UNIVERSIDADE DE SÃO PAULO

ESCOLA DE ARTES, CIÊNCIAS E HUMANIDADES

Relatório Exercício Programa

Remote Procedure Calls (RPC)

Gustavo Jun Nagatomo - 12542571

Edgar Henrique de Oliveira Lira - 12717266

São Paulo

2023

SUMÁRIO

1. Especificações

1.1 Infraestrutura

Neste trabalho, foram utilizados na comunicação duas máquinas do tipo PC (Personal Computer) dos alunos denominadas como **máquina A** e **máquina B**. A máquina A utiliza o Ubuntu 22.10, enquanto a máquina B utiliza o Fedora 37.

1.2 Código

Já em termos do código propriamente dito, temos:

	Google Remote Procedure Call (gRPC)	**Java Remote Method Invocation (JRMI)**
Linguagem de Programação	python 3.11.3	openjdk 20.0.1 (2023-04-18)
Bibliotecas Adicionais	pip 23.1.2 (apenas para instalação de pacotes) grpcio 1.54.2 grpcio-tools 1.54.2	nenhuma

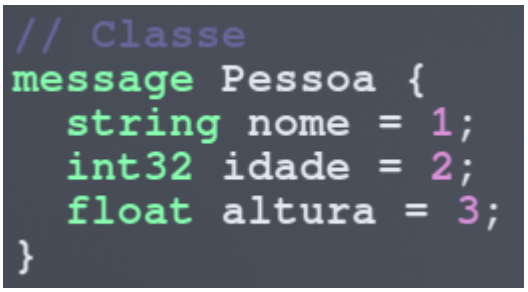
2. Testes das Operações

1.4. Comparação de Tempos de Chamada

As operações básicas escolhidas para comparação de tempo foram:

- operação sem argumentos e sem valor de retorno (void)
- operação com um argumento long e valor de retorno long
- operação com oito argumentos long e valor de retorno long

- operação com um argumento String e valor de retorno String, para strings de tamanhos 1, 2, 4, …, 2048
- operação com um argumento e um valor de retorno da classe básica a seguir:



Fonte: *ep-dsid/gRpc/protos/greet.proto (linha: 65)*

Nas comparações locais, o cliente e servidor foram executados na máquina A para o gRPC, e na máquina B para o jrmi.

Todas as execuções foram feitas em servidores não viciados, isto é, cuja primeira e única conexão foi o cliente e as únicas execuções foram dos experimentos.

É importante ressaltar que o 1° teste de algumas operações contém um atraso maior devido à estruturação da conexão ao servidor. Portanto, quando isso ocorreu, retiramos-os do cálculo da média (que passou a conter apenas os 9 *data points* restantes).

2.1 gRPC

2.1.1 Como executar

Dentro da pasta do EP existem duas pastas: jrmi grpc, entre na pasta do grpc.

Ao abrir a pasta do gRpc, haverá 3 pastas:

1. protos
2. servidor
3. cliente

Não há necessidade de gerar os arquivos de configuração novamente, mas caso tenha interesse em gerar o arquivo de configuração, na pasta protos há um arquivo indicando o comando para gerar o arquivo de configuração, mas lembre-se

de estar no nível de diretório do gRPC e não dentro da pasta do protos para executar o comando.

Para executar o cliente e servidor do gRpc:

Passo 1: Abra 2 terminais, em um deles entre na pasta servidor e execute o arquivo da seguinte forma:

$ python servidor.py

Após executar o comando é solicitada a porta usada na comunicação, digite a porta do seu interesse.

Passo 2: No segundo terminal entre na pasta cliente e execute o comando:

$ python cliente.py

Após executar o comando é solicitado o IP e a porta, basta digitar a porta que foi utilizada no servidor e o IP do servidor

Passo 3: Utilizar a interface interativa.

* Caso queira gerar os tempos de execução, basta executar o comando

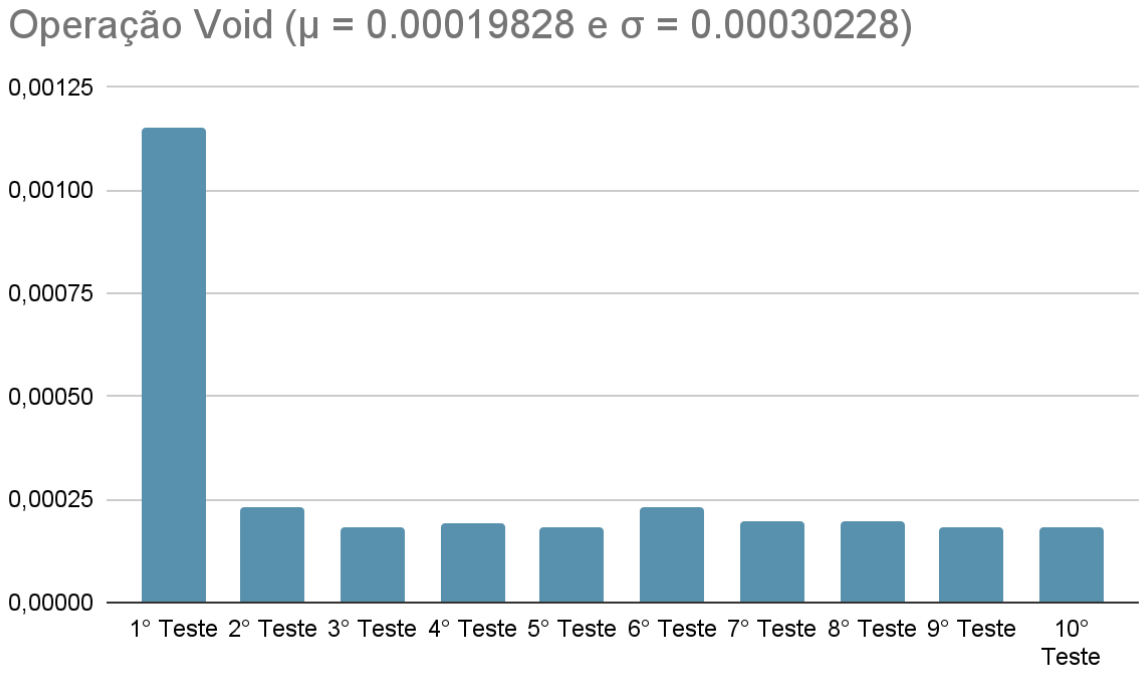
$ python cliente_geraTxt.py

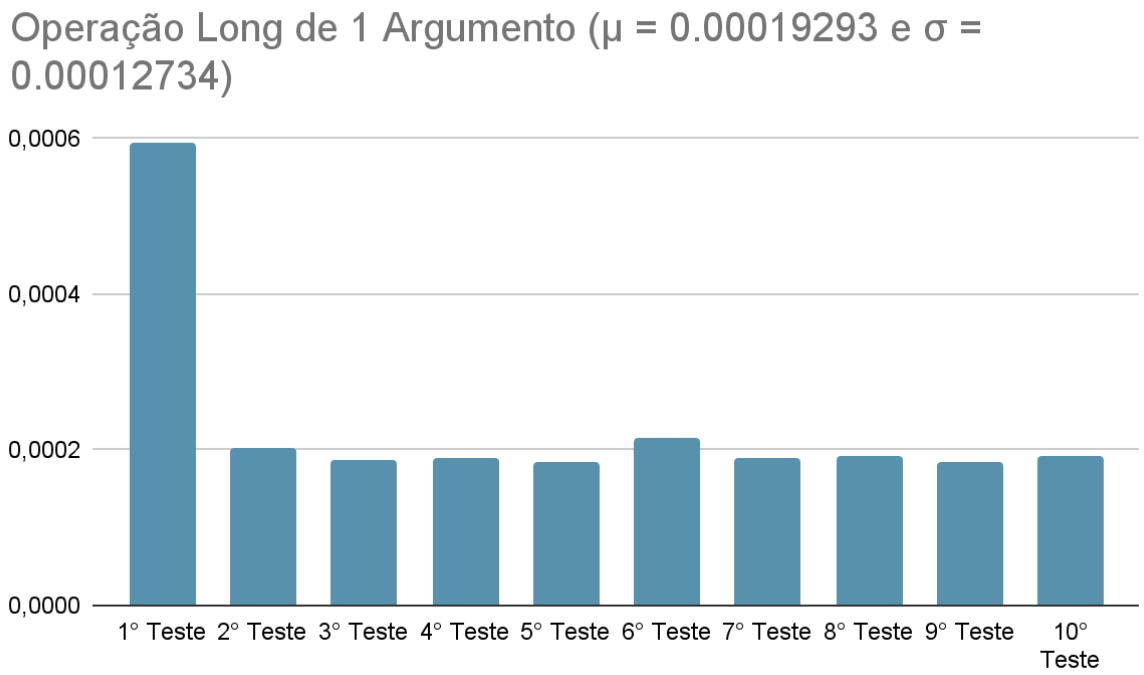
Após a execução deste comando, digite o IP e portas desejadas e os tempos estarão na pasta “tempos”.

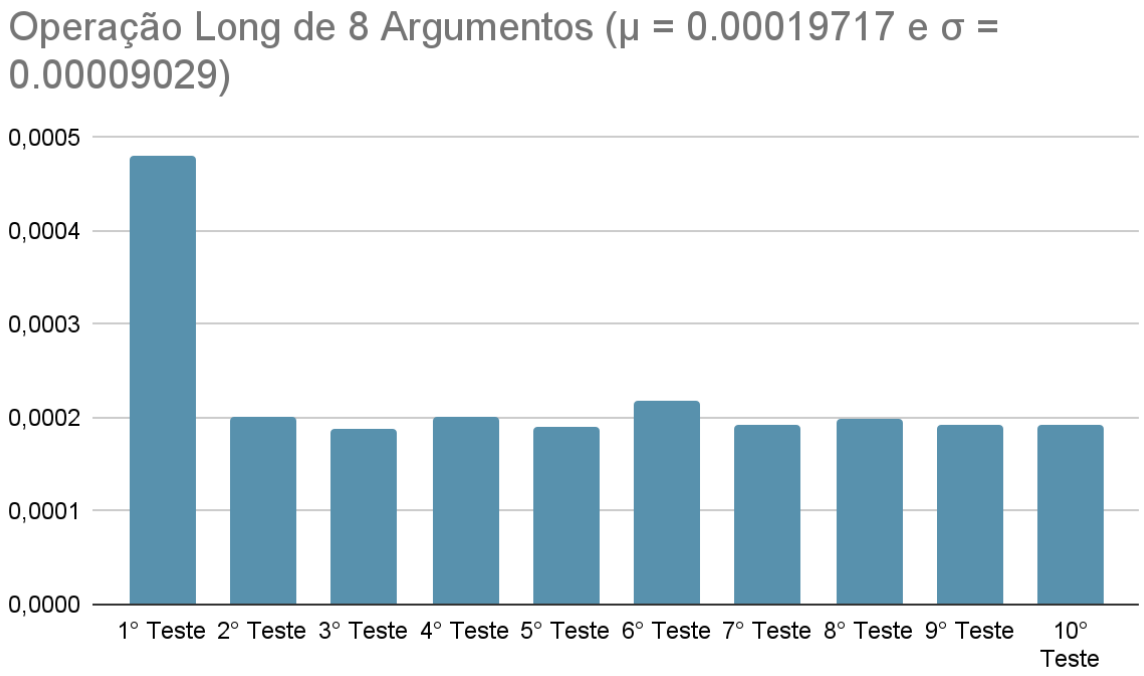
2.1.2 gRPC Local

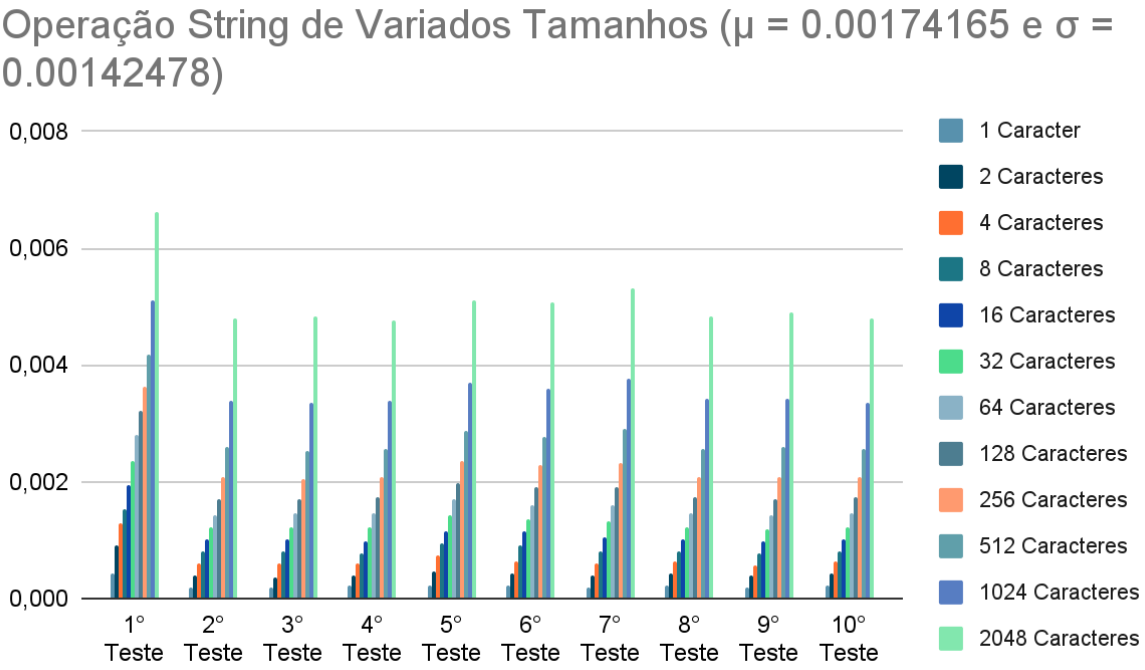
Para o gRPC, foram criados dois programas: um que nos permite testes individuais de cada uma das operações e outro que testa realiza os experimentos 5 vezes, guarda os resultados parciais e também guarda as médias finais dos tempos de cada experimento (todos em arquivos .txt na pasta *ep-dsid/gRpc/cliente/tempos*).

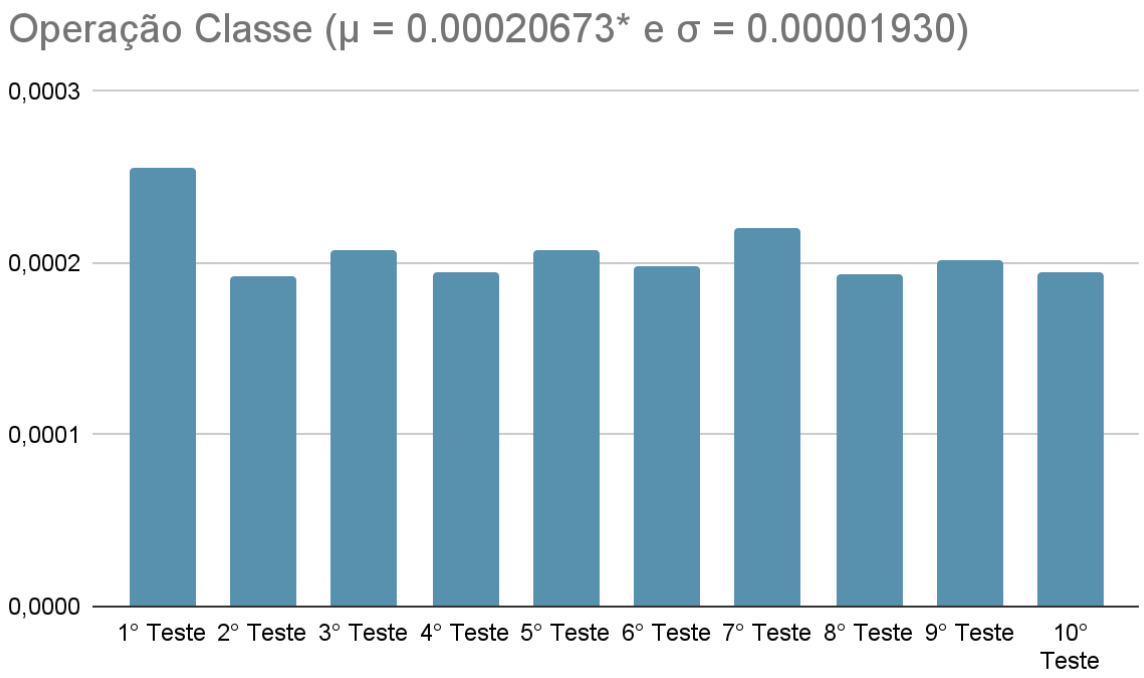
Para cada uma das operações, temos:


Operação Void (μ = 0.00019828 e σ = 0.00030228)
0,00125
0,00100
0,00075
0,00050
0,00025
0,00000
1° Teste
2° Teste
3° Teste
4° Teste
5° Teste
6° Teste
7° Teste
8° Teste
9° Teste
10° Teste

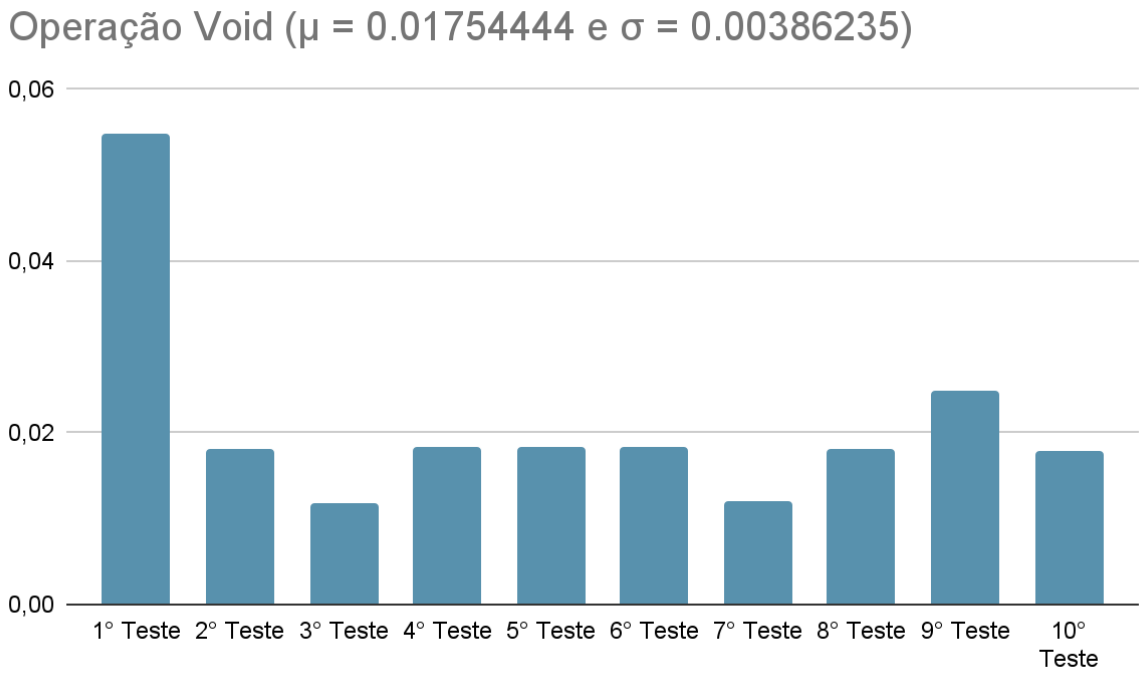

Operação Long de 1 Argumento (μ = 0.00019293 e σ = 0.00012734)
0,0006
0,0004
0,0002
0,0000
1° Teste
2° Teste
3° Teste
4° Teste
5° Teste
6° Teste
7° Teste
8° Teste
9° Teste
10° Teste

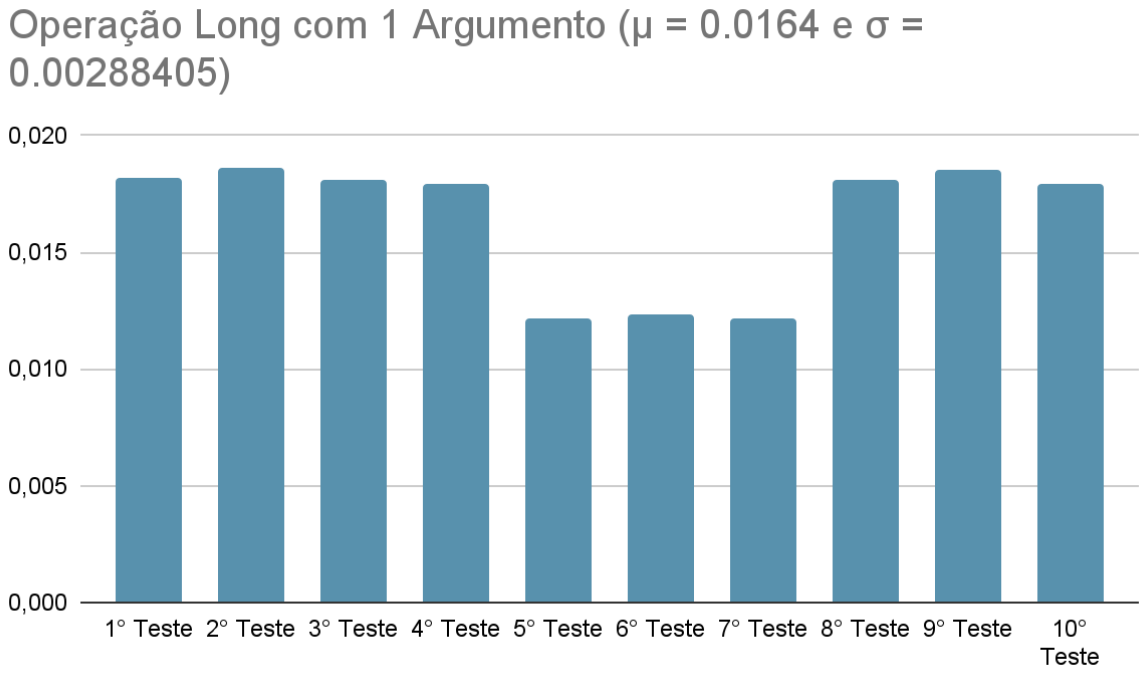

Operação Long de 8 Argumentos (μ = 0.00019717 e σ = 0.00009029)
0,0005
0,0004
0,0003
0,0002
0,0001
0,0000
1° Teste
2° Teste
3° Teste
4° Teste
5° Teste
6° Teste
7° Teste
8° Teste
9° Teste
10° Teste

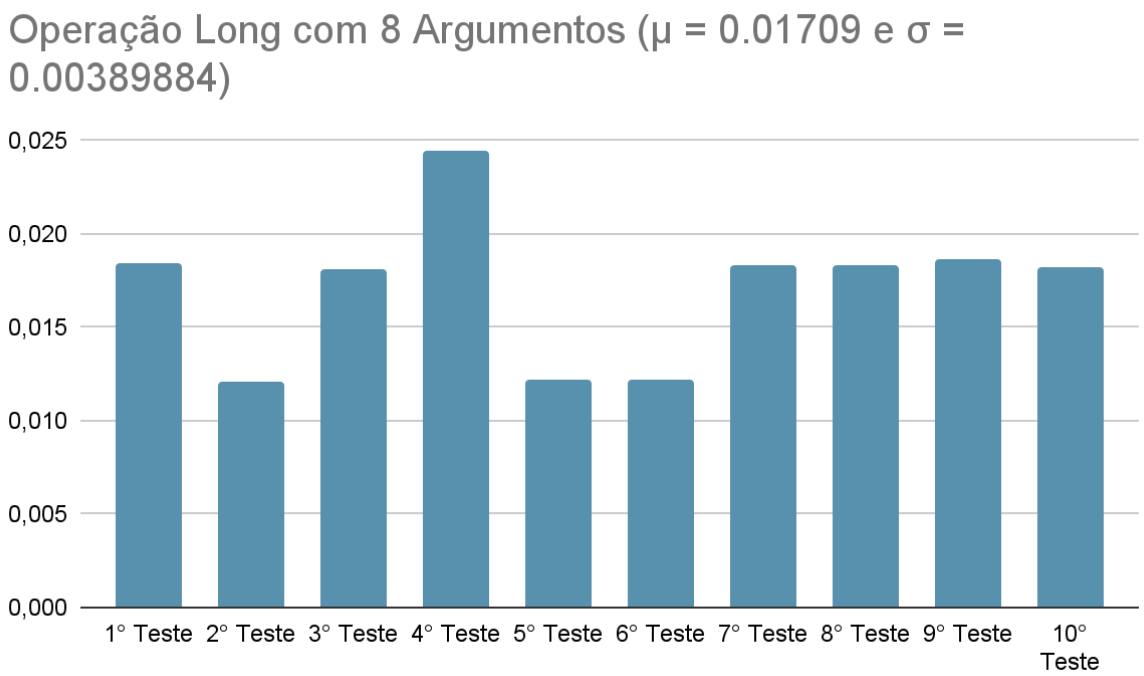

Operação String de Variados Tamanhos (μ = 0.00174165 e σ = 0.00142478)
0,008
0,006
0,004
0,002
0,000
1° Teste
2° Teste
3° Teste
4° Teste
5° Teste
6° Teste
7° Teste
8° Teste
9° Teste
10° Teste
1 Caracter
2 Caracteres
4 Caracteres
8 Caracteres
16 Caracteres
32 Caracteres
64 Caracteres
128 Caracteres
256 Caracteres
512 Caracteres
1024 Caracteres
2048 Caracteres

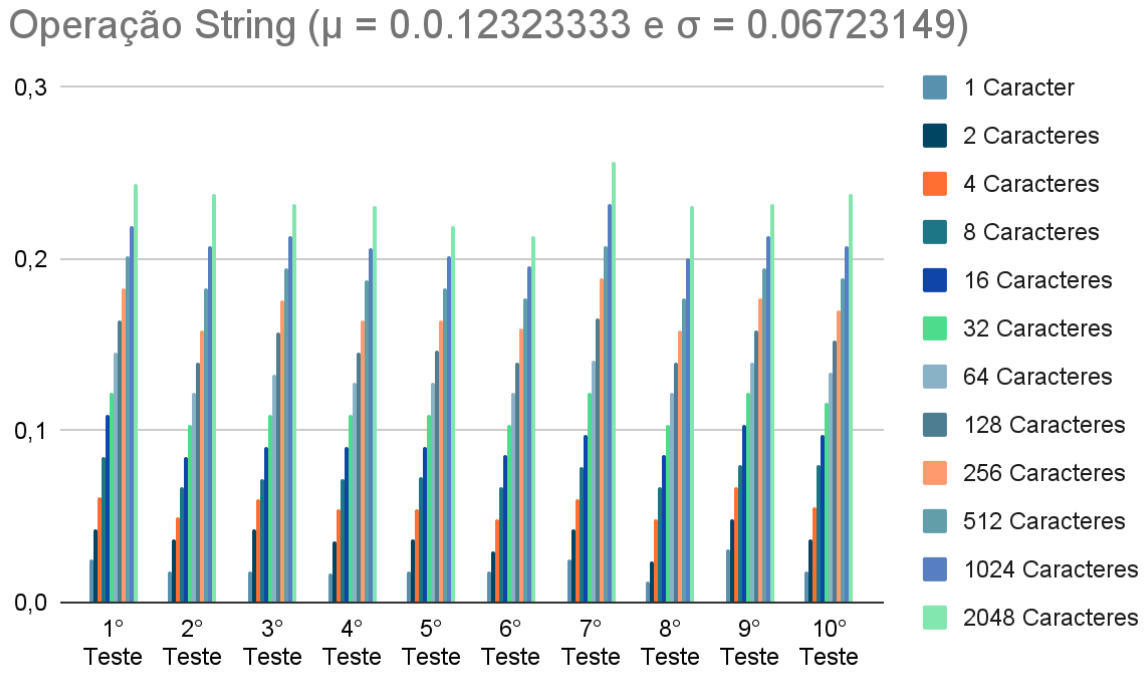


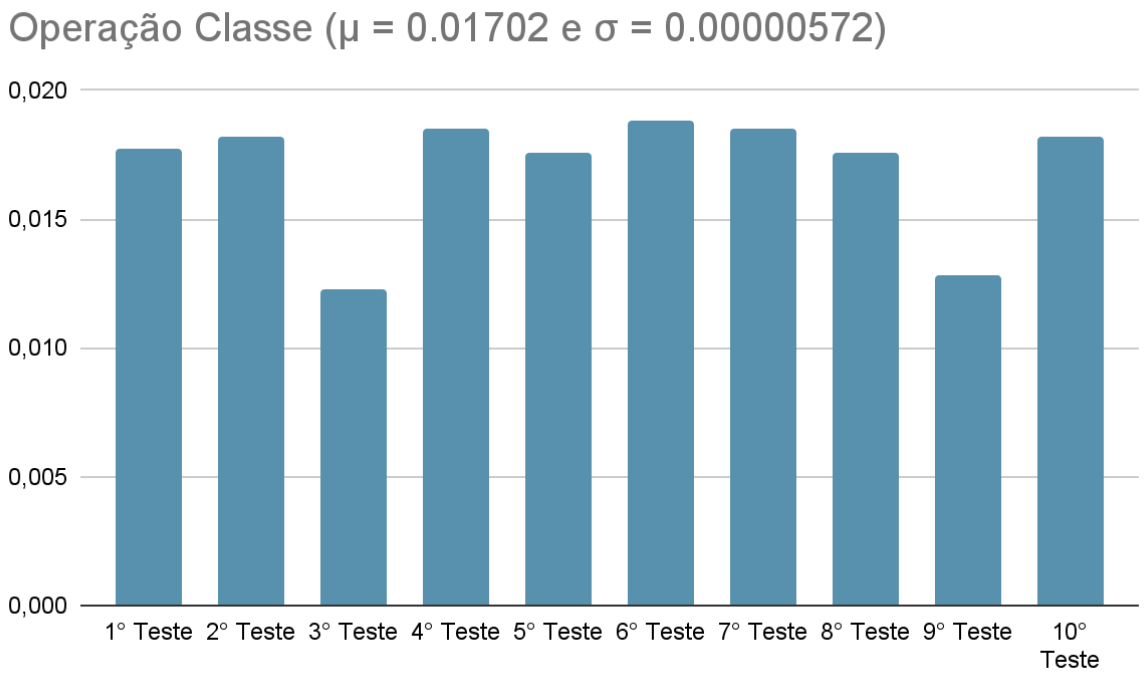
2.1.3 gRPC com Duas Máquinas




Operação Long com 1 Argumento (μ = 0.0164 e σ = 0.00288405)
0,020
0,015
0,010
0,005
0,000
1° Teste
2° Teste
3° Teste
4° Teste
5° Teste
6° Teste
7° Teste
8° Teste
9° Teste
10° Teste


Operação Long com 8 Argumentos (μ = 0.01709 e σ = 0.00389884)
0,025
0,020
0,015
0,010
0,005
0,000
1° Teste
2° Teste
3° Teste
4° Teste
5° Teste
6° Teste
7° Teste
8° Teste
9° Teste
10° Teste


Operação String (μ = 0.0.12323333 e σ = 0.06723149)
0,3
0,2
0,1
0,0
1° Teste
2° Teste
3° Teste
4° Teste
5° Teste
6° Teste
7° Teste
8° Teste
9° Teste
10° Teste
1 Caracter
2 Caracteres
4 Caracteres
8 Caracteres
16 Caracteres
32 Caracteres
64 Caracteres
128 Caracteres
256 Caracteres
512 Caracteres
1024 Caracteres
2048 Caracteres


Operação Classe (μ = 0.01702 e σ = 0.00000572)
0,020
0,015
0,010
0,005
0,000
1° Teste
2° Teste
3° Teste
4° Teste
5° Teste
6° Teste
7° Teste
8° Teste
9° Teste
10° Teste

2.2 Java RMI

2.2 Como executar

Passo 1: Entre nas pastas referentes ao cliente e servidor do JRMI e compile os arquivos utilizando o comando:

```
$ javac *.java
```

Passo 2: Abra 2 terminais

No terminal 1 inicie o servidor

```
$ java ServidorRMI
```

Após usar este comando, será solicitado o IP do servidor e a porta que deseja utilizar para comunicação (a porta 50050 está reservada para os stubs).

No terminal 2 inicie o cliente:

```
$ java ClienteRMI
```

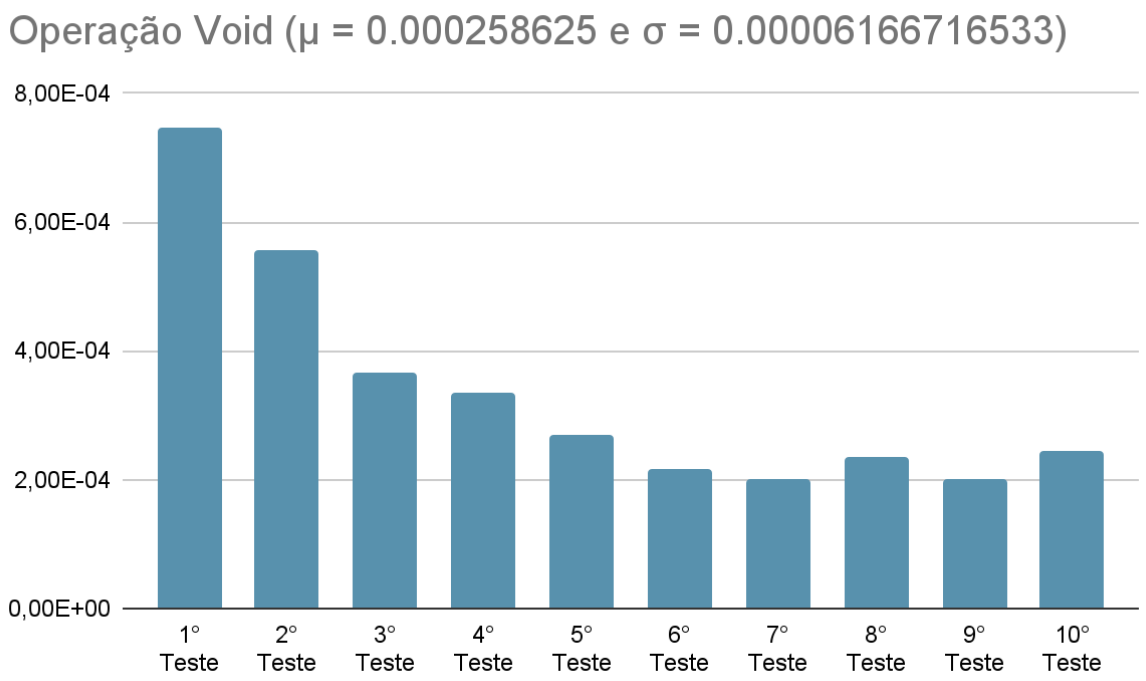
Após utilizar este comando, será solicitado o IP do servidor e a porta para realizar a comunicação, depois basta utilizar a interface interativa do terminal

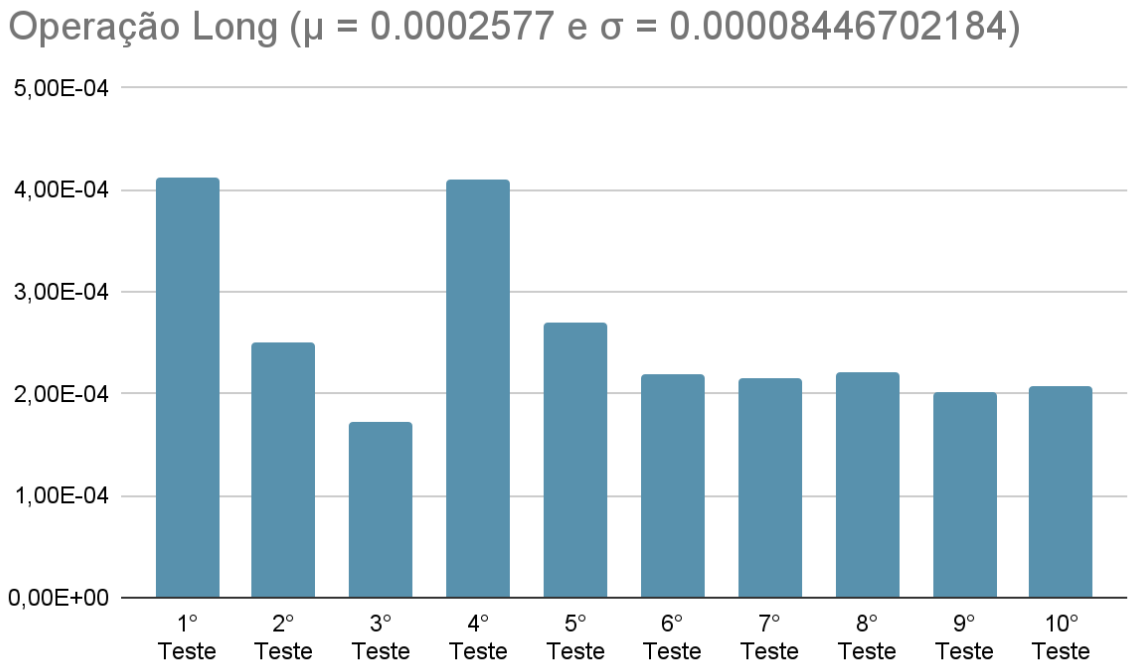
*Cliente que gera arquivos com os tempos de execução (tempos em segundos)

```
$ java ClienteRMI_Gera_txt
```

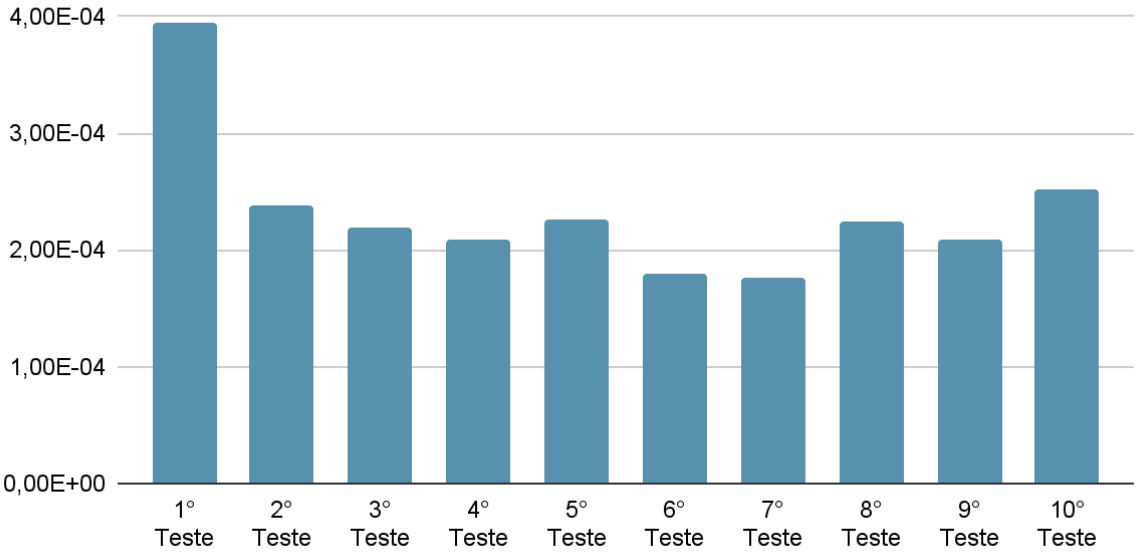
Digite também o IP e a Porta e ele gerará os tempos na pasta *tempos*.

2.2.1 Java RMI Local

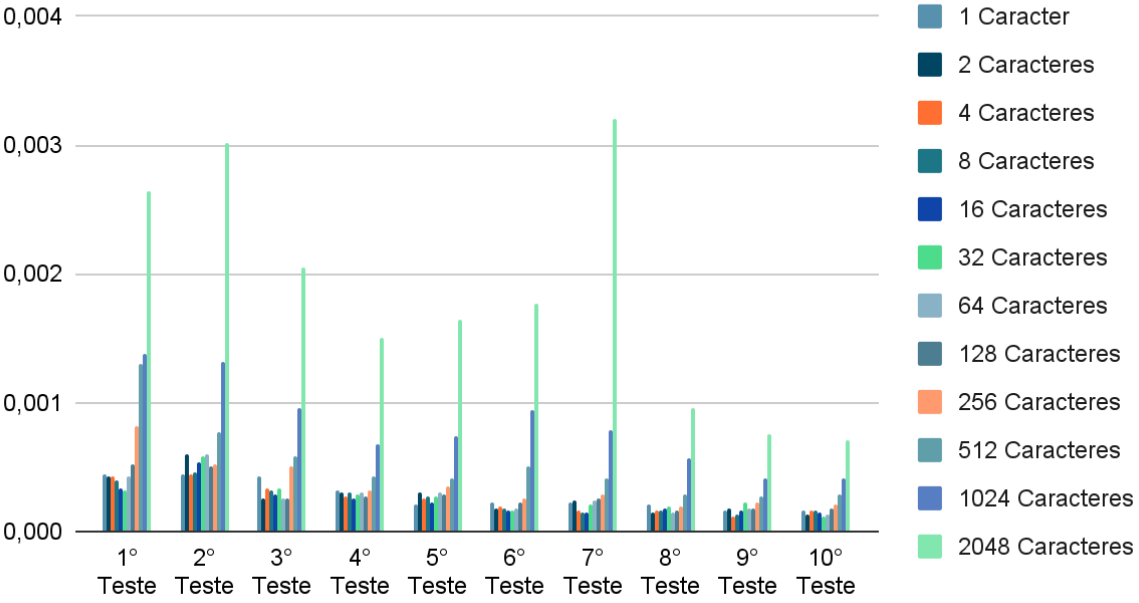


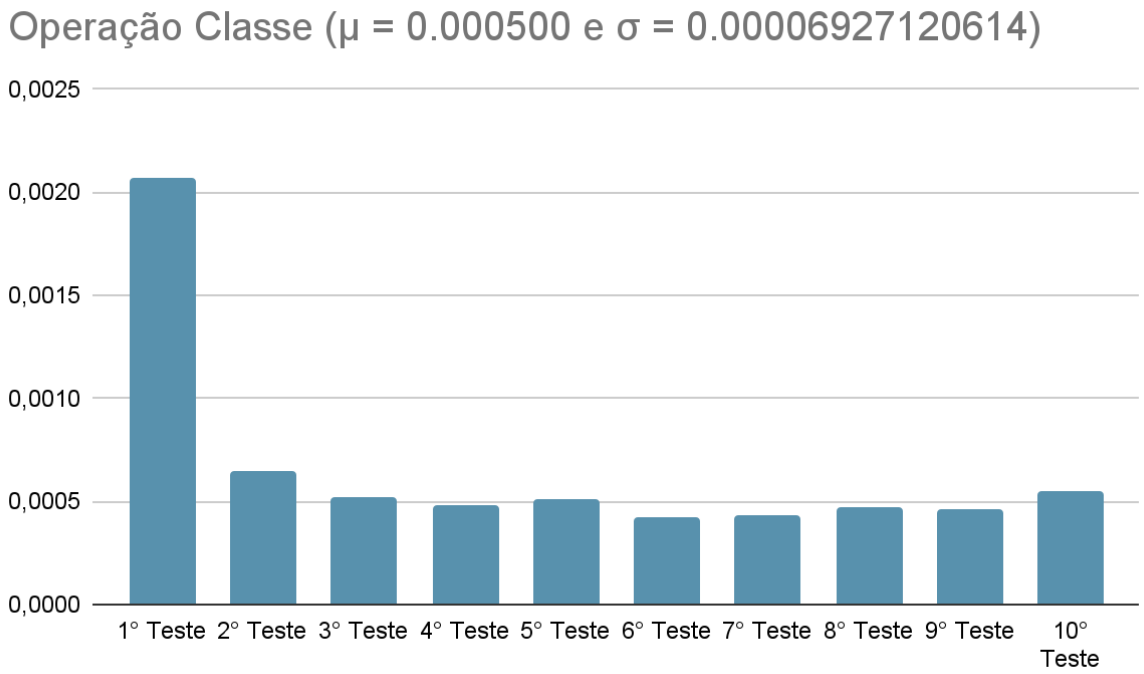


Operação Long com 8 Argumentos (μ = 0.0002327 e σ = 0.00005842097226)

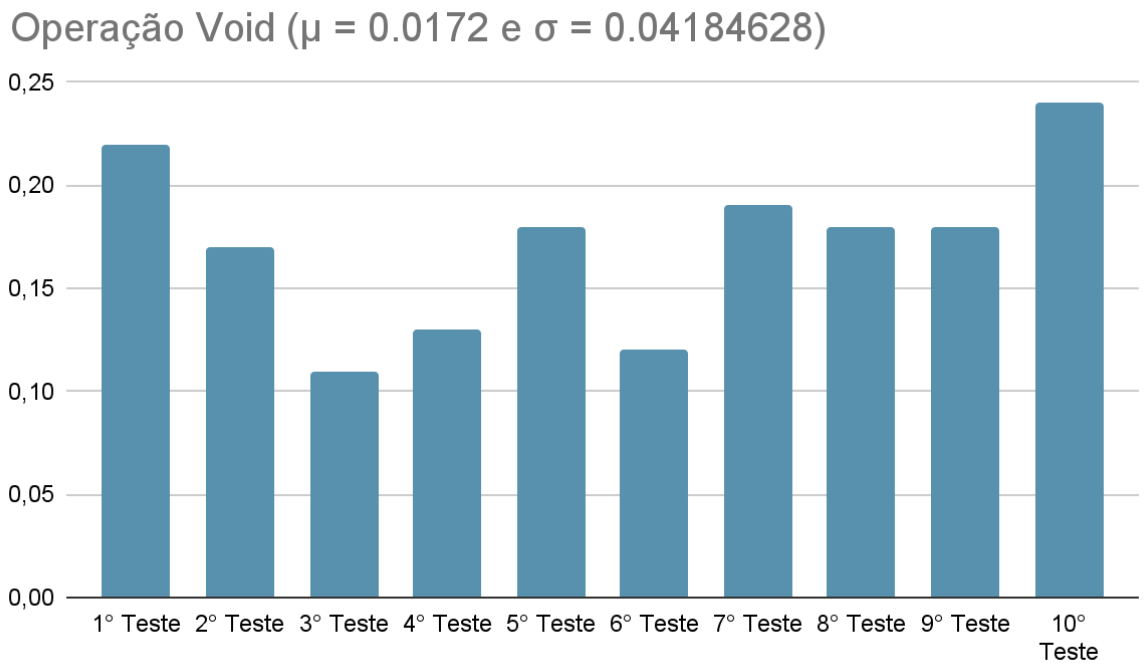


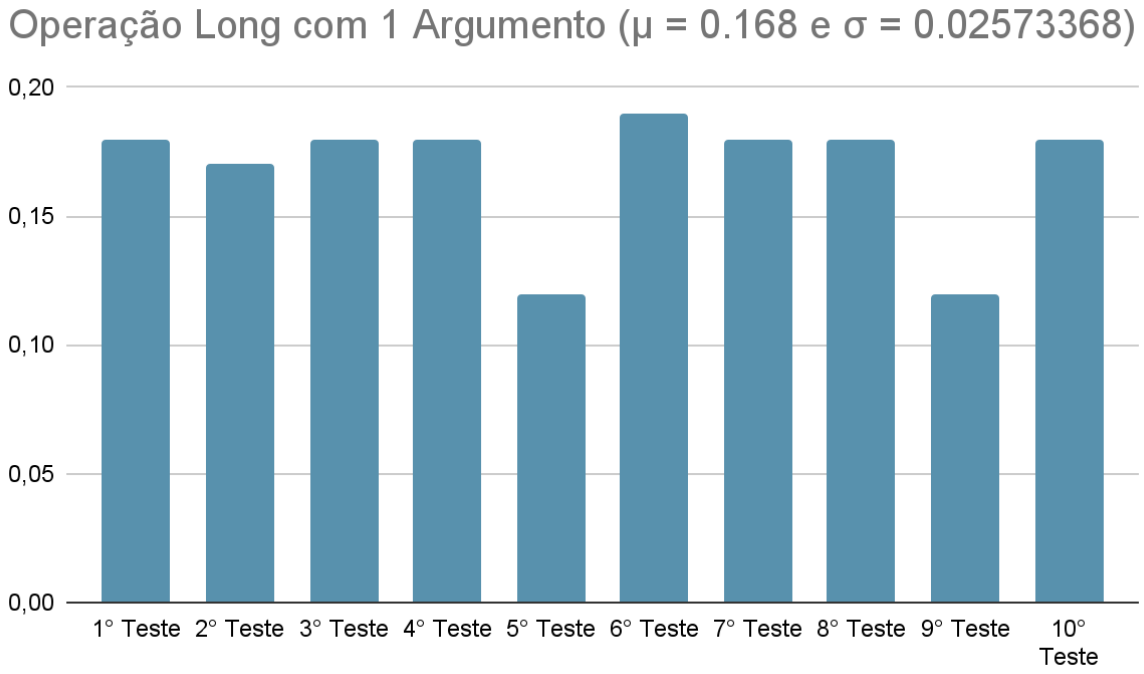
Operação String (μ = 0.0002327 e σ = 0.00006158111182)

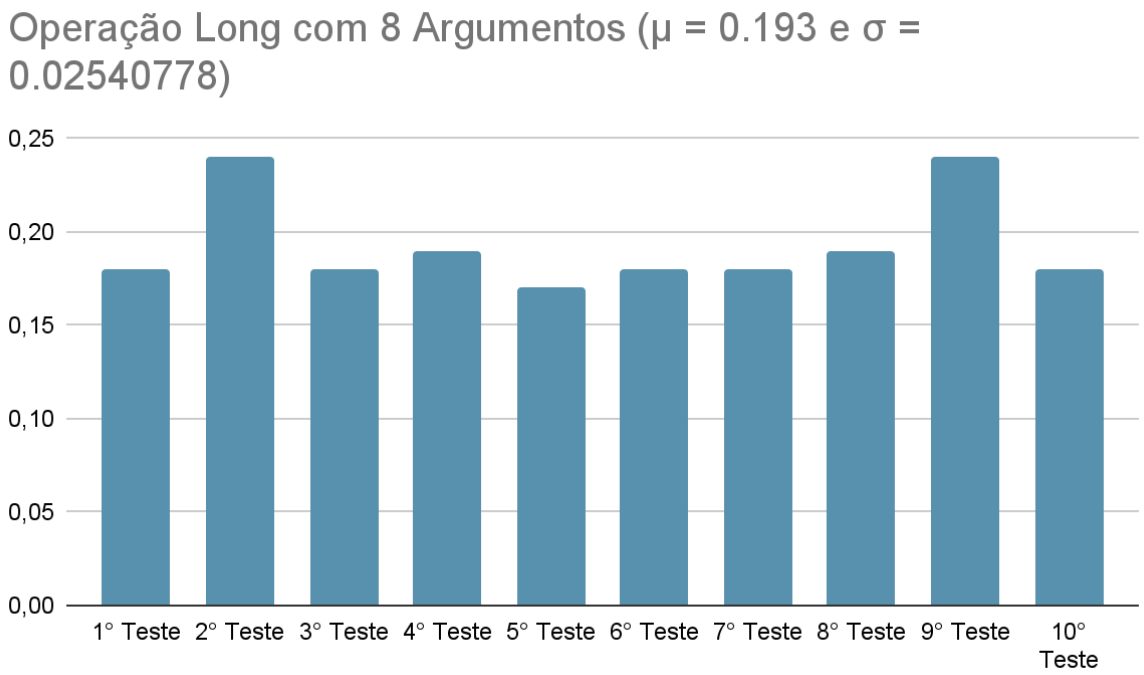


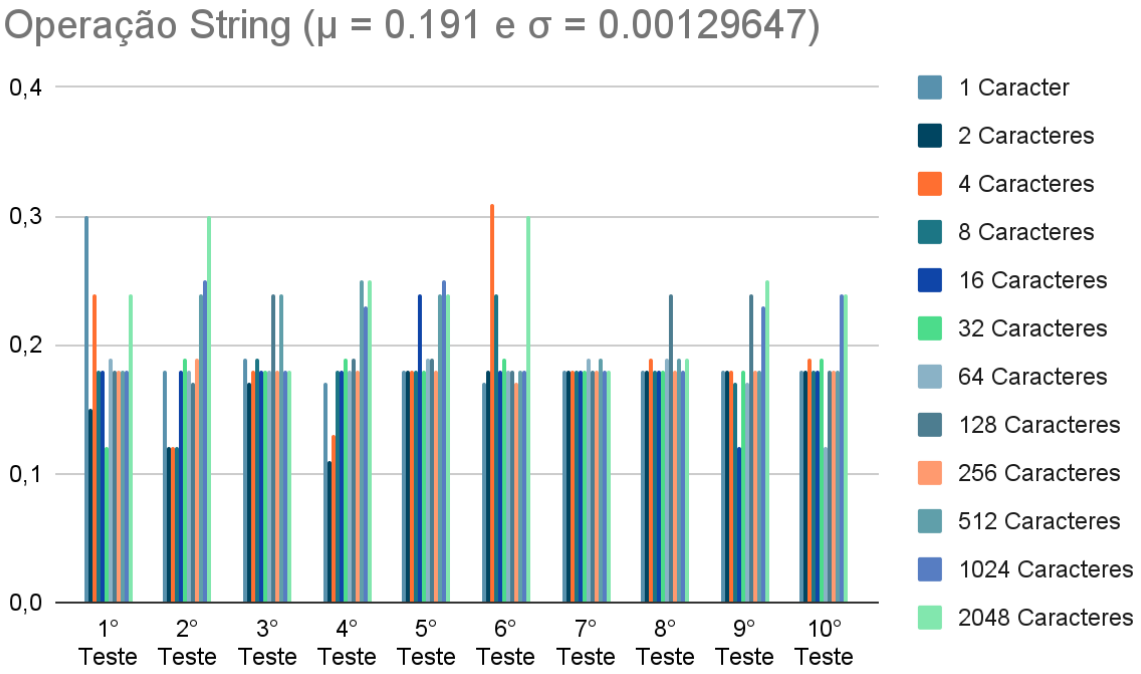


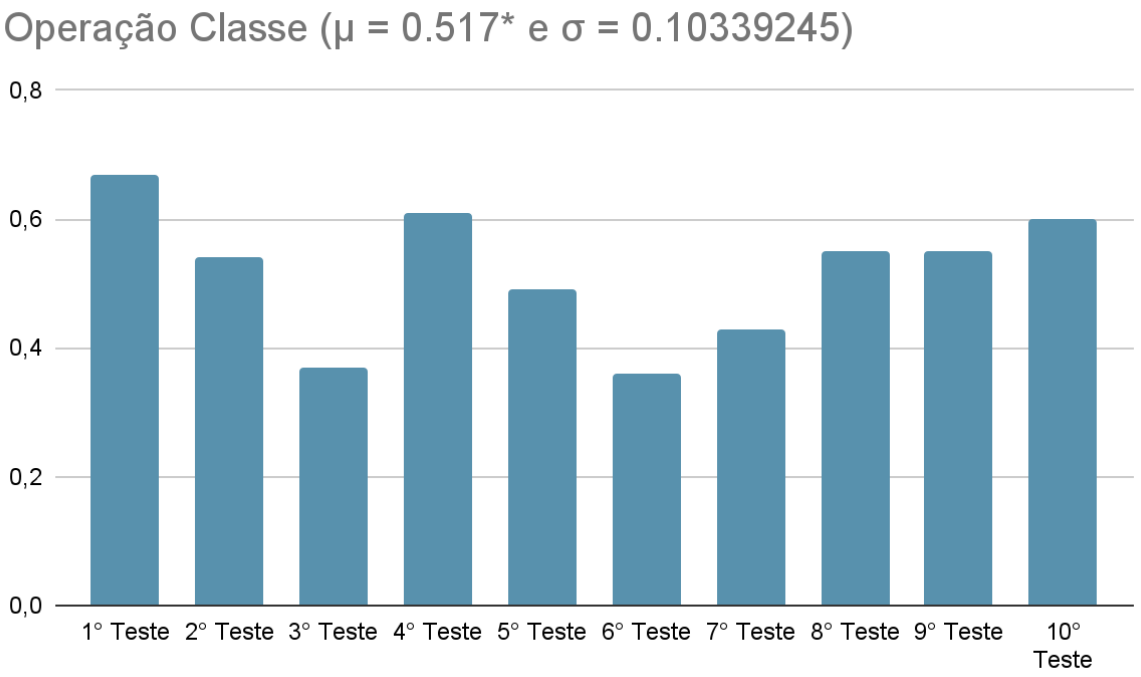
2.2.2 Java RMI em Duas Máquinas




Operação Long com 1 Argumento (μ = 0.168 e σ = 0.02573368)
0,20
0,15
0,10
0,05
0,00
1° Teste
2° Teste
3° Teste
4° Teste
5° Teste
6° Teste
7° Teste
8° Teste
9° Teste
10° Teste


Operação Long com 8 Argumentos (μ = 0.193 e σ = 0.02540778)
0,25
0,20
0,15
0,10
0,05
0,00
1° Teste
2° Teste
3° Teste
4° Teste
5° Teste
6° Teste
7° Teste
8° Teste
9° Teste
10° Teste


Operação String (μ = 0.191 e σ = 0.00129647)
0,4
0,3
0,2
0,1
0,0
1° Teste
2° Teste
3° Teste
4° Teste
5° Teste
6° Teste
7° Teste
8° Teste
9° Teste
10° Teste
1 Caracter
2 Caracteres
4 Caracteres
8 Caracteres
16 Caracteres
32 Caracteres
64 Caracteres
128 Caracteres
256 Caracteres
512 Caracteres
1024 Caracteres
2048 Caracteres


Operação Classe (μ = 0.517* e σ = 0.10339245)
0,8
0,6
0,4
0,2
0,0
1° Teste
2° Teste
3° Teste
4° Teste
5° Teste
6° Teste
7° Teste
8° Teste
9° Teste
10° Teste

3. Conclusões

Algumas tendências inicialmente previstas puderam ser observadas com os testes. As operações remotas demoraram mais que as locais, devido ao atraso de propagação, e também contaram com uma variação (desvio padrão) de valores ao longo dos testes, o que pode ser atribuído tanto à imprevisibilidade maior do estado momentâneo da rede, quanto à maior margem bruta para variação, já que estamos tratando de um intervalo maior de tempo (os testes locais tinham pouca liberdade de variação por serem muito rápidos em geral).

Além disso, as operações do gRPC demoraram mais que as em JRMI, muito provavelmente em função das linguagens usadas na implementação (python é consideravelmente menos performático do que Java).

Por fim, os *spikes* (picos momentâneos) de tempo da primeira operação são bem mais perceptíveis nos testes locais, possivelmente porque são a única variável de interferência agindo nos tempos, enquanto nas conexões locais eles “disputam” influência principalmente com o atraso de rede.