failed to serialize the message")
```

```go
//send(publish) response
err = ch.Publish(
        "",         // exchange
        d.ReplyTo, // routing key
        false,      // mandatory
        false,      // immediate
        amqp.Publishing{
                ContentType:   "text/plain",
                CorrelationId: d.CorrelationId,
                Body:          replymsgBytes,
        })
```

Producer.go - RabbitMQ

```go
// Other configurations
const SampleSize = 10000
const RequestQueue = "request_queue"
const ResponseQueue = "response_queue"

var m runtime.MemStats

type Message struct {
        Num int
}
```

```go
func main() {
        //connect to broker
        conn, err := amqp.Dial("amqp://guest:guest@172.17.0.3:5672/")
        failOnError(err, "Failed to connect to RabbitMQ")
        defer conn.Close()

        //create channel
        ch, err := conn.Channel()
        failOnError(err, "Failed to open a channel")
        defer ch.Close()
```

Producer.go - RabbitMQ

```
//create response queue
replyQueue, err := ch.QueueDeclare(
        ResponseQueue+"/"+strconv.Itoa(os.Getpid()), //routing key(queue's name)
        false, //durable
        false, //autodelete
        true,  //exclusive
        false, //nowait
        nil,   //args
)
```

```
//create response queue's consumer
msgs, err := ch.Consume(
        replyQueue.Name, //routing key(queue's name)
        "",              //consumer
        true,            //autoACK
        false,           //exclusive
        false,           //noLocal
        false,           //nowait
        nil,             //args
)
```

Producer.go - RabbitMQ

```go
//send the message
for i := 0; i < SampleSize; i++ {
        msg := Message{
                Num: i % 50,
        }

        //serialize
        msgBytes, err := json.Marshal(msg)
        //fmt.Println(string(msgBytes), msg)
        failOnError(err, "Failed to serialize the message")

        correlationID := RandomString(32)
```

```go
//publish
err = ch.Publish(
        "",           // exchange
        RequestQueue, // routing key
        false,        // mandatory
        false,        // immediate
        amqp.Publishing{
                ContentType:   "text/plain",
                CorrelationId: correlationID,
                ReplyTo:       replyQueue.Name,
                Body:          msgBytes,
        })
```

Especificações do computador de teste

SO: Ubuntu 18.04.6 LTS;

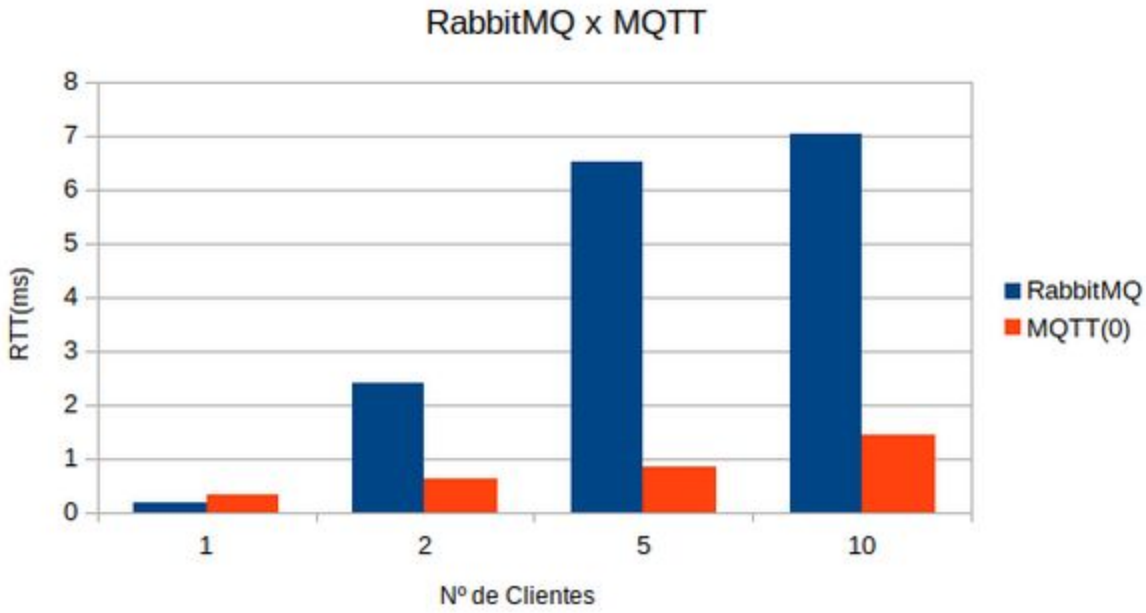
Memória RAM: 7.6 GiB;

Processador: Intel® Core™ i5-7200U CPU @ 2.50GHz × 4 ;

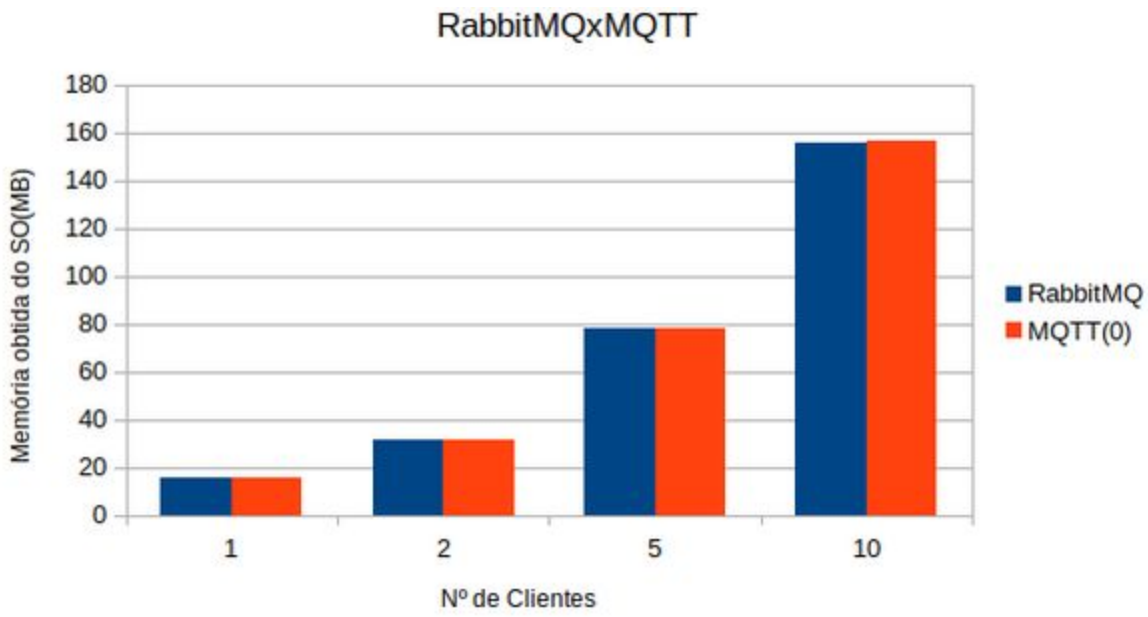
Arquitetura: 64 bits;

Armazenamento: HDD 53.8 GB.

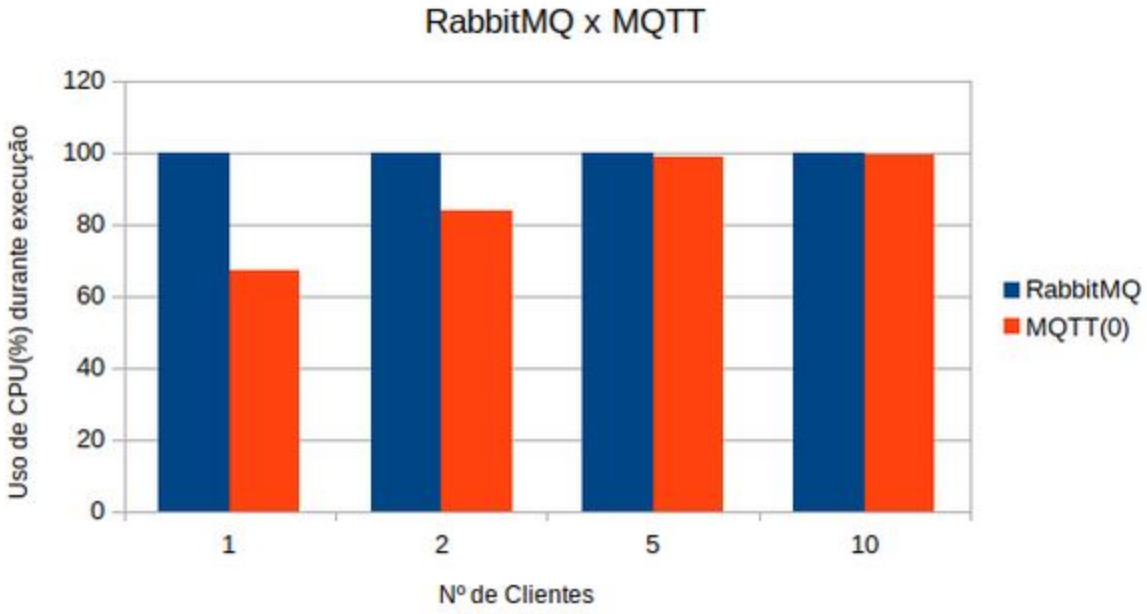
Analisando resultados e diferenças



Analisando resultados e diferenças



Analisando resultados e diferenças



Analisando resultados e diferenças

RTT(ms)				
nº de clientes	1	2	5	10
RabbitMQ	0.1819291977	2.390047955	6.509655146	7.033732521
MQTT(0)	0.329180222	0.6184656108	0.8541543754	1.433392212
MQTT(1)	2.798238503	3.722109346		
MQTT(2)	12.45931567	12.05552071		

Uso de CPU(%)				
nº de clientes	1	2	5	10
RabbitMQ	100	99.99105674	100	99.99680247
MQTT(0)	67.28247914	83.63165749	98.6000163	99.03496065
MQTT(1)	67.58533502	100		
MQTT(2)	100	100		

Memória obtida do SO(MB)				
nº de clientes	1	2	5	10
RabbitMQ	15.42438245	31.04967739	78.15547473	155.6294932
MQTT(0)	15.5233512	31.49223989	78.11556223	156.1545245
MQTT(1)	15.61595745	31.32403364		
MQTT(2)	15.9311637	32.06025239		