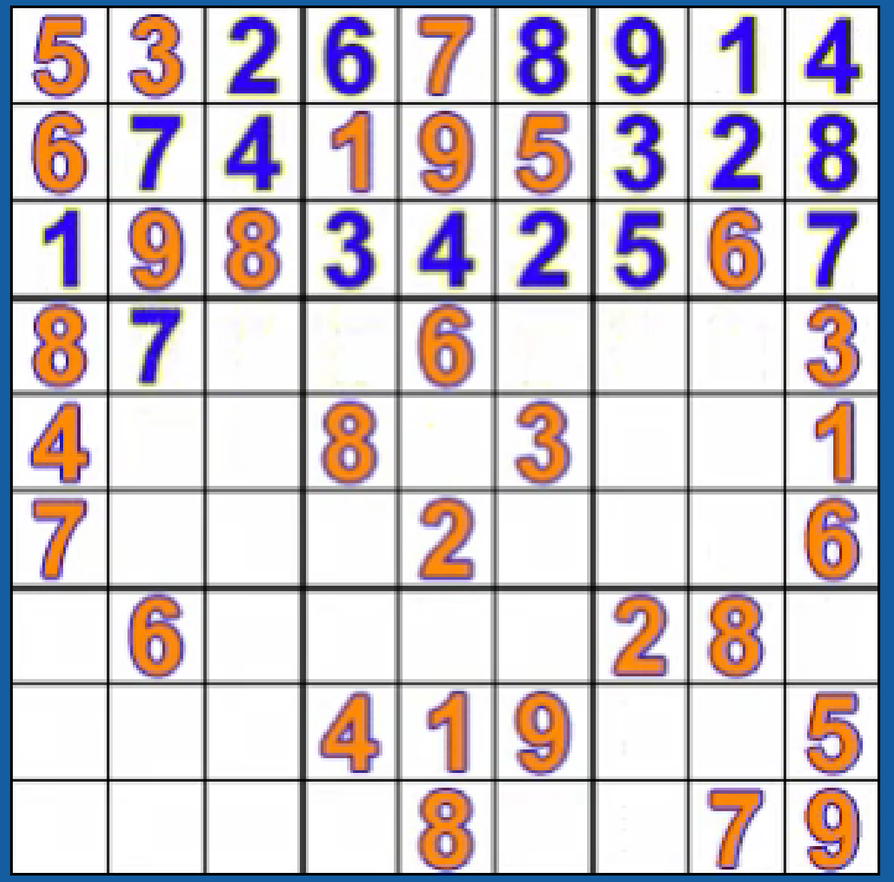
Atividade 5
RabbitMQ

Grupo:
Ekistoclecio Heleno Duarte de Lima
Ian Karlo Torres dos Santos
Samuel Simões de Souza Filho

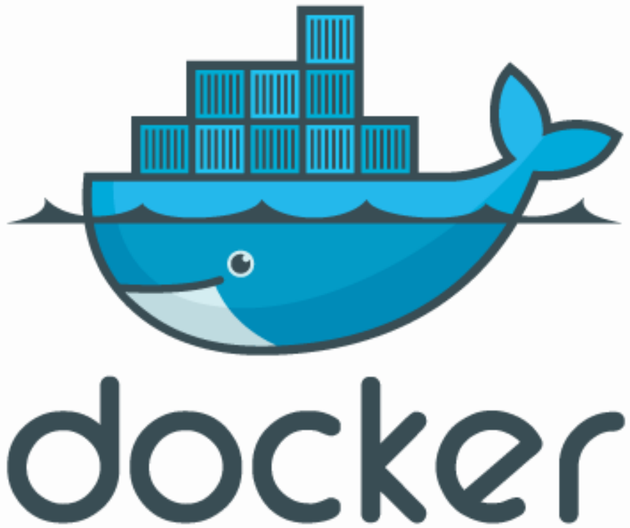
O projeto

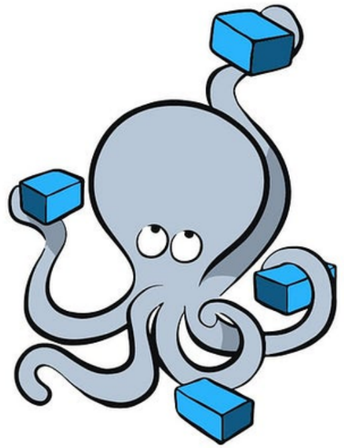
O objetivo é medir o desempenho da conexão cliente e servidor utilizando o RabbitMQ em comparação com uma conexão estabelecida usando o RPC (Remote Procedure Call) feito no exercício anterior. Para isso, foram utilizadas implementações semelhantes de cliente e servidor, e medidos os parâmetros de desempenho associados ao RTT (Round-Trip Time).



Backtracking

Serviço do RabbitMQ





docker
Compose

Serviço do RabbitMQ - Docker Compose

```yaml
version: '3'
services:
  rabbitmq:
    image: rabbitmq:3.13.0
    environment:
      - RABBITMQ_DEFAULT_USER=user
      - RABBITMQ_DEFAULT_PASS=password
    ports:
      - 5672:5672
```

Implementação - Servidor

```
conn, err := amqp.Dial(common.GetConnectionString())
common.HandleError(err, "Unable to connect to broker")

defer conn.Close()

ch, err := conn.Channel()
common.HandleError(err, "Unable to establish a communication channel with the broker")

defer ch.Close()

q, err := ch.QueueDeclare(…
common.HandleError(err, "Unable to create queue in broker")

msgs, err := ch.Consume(…
common.HandleError(err, "Failed to register the consumer with the broker")
```

Implementação - Servidor

```go
for data := range msgs {
    msg := common.Request{}
    err := json.Unmarshal(data.Body, &msg)
    common.HandleError(err, "Failed to deserialize message")

    solver := impl.SudokuSolver{}

    if msg.ShouldTurnOff {
        fmt.Println("Shutting Down Server")
        os.Exit(0)
    }

    r := solver.Run(msg)

    replyMsg := common.Reply{R: r}
    replyMsgBytes, err := json.Marshal(replyMsg)
    common.HandleError(err, "Failed to serialize message")
```

```go
err = ch.Publish(
    "",
    data.ReplyTo,
    false,
    false,
    amqp.Publishing{
        ContentType:   "text/plain",
        CorrelationId: data.CorrelationId,
        Body:          replyMsgBytes,
    },
)
common.HandleError(err, "Failed to send the message to the broker
```

Solucionador

```go
type SudokuSolver struct{}

func (s *SudokuSolver) Run(req common.Request) [][]int {

    board := req.Board

    numRoutines := 5

    matrizChannel := make(chan [][]int)
    signalChannel := make(chan int, numRoutines)

    // fmt.Println("-----------", request)
    rand.Seed(42)
    for i := 0; i < numRoutines; i++ {
        go Solve(&matrizChannel, &signalChannel, board, i)
    }

    matrix := <-matrizChannel

    // PrintBoard(matrix, 9)

    return matrix
}
```

Implementação - Cliente

```
conn, err := amqp.Dial(common.GetConnectionSting())
common.HandleError(err, "Unable to connect to messaging server")
defer conn.Close()

ch, err := conn.Channel()
common.HandleError(err, "Unable to establish a communication channel with the messaging server")
defer ch.Close()

replyQueue, err := ch.QueueDeclare(…

common.HandleError(err, "Unable to connect to replyQueue")

msgs, err := ch.Consume(…
common.HandleError(err, "Failed to register the server with the broker")

times := make([]float64, 0)

board, err := utils.GetBoard(boardNumber)
common.HandleError(err, "Unable to get a board")
```

Implementação - Cliente

```go
for i := 0; i < iterations; i++ {
    startTime := time.Now()

    // prepara mensagem
    msgRequest := common.Request{Board: board, ShouldTurnOff: false}
    msgRequestBytes, err := json.Marshal(msgRequest)
    common.HandleError(err, "Failed to serialize message")

    correlationID := common.RandomId(32)

    err = ch.Publish(
        "",
        common.RequestQueue,
        false,
        false,
        amqp.Publishing{
            ContentType:   "text/plain",
            CorrelationId: correlationID,
            ReplyTo:       replyQueue.Name,
            Body:          msgRequestBytes,
        },
    )

    common.HandleError(err, "Failed to publish message")
```

```go
m := <-msgs

msgResponse := common.Reply{}
err = json.Unmarshal(m.Body, &msgResponse)
common.HandleError(err, "Error deserializing the response")

// common.PrintBoard(msgResponse.R, 9)

duration := time.Since(startTime)

times = append(times, float64(duration.Microseconds()))
```

Análise de Desempenho - Especificações do Sistema

- Processador: Intel(R) Core (TM) i7-1165G7 @ 2.80GHz, 4 núcleos, 8 processadores lógicos
- Memória: 16GB
- Sistema operacional: Linux Mint 21.1
- Linguagem de programação: Go (Versão 1.21.1)
- Fonte de alimentação: Rede elétrica
- Processos em execução: Processos essenciais do sistema.

Análise de Desempenho - Métrica

A métrica utilizada é o tempo entre o envio de uma chamada pelo cliente ao servidor e o recebimento da resposta para essa chamada. **Após estabeler uma conexão TCP com o servidor, é estabelecida uma conexão AMQP e criados os canais de envio e recebimento de mensagens.** O cliente faz chamadas em sequência, enviando um tabuleiro de sudoku **em um canal** e esperando que o servidor o responda com o tabuleiro resolvido **em outro canal.**

Técnica de avaliação

Foi utilizado um par de aplicações cliente e servidor **se comunicando utilizando o RabbitMQ.** São feitos dez mil envios em sequência, cada um contendo um array que descreve um tabuleiro de sudoku. Sempre realizando um novo envio somente após receber a resposta do anterior.

Análise de Desempenho - Carga

3		6	5		8	4		
5	2							
	8	7					3	1
		3		1			8	
9			8	6	3			5
	5			9		6		
1	3					2	5	
							7	4
		5	2		6	3		

					5		6	
		5		6				
	7					4		
	8		1				2	
	3			7				
			5		2	1		3
		6			1		4	
		2					7	6
		3			4	5		1

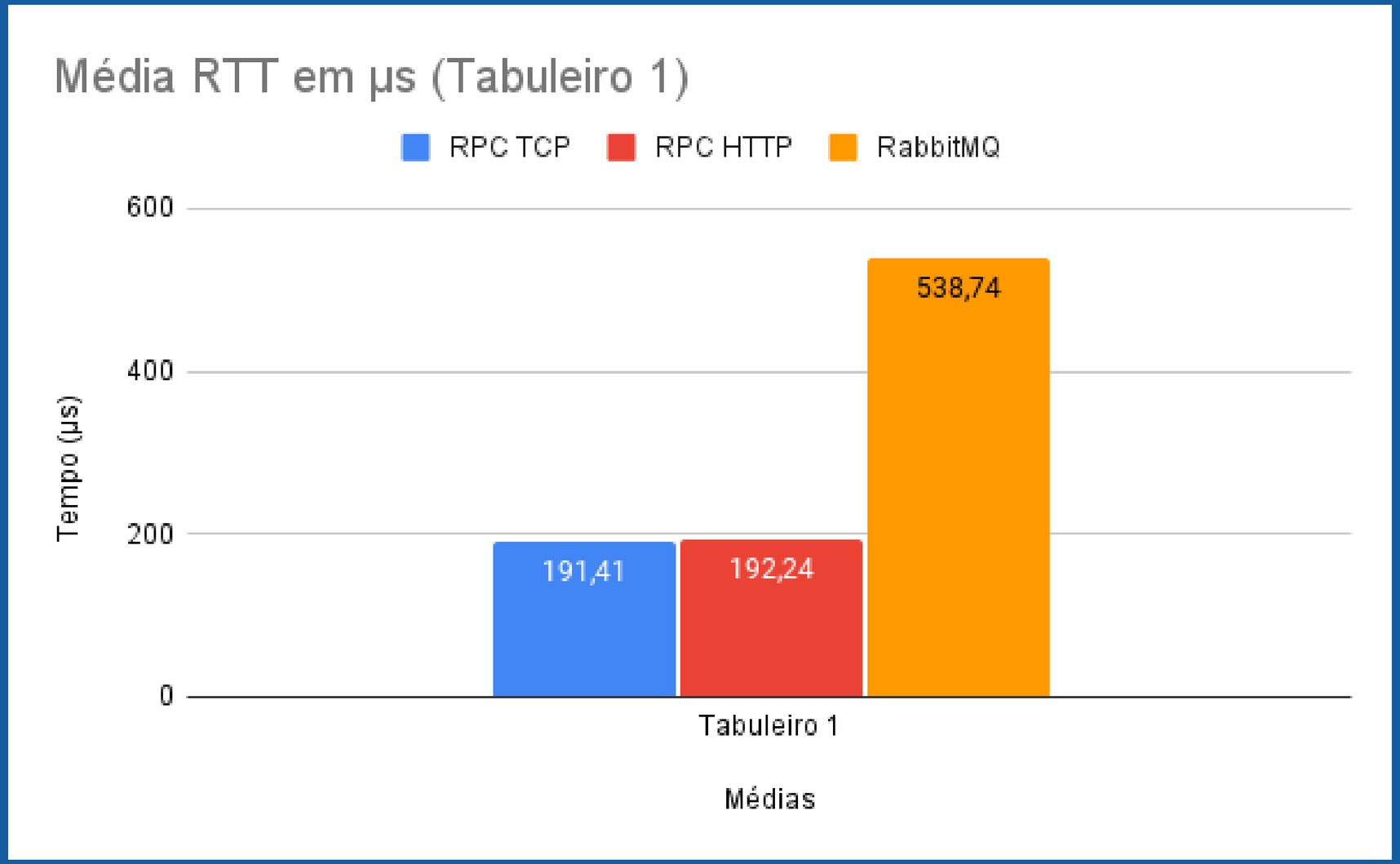
	9		2					
						5		
4	8			5		6	9	2
					9			
			4	2			8	
			8		7			5
	6	1	7		5	9		4
	4			6			5	1
2			1					6

Para evitar que configurações diferentes de tabuleiro tivessem impacto nos testes foram utilizados 3 modelos diferentes para o processo de análise.

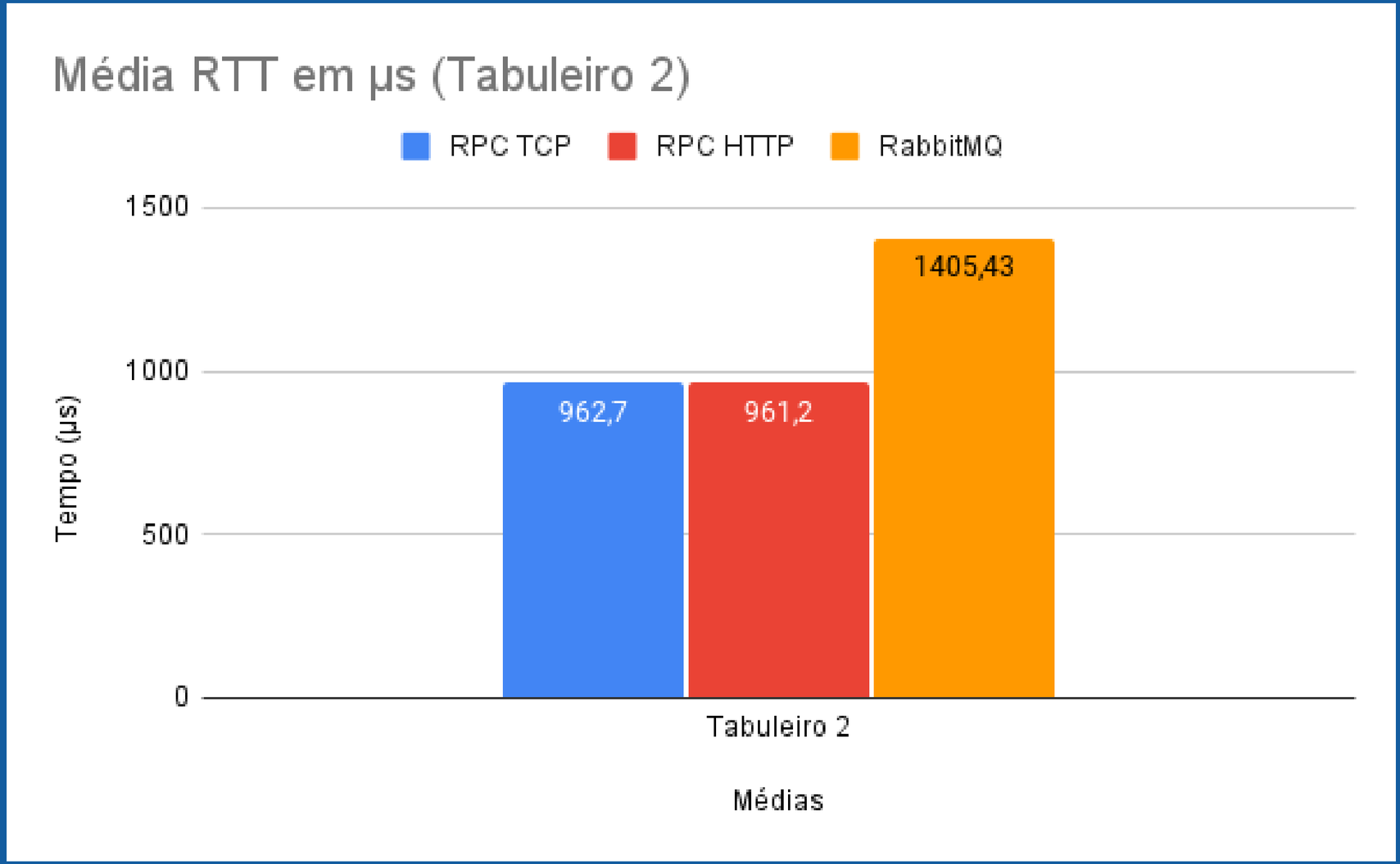
Análise

Análise estatística dos dados obtidos

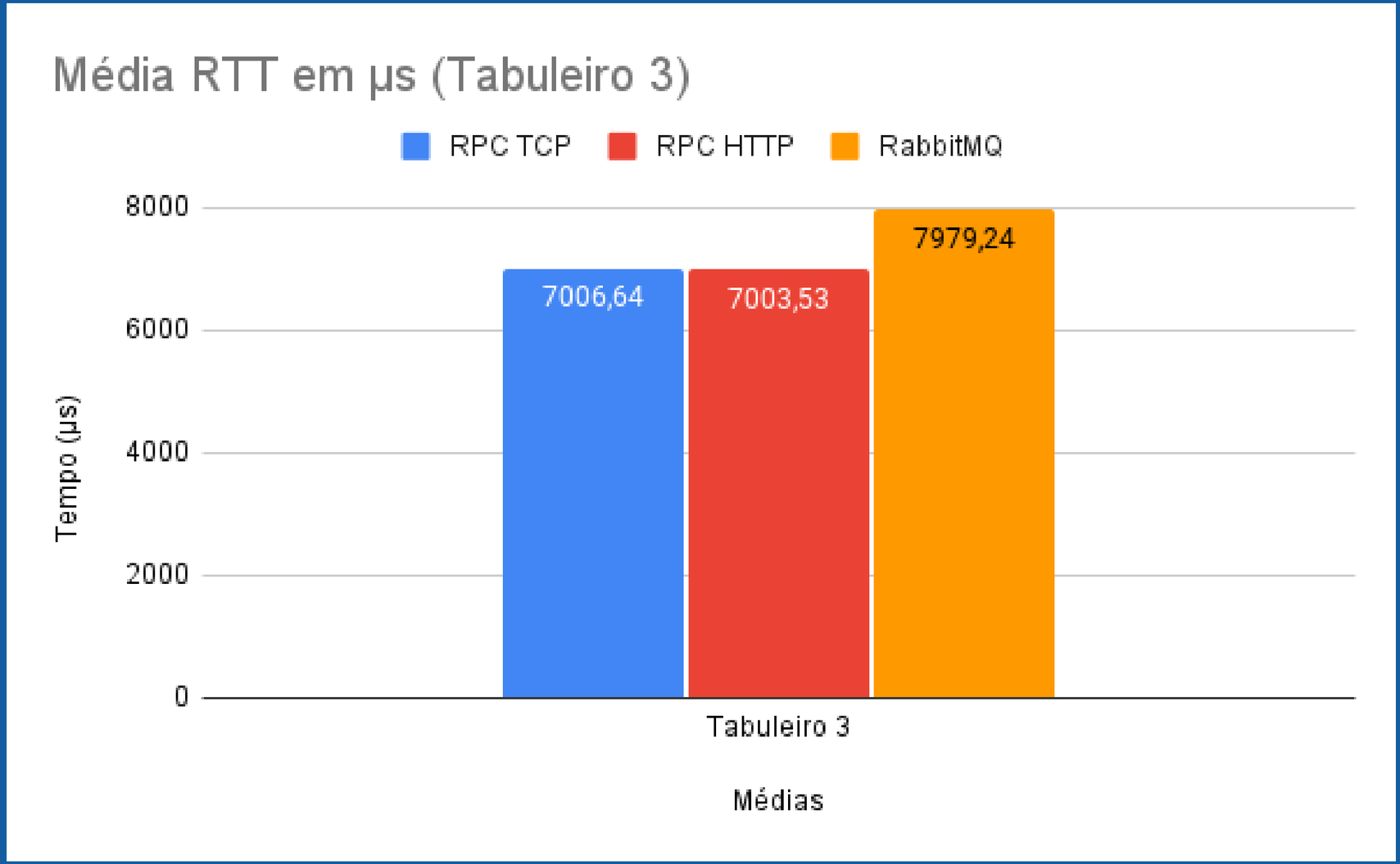
Análise de Desempenho (Tabuleiro 1)



Análise de Desempenho (Tabuleiro 2)



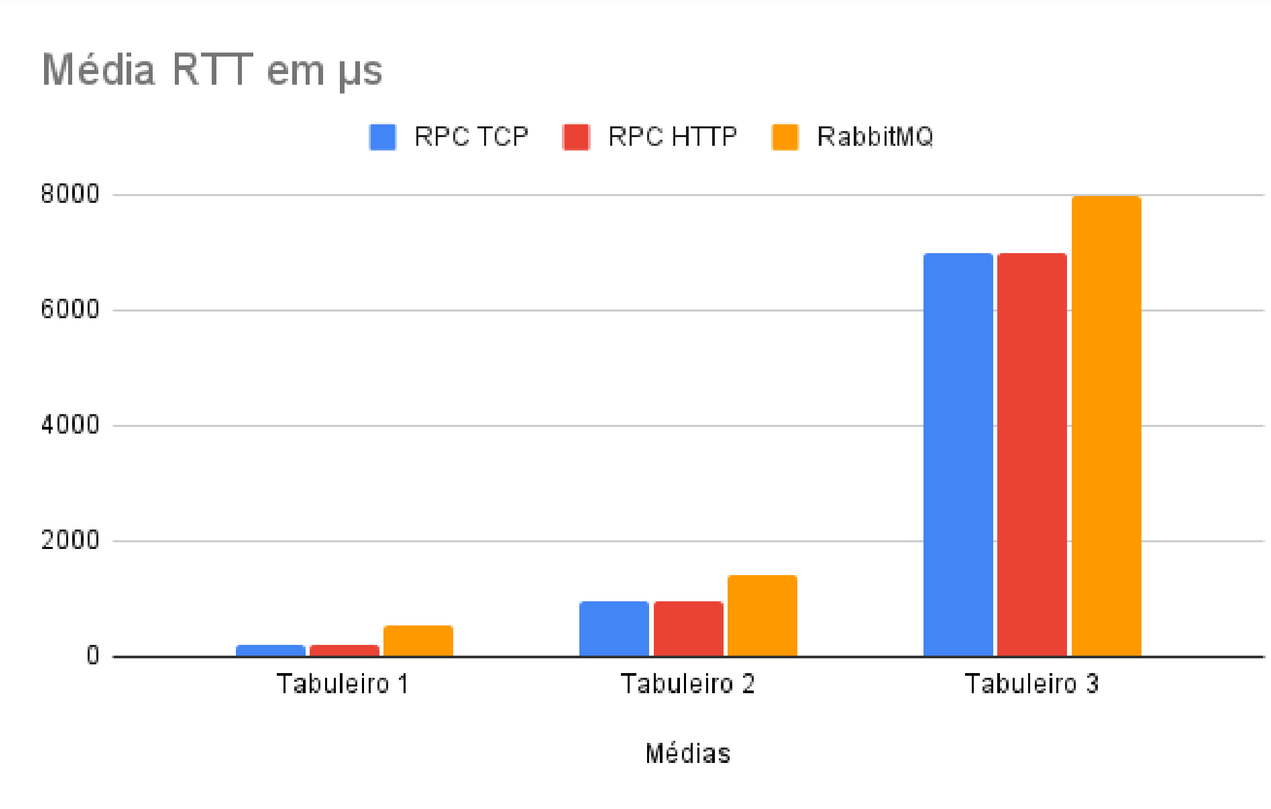
Análise de Desempenho (Tabuleiro 3)



Resultados

Partindo das análises realizadas, é possível concluir que, devido aos processamentos adicionais envolvendo a conexão RabbitMQ o tempo médio RTT teve um aumento consideravel, esse aumento fica evidente nas configurações mais simples do tabuleiro onde o tempo de resolução é menor e a maior parte do tempo RTT é consequencia da conexão RabbitMQ.
Também é possível perceber que a influência do RabbitMQ, vai ficando menos relevante conforme a complexidade do tauleiro aumenta, pois o custo da operação aumenta também.

Comparação de desempenho das aplicações RabbitMq e GoRPC para cada tabuleiro



Obrigado