Universidade de São Paulo

Escola de Artes, Ciências e Humanidades

ACH2147 - Desenvolvimento de Sistemas de Informação Distribuídos

Camila Faria de Castro - nº 9019341

Leonardo de Melo Fontes - nº 9019390

**Exercício Programa**

**Remote Procedure Call (RPC)**

São Paulo

2018

1. **Introdução**

No contexto da disciplina ACH2147 - Desenvolvimento de Sistemas Distribuídos, o presente relatório tem como objetivo apresentar os resultados de uma avaliação de desempenho conduzida sob mecanismos de chamada de procedimento remoto (RPC - *Remote Procedure Call*).

Em sistemas distribuídos, chamadas de procedimento remoto são utilizadas para que programas possam executar procedimentos cuja implementação encontra-se em outra máquina. Segundo Steen e Tanenbaum (2017) [1], quando um processo em uma máquina A chama um procedimento implementado em uma máquina B, a chamada em A é transformada em uma requisição e transportada à B, que executa o procedimento normalmente e retorna à A o resultado. Essa troca de mensagens entre A e B é transparente para o programador, que não precisa ter conhecimento em relação ao local onde o procedimento está implementado. Portanto, no código-fonte do programa a chamada remota é declarada como se fosse local.

No presente trabalho foram avaliados os mecanismos **gRPC** [2] e **Java RMI** [3], ambos implementados usando-se a linguagem de programação Java. Para os testes, foi considerado um conjunto de trinta métodos, através dos quais exercitou-se diferentes combinações de parâmetros de entrada e saída, usando-se majoritariamente os tipos *int*, *long* e *string*. Cada mecanismo foi primeiro avaliado individualmente e depois de maneira comparativa, considerando-se dois cenários: *(i)* cliente e servidor na mesma máquina; *(ii)* cliente e servidor em máquinas diferentes.

Este relatório foi estruturado em seções da seguinte maneira: na seção II, descreve-se as instruções para execução dos projetos de cada mecanismo; na seção III apresenta-se a configuração das máquinas usadas nos testes. Na seção IV apresenta-se as avaliações de desempenho dos mecanismos, individual e comparativamente. Na seção V são discutidas algumas conclusões e considerações finais e, por último, na seção VI apresenta-se as referências bibliográficas utilizadas como apoio.

# Instruções para execução

* **gRPC**

Para executar o projeto de teste do mecanismo gRPC, primeiramente é necessário extrair a pasta “gRPC” anexada ao presente trabalho em um diretório do computador (exemplo: extrair para C:\gRPC).

No cenário em que *cliente e servidor estão na mesma máquina*, deve-se abrir dois prompts de comando no mesmo computador, um para executar o cliente e outro para executar o servidor. No cenário em que *cliente e servidor estão em máquinas diferentes*, deve-se abrir um prompt de comando em cada computador, pela mesma razão.

No prompt de comando, deve-se acessar o diretório principal do projeto (exemplo: cd C:\gRPC). Os comandos a serem utilizados para compilar o projeto, executar o cliente e o servidor são os seguintes:

|  |
| --- |
| **Para compilar**: mvn verify  **Para executar o servidor**: mvn exec:java -Dexec.mainClass=grpc.GRPCServer  **Para executar o cliente**: mvn exec:java -Dexec.mainClass=grpc.GRPCClient -Dexec.args="*hostname*", onde <*hostname*> é o nome do host no qual o servidor está sendo executado |

No cenário em que cliente e servidor estão na mesma máquina, o atributo *hostname* no comando para execução do cliente deve ser *localhost*. Já no cenário em que cliente eservidor estão em máquinas diferentes, deve-se incluir o número do IP da máquina na qual o servidor está rodando.

* **RMI**

Para executar o projeto de teste do mecanismo RMI, primeiramente é necessário extrair a pasta “RMI” anexada ao presente trabalho em um diretório do computador (exemplo: extrair para C:\RMI).

No cenário em que *cliente e servidor estão na mesma máquina*, deve-se abrir dois prompts de comando, um para executar o cliente e outro para executar o servidor. No cenário em que *cliente e servidor estão em máquinas diferentes*, deve-se abrir um prompt de comando em cada computador, pela mesma razão.

No prompt de comando, deve-se acessar o diretório principal do projeto (exemplo: cd C:\RMI). Os comandos a serem utilizados para compilar o projeto, executar o cliente e o servidor são os seguintes:

|  |
| --- |
| **Para compilar**: mvn verify  **Para executar o servidor**: mvn exec:java -Djava.rmi.server.hostname=*hostname* -Dexec.mainClass=rmi.RMIServer, onde <*hostname*> é o nome do host no qual o servidor está sendo executado.  **Para executar o cliente**: mvn exec:java -Dexec.args="*hostname*" -Dexec.mainClass=rmi.RMIClient, onde <*hostname*> é o nome do host no qual o servidor está sendo executado. |

O atributo *hostname* nas instruções de execução do servidor e do cliente deve ser o mesmo, correspondente ao nome do host no qual o servidor está sendo executado. No cenário em que cliente e servidor estão na mesma máquina, o atributo *hostname* deve ser *localhost*. Já no cenário em que cliente eservidor estão em máquinas diferentes, deve-se incluir o número do IP da máquina na qual o servidor está rodando.

1. **Configuração das máquinas utilizadas no teste**

No cenário em que *cliente e servidor estão na mesma máquina*, os testes de ambos os mecanismos foram conduzidos em uma máquina com sistema operacional Windows 10, 64 bits, processador Intel(R) Core(™) i7-3517U CPU @ 1.90GHz, 2 cores, com 6GB de memória RAM.

No cenário em que *cliente e servidor estão em máquinas diferentes*, para ambos os mecanismos a máquina utilizada para executar o servidor é a mesma cujas configurações estão descritas no parágrafo anterior. Para executar o cliente, por sua vez, utilizou-se uma máquina com sistema operacional Linux Mint 18.3, processador Intel(R) Core(™) i7-7700HQ @ 2.80GHz, 4 cores, com 4GB de memória RAM. Nesse caso, a comunicação entre o cliente e o servidor foi feita através de uma rede wireless, usando um roteador do modelo TL-WR541G, frequência de 2.4 GHz e velocidade máxima de 54MBps. O ambiente da rede não estava totalmente isolado, estando conectados outros dispositivos ao roteador ao mesmo tempo.

Em ambas as máquinas consideradas, utilizou-se a linguagem de programação Java, considerando-se a versão 1.8.0\_151 da JDK e a versão 3.5.3 do Apache Maven. Para o teste do mecanismo gRPC, utilizou-se a build de versão 1.12.0 e seus respectivos compiladores.

1. **Avaliação de desempenho**

Na presente seção, apresenta-se a avaliação de desempenho de cada mecanismo de RPC, individual e comparativamente, considerando-se dois cenários: *(i)* cliente e servidor na mesma máquina; *(ii)* cliente e servidor em máquinas diferentes.

Para os testes, foi considerado um mesmo conjunto de trinta métodos com diferentes combinações de parâmetros de entrada e saída, usando-se majoritariamente os tipos *int*, *long* e *string*, sendo a última de diferentes tamanhos. Também foi criada uma classe de teste denominada *Localizacao*, a qual foi utilizada para testar procedimentos com valor de retorno de tipo complexo (i.e. objetos).

A implementação dos métodos nos dois mecanismos é praticamente idêntica, salvo apenas diferenças relativas ao uso de bibliotecas distintas. Para garantir que não existirão otimizações ou caches criados pelo compilador, um mesmo conjunto arbitrário de operações simples foi executado em todos os métodos, conforme mostrado abaixo. A depender do método, o valor de c pode ser concatenado a uma string, somado a um long, entre outras situações.

int a = 1 + 1;

int b = 1 \* 1;

int c = a/b;

Buscando-se evitar possíveis interferências nos resultados, a execução dos testes foi isolada, isto é, apenas os prompts de comando executando-se o cliente e o servidor foram mantidos em execução na(s) máquina(s). O conjunto de métodos foi executado 10 vezes em cada um dos cenários, ou seja, executamos todos os métodos uma vez, em seguida todos os métodos em sequência outra vez, até completar 10 execuções, e computamos os tempos de cada execução separadamente. Dessa forma garantimos que, se houver alguma oscilação de rede ou de desempenho, é mais provável que a mesma afete apenas uma execução de cada método, podendo posteriormente ser desconsiderada no processo de remoção dos valores extremos.

O tempo de execução foi medido a partir do momento em que o procedimento remoto é invocado no cliente, até o momento em que a resposta é retornada ao mesmo, desconsiderando-se o tempo de criação do *stub*.

Para aumentar o grau de confiabilidade nos dados da amostra, os resultados obtidos foram filtrados e os valores extremos descartados considerando-se um intervalo de confiança de 95%. Posteriormente, a média e o desvio padrão foram calculados, resultando-se nos valores a serem apresentados a seguir.

No cenário em que cliente e servidor foram executados em máquinas diferentes, mediu-se a latência estimada de rede na comunicação usando-se o comando **ping <número\_do\_ip>**, e verificou-se um *round-trip time* (RTT) de no mínimo 3ms e no máximo 10ms, sendo a média 5ms. Portanto, é razoável esperar-se que as médias do tempo de execução dos métodos em máquinas diferentes seja, no mínimo, 3ms a mais do que a média de tempo de execução do mesmo método estando o cliente e o servidor na mesma máquina.

Neste contexto, a avaliação de desempenho dos mecanismos foi dividida em cinco seções, de acordo com o conjunto de métodos a serem avaliados: *(a)* métodos que recebem e retornam o mesmo tipo de variável; *(b)* métodos que recebem void e retornam um tipo de variável; *(c)* métodos que recebem um tipo de variável e retornam void; *(d)* métodos que recebem mais de um argumento; *(e)* métodos que retornam objetos. O desempenho dos mecanismos em cada um dos critérios mencionados é apresentado a seguir.

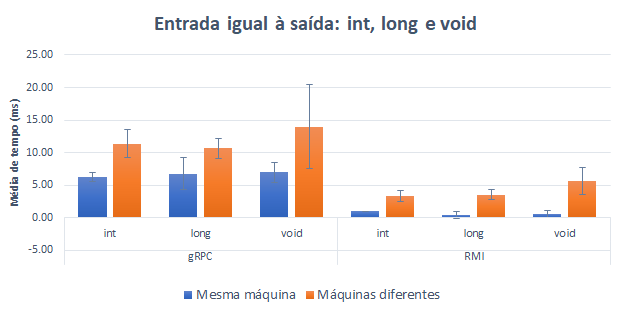
1. **Métodos que recebem e retornam o mesmo tipo**

O primeiro aspecto sob o qual o gRPC e o RMI foram avaliados refere-se ao teste de operações que recebem como argumento uma variável de determinado tipo e retornam um valor com o mesmo tipo da entrada. Também são considerados nessa categoria métodos que não recebem e não retornam nenhum argumento (void).

Sendo assim, os métodos considerados nessa análise são: **voidVoid**, **intInt**, **longLong** e **stringString**, sendo este último testado considerando-se strings de tamanho 2*n*, variando *n* de 1 à 10. Na Figura 1, apresenta-se os resultados obtidos para a execução dos métodos voidVoid, intInt e longLong, na mesma máquina e em máquinas diferentes, para ambos os mecanismos, respectivamente.

Em relação ao mecanismo gRPC, observa-se que a média de tempo de execução dos três métodos foi bastante próxima: no cenário em que cliente e servidor estavam na mesma máquina foi em torno de 6ms, enquanto em máquinas diferentes foi em torno de 12ms. Neste último, observa-se que a média do tempo de execução do método voidVoid foi ligeiramente maior em relação aos demais, porém os dados obtidos também apresentaram uma maior dispersão.

A proximidade da média do tempo de execução nos três métodos também pôde ser observada nos resultados do mecanismo RMI: no cenário em que cliente e servidor estavam na mesma máquina, foi em torno de 1ms, enquanto em máquinas diferentes foi em torno de 4ms. Mais uma vez, a média do tempo de execução do método voidVoid foi ligeiramente maior em relação aos demais no cenário em que cliente e servidor foram executados em máquinas diferentes, também apresentando uma maior dispersão.



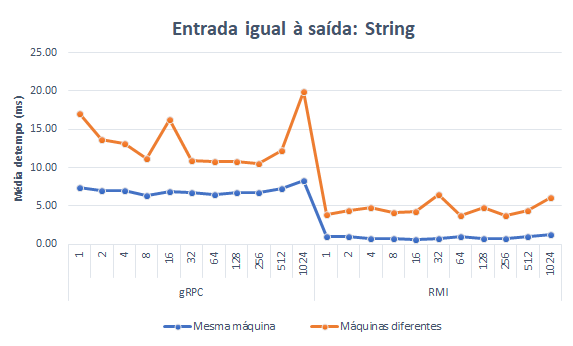
**Figura 1:** Testes em procedimentos cujo tipo de entrada igual à saída (int, long e void)

Dessa forma, o mesmo comportamento pôde ser observado em ambos os mecanismos: a média dos tempos de execução dos respectivos métodos se manteve muito próxima, mostrando-se ser ligeiramente maior apenas quando o voidVoid foi executado em máquinas diferentes. No entanto, neste último caso, os dados obtidos também apresentaram um desvio padrão maior em relação aos outros métodos, o que pode indicar que a execução do voidVoid foi mais afetada pela latência da rede. Essa discrepância se tornou mais evidente ao analisar a tabela de dados obtidos para o método voidVoid em máquinas diferentes: no gRPC o tempo de execução do método variou de 9ms a 17ms, havendo um valor extremo de 30ms não descartado, enquanto no RMI variou de 3ms a 8ms.

O fato de que a média do tempo de execução do voidVoid foi próxima ou ligeiramente maior do que a média dos métodos intInt e longLong em ambos os mecanismos é, de certa forma, surpreendente: como no primeiro não é passado nenhum argumento, esperava-se que o tempo de execução fosse menor. Na prática, esse resultado indica que, usando-se os mecanismos gRPC e RMI para chamada remota, não faz muita diferença em termos de desempenho não passar nenhum argumento, ou passar um int ou um long.

Na Figura 2, apresenta-se os resultados obtidos para a execução do método stringString, que recebe uma string de tamanho 3 (“Ola”) e retorna a mesma concatenada a uma string de tamanho *n*, estando n contido no conjunto { 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024 }. Na figura em questão, são apresentadas as médias do tempo de execução desse método nos mecanismos gRPC e RMI para cada tamanho de string, considerando-se os cenários de execução do cliente e servidor na mesma máquina e em máquinas diferentes.

Em relação ao gRPC, no cenário em que cliente e servidor estão na mesma máquina, observa-se que até o tamanho 512 a média do tempo de execução foi próxima a 7ms, oscilando para um pouco menos no tamanho 8. A partir de 1.024, nota-se que a linha do gráfico começa em uma tendência de subida, fato que também pode ser observado nos resultados obtidos para cliente e servidor em máquinas diferentes.



**Figura 2:** Testes em procedimentos cujo tipo de entrada igual à saída (string)

Sobre este último cenário, é interessante notar que a média de tempo de execução foi diminuindo do tamanho 1 a 8, atingindo um grande pico no tamanho 16 e depois voltando a diminuir até o tamanho 256, quando volta novamente a subir. Também observa-se que a inclinação da subida da média de tempo de execução do tamanho 512 para 1.024 foi muito maior no cenário de máquinas diferentes em relação ao cenário em que cliente e servidor estão na mesma máquina.

Em relação ao RMI, no cenário em que cliente e servidor estão na mesma máquina, observa-se que a média do tempo de execução foi próxima a 1ms, atingindo seu máximo no tamanho 1.024. Em máquinas diferentes, a média segue uma tendência de subida até o tamanho 4, atingindo seu máximo no tamanho 32 e depois voltando a subir a partir do tamanho 256. Neste caso, é interessante notar que a média do tempo de execução do tamanho 1.024 foi minimamente inferior à do tamanho 32.

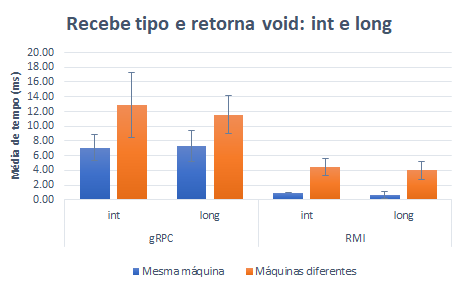
De maneira geral, em ambos os mecanismos não foi observada de maneira evidente uma tendência de crescimento da média do tempo de execução como reflexo do crescimento do tamanho da string. Essa associação só começou a se tornar mais clara a partir do tamanho 256, em que a média de tempos começou a subir nos dois mecanismos, para os dois cenários considerados. Isso indica que strings de 1 a 128 caracteres ainda são pequenas, e o crescimento de seu tamanho neste intervalo ainda acontece de maneira muito gradual: aumentar de 16 para 32 caracteres, ou de 32 para 64 caracteres, ainda não é tão significativo a ponto de afetar o desempenho da chamada remota. A partir de 256 caracteres, a curva de crescimento já se torna um pouco mais acentuada, impactando mais significativamente no tamanho e, consequentemente, no tempo de execução.

Caso fossem feitos testes com tamanhos de string superiores a 1.024, provavelmente observar-se-ia a continuação do crescimento das médias de tempo de execução. É natural pensar dessa forma pois, quanto maior o tamanho da string, mais dados devem ser transferidos do cliente para o servidor e, portanto, maior o tempo para conclusão da requisição da chamada remota. Tal aspecto fica mais evidente quando cliente e servidor estão sendo executados em máquinas diferentes, pois agrega-se no RTT o tempo para o conjunto de caracteres sair da máquina do servidor e chegar à máquina do cliente. Isso pode ser observado na Figura 2: ao aumentar o tamanho da string de 512 para 1.024 no mecanismo gRPC, a média de tempo de execução aumentou 63% no cenário de máquinas diferentes, e 15.2% em máquinas iguais.

Para ambos os testes discutidos nesta seção, ao comparar os mecanismos gRPC e RMI em termos de desempenho, pode-se observar que os métodos do RMI foram executados mais rapidamente nos dois cenários considerados - mesma máquina e máquinas diferentes.

1. **Métodos que recebem um tipo e retornam void**

O segundo aspecto sob o qual o gRPC e o RMI foram avaliados refere-se ao teste de operações que recebem como argumento uma variável de determinado tipo e não retornam nada, isto é, *void.* Para essa análise, foram considerados os métodos **intVoid**, **longVoid** e **stringVoid**, sendo este último testado considerando-se strings de tamanho 10, 1.024 e 102.400. Na Figura 3, apresenta-se os resultados obtidos para a execução dos métodos intVoid e longVoid, na mesma máquina e em máquinas diferentes, para ambos os mecanismos, respectivamente.



**Figura 3:** Testes em procedimentos que recebem int ou long, e retornam void

Em relação ao mecanismo gRPC, observa-se que a média de tempo de execução dos dois métodos foi bastante próxima: no cenário em cliente e servidor estavam na mesma máquina foi em torno de 7ms, enquanto em máquinas diferentes foi em torno de 12ms. Neste último, observa-se que a média do tempo de execução do método intVoid foi ligeiramente maior em relação ao longVoid, porém os dados obtidos também apresentaram uma maior dispersão.

A proximidade da média do tempo de execução nos dois métodos também pôde ser observada nos resultados do mecanismo RMI: no cenário em que cliente e servidor estavam na mesma máquina, foi em torno de 1ms, enquanto em máquinas diferentes foi em torno de 4ms. Mais uma vez, a média do tempo de execução do método intVoid foi ligeiramente maior em relação ao longVoid no cenário em que cliente e servidor foram executados em máquinas diferentes. Porém, diferentemente do gRPC, o desvio padrão observado para este método não foi muito maior do que para longVoid.

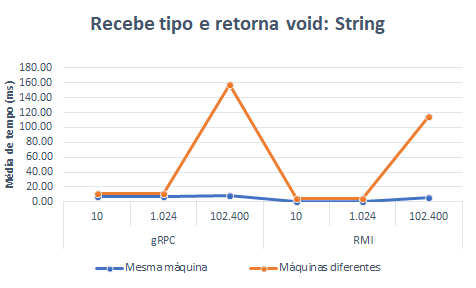
Dessa forma, o mesmo comportamento pôde ser observado em ambos os mecanismos: a média de tempos de execução dos respectivos métodos se manteve muito próxima, mostrando-se ser ligeiramente maior apenas quando o intVoid foi executado em máquinas diferentes. Este resultado é, de certa forma, surpreendente: como o tamanho de um long é maior do que um int, esperava-se que o tempo de execução de uma chamada remota passando-se um long como argumento fosse maior. Mesmo que os testes realizados consideraram o envio por parte do cliente de um int de tamanho 5 e um long de tamanho 12, em Java isso não faz diferença, pois inteiros são sempre representados com 32 bits, e longs com 64.

Outro ponto interessante a ser notado é que a média de tempo de execução destes métodos não foi muito diferente dos métodos apresentados na seção (A), cujo tipo do argumento de entrada é o mesmo do valor a ser retornado. Na prática, esse resultado pode indicar que, usando-se os mecanismos gRPC e RMI para chamada remota, não faz muita diferença em termos de desempenho passar argumentos do mesmo tipo (int e long) como parâmetro, ou passar um determinado tipo e receber void.

Na Figura 4, apresenta-se os resultados obtidos para a execução do método stringVoid, que recebe como argumento uma string de tamanho 10, 1.024 ou 102.400 e não retorna nada. Nestes testes, optou-se por utilizar tamanhos mais esparsos, numa tentativa de verificar o quanto um crescimento mais acentuado no número de caracteres de uma string impacta no tempo de execução da chamada remota.

Em relação ao gRPC, no cenário em que cliente e servidor estão na mesma máquina, o aumento do tamanho da string refletiu no aumento da média de tempo de execução, porém não tão significativamente dado o índice de crescimento do tamanho: aumentando-se de 1.024 para 102.400 caracteres, a média de tempo de execução subiu de 6.60ms para 8.11ms, apenas. O RMI, por sua vez, apresentou um aumento muito mais significativo - de 0.70ms para 5.44ms.

O aumento mais significativo observado neste teste certamente se refere ao cenário em que cliente e servidor estão executando em máquinas diferentes. No caso do gRPC, aumentando-se de 1.024 para 102.400 caracteres, a média de tempo de execução subiu de 11.67ms para 158.22ms, enquanto no RMI subiu de 4.78ms para 114.89ms. Esse resultado confirma o que foi mencionado nos testes com string descritos em (A), indicando que o desempenho de chamadas remotas considerando-se strings de tamanhos grandes é mais afetado quando cliente e servidor estão em máquinas diferentes.



**Figura 4:** Testes em procedimentos que recebem string de diferentes tamanhos, e retornam void

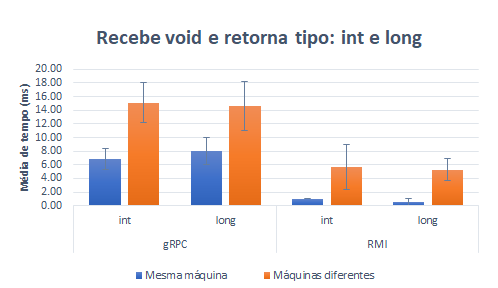
Por fim, comparando-se o gRPC com o RMI em termos de desempenho, pode-se observar que os métodos do RMI foram executados mais rapidamente, porém a diferença nas médias observadas em ambos os mecanismos diminuiu bastante nos testes com strings de tamanho 102.400.

1. **Métodos que recebem void e retornam um tipo**

O terceiro aspecto sob o qual o gRPC e o RMI foram avaliados refere-se ao teste de operações que não recebem argumentos e retornam uma variável de determinado tipo. Para essa análise, foram considerados os métodos **voidInt, voidLong, voidString** (tamanho 10)**, voidBigString** (tamanho 1.024)e **voidSuperString** (tamanho 102.400). Na Figura 5, apresenta-se os resultados obtidos para a execução dos métodos voidInt e voidLong, na mesma máquina e em máquinas diferentes, para ambos os mecanismos, respectivamente.

Em relação ao mecanismo gRPC, observa-se que a média de tempo de execução dos dois métodos foi bastante próxima: no cenário em cliente e servidor estavam na mesma máquina foi em torno de 7ms, enquanto em máquinas diferentes foi em torno de 15ms. Neste último, observa-se que a média do tempo de execução do método voidInt foi ligeiramente maior em relação ao voidLong, porém foi identificado um desvio padrão maior do último em relação ao primeiro.

A proximidade da média do tempo de execução nos dois métodos também pôde ser observada nos resultados do mecanismo RMI: no cenário em que cliente e servidor estavam na mesma máquina, foi em torno de 1ms, enquanto em máquinas diferentes foi em torno de 5ms. Mais uma vez, a média do tempo de execução do método voidInt foi ligeiramente maior em relação ao voidLong no cenário em que cliente e servidor foram executados em máquinas diferentes. Porém, diferentemente do gRPC, o desvio padrão observado para o primeiro foi muito maior do que para o segundo.



**Figura 5:** Testes em procedimentos que recebem void, e retornam int ou long

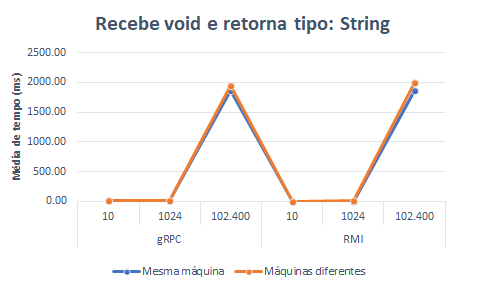
Seguindo o padrão de resultados dos métodos discutidos em (A) e (B), pode-se observar o mesmo comportamento em ambos os mecanismos: a média de tempos de execução dos respectivos métodos se manteve muito próxima, mostrando-se ser ligeiramente maior apenas quando o voidInt foi executado em máquinas diferentes.

A principal diferença desses métodos para os discutidos em (A) e (B) refere-se ao processamento executado no servidor para, a partir de uma chamada remota sem argumentos, gerar um valor de retorno. Como as operações executadas em todos os métodos considerados foram as mesmas, divergindo-se apenas em relação ao recebimento ou não de argumentos de entrada e saída, pôde-se isolar aspectos relacionados ao processamento e de fato observar a influência dos argumentos no tempo de execução das chamadas remotas. Comparando-se os valores obtidos com os métodos discutidos em (A) e (B), observa-se que a média de tempo de execução destes métodos não foi muito diferente. Na prática, esse resultado pode indicar que, usando os mecanismos gRPC e RMI para chamada remota, não faz muita diferença em termos de desempenho passar argumentos do mesmo tipo (int e long) como parâmetro, passar um determinado tipo e receber void, ou mandar void e receber um resultado de determinado tipo.

Para este teste, ao comparar os mecanismos gRPC e RMI em termos de desempenho, pode-se observar que os métodos do RMI foram executados mais rapidamente nos dois cenários considerados - mesma máquina e máquinas diferentes.

Na Figura 6, apresenta-se os resultados obtidos para a execução do método voidString, que não recebe argumentos (isto é, *void*) e retorna uma string de tamanho 10, 1.024 ou 102.400. Nestes testes, optou-se por utilizar tamanhos mais esparsos, numa tentativa de verificar o quanto um crescimento mais acentuado no número de caracteres de uma string impacta no tempo de execução da chamada remota.

Em relação ao gRPC, no cenário em que cliente e servidor estão na mesma máquina, o aumento do tamanho da string refletiu em um aumento bastante significativo da média de tempo de execução: aumentando-se de 1.024 para 102.400 caracteres, a média de tempo de execução subiu de 9.22ms para 1862.56ms. O RMI, por sua vez, aumentou mais ainda - de 1.56ms para 1862ms.

****

**Figura 6:** Testes em procedimentos que recebem void, e retornam string

Ao comparar o presente teste com o método stringVoid discutido em (B), em que se recebe uma variável de um determinado tipo e se retorna *void*, observa-se que a curva de crescimento do tempo de execução foi muito mais acentuada neste teste ao aumentar o tamanho da string. Isso se deve ao fato de que, ao receber *void* e retornar uma string de determinado tamanho, o método executa um processamento para criar a string com um número específico de caracteres. Por mais que os métodos considerem um mesmo conjunto padrão de operações, neste caso o tempo para o servidor criar uma string de 102.400 caracteres a ser retornada ao cliente é contemplado no tempo de execução. No método discutido em (B), por sua vez, o tempo começa a contar a partir do momento em que a chamada remota é realizada e, portanto, a string de 102.400 caracteres já foi gerada anteriormente.

Em relação ao cenário em que cliente e servidor estão sendo executados em máquinas diferentes, não foi observado um aumento tão significativo da média do tempo de execução em comparação ao cenário de mesma máquina. Pode-se observar que, independentemente do processamento mencionado, ao crescer de 1.024 para 102.400 caracteres, a proporção da média de tempo de execução na mesma máquina e em máquinas diferentes se manteve próxima: no gRPC, a média de tempo de execução para se retornar uma string de 102.400 caracteres foi 1862.56ms no cenário em que cliente e servidor estavam na mesma máquina, e 1938.11ms em máquinas diferentes. No caso do RMI, no primeiro cenário obteve-se uma média de tempo de execução de 1862ms, e em máquinas diferentes foi 1994.80ms. De maneira geral, para o servidor enviar ao cliente strings com tamanho equivalente a 102.400 caracteres, a média de tempo de execução de um cenário com mesma máquina para máquinas diferentes cresceu 5% no gRPC e 7% no RMI.

Esse resultado difere do que foi obtido para o método stringVoid discutido em (B). Neste caso, no gRPC a média de tempo de execução para enviar uma string de 102.400 caracteres foi 8.11ms no cenário em que cliente e servidor estavam na mesma máquina, e 158.22ms em máquinas diferentes. No caso do RMI, no primeiro cenário obteve-se uma média de tempo de execução de 5.22ms, e em máquinas diferentes foi 114.89ms. De maneira geral, para o cliente enviar ao servidor strings com tamanho equivalente a 102.400 caracteres, a média de tempo de execução de um cenário com mesma máquina para máquinas diferentes cresceu cresceu 1850% no gRPC e 2100% no RMI.

Trabalhando-se apenas com as proporções entre as médias em ambos os cenários, esse resultado pode indicar que o custo para um servidor mandar uma string de tamanho grande para um cliente é alto, independentemente se ambos estão na mesma máquina ou em máquinas diferentes, ao passo que o custo para um cliente mandar uma string de tamanho grande para o servidor é consideravelmente mais baixo se ambos estiverem na mesma máquina.

Comparando-se os mecanismos gRPC e RMI em termos de desempenho, observa-se que o RMI foi executado mais rapidamente para testes com strings de tamanho 10 e 1.024, nos dois cenários considerados (mesma máquina e máquinas diferentes). Para testes com strings de tamanho 102.400, por sua vez, no cenário de mesma máquina a média de tempo de execução se manteve bastante próxima entre ambos os mecanismos: no gRPC foi de 1862.56ms e no RMI 1862.00ms. Em máquinas diferentes, por sua vez, o RMI levou mais tempo para executar: 1994.80ms, em contraste aos 1938.11ms do gRPC.

1. **Métodos que recebem mais de um argumento**

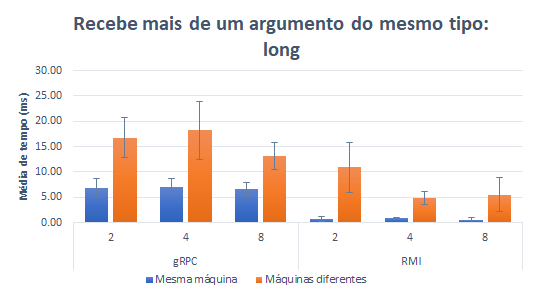
O quarto aspecto sob o qual o gRPC e o RMI foram avaliados refere-se ao teste de operações que recebem mais de um argumento. Para essa análise, foram criados dois tipos de métodos: os que recebem argumentos do mesmo tipo long, identificados por **longDoisArg, longQuatroArg** e **longOitoArg**, e os que recebem argumentos de tipos diferentes, identificados por **longStringDoisArg** (que envia uma string e um long, e recebe uma string)e **longStringQuatroArg** (que envia duas strings e dois longs, e recebe uma string).

A começar pela análise dos métodos que recebem argumentos do mesmo tipo, os resultados obtidos podem ser observados na Figura 7. Em relação ao mecanismo gRPC, observa-se que a média de tempo de execução dos três métodos foi bastante próxima: no cenário em cliente e servidor estavam na mesma máquina, foi em torno de 7ms, enquanto em máquinas diferentes variou de 13ms a 18ms.

Ainda sobre o gRPC, neste segundo cenário observa-se que a média do tempo de execução do método que recebe 8 argumentos do tipo long foi menor que a média dos demais métodos analisados. O método que apresentou a maior média do tempo de execução foi o que recebe 4 argumentos do tipo long, o qual também apresentou o maior desvio padrão.

Um comportamento semelhante foi observado com a utilização do mecanismo RMI: no cenário em que cliente e servidor estavam na mesma máquina, a média de tempo de execução foi em torno de 1ms, enquanto em máquinas diferentes variou de 5ms a 11ms. Neste último caso, o método que recebe 2 argumentos do tipo long demorou em média o dobro de tempo para execução em relação aos demais métodos, também apresentando o maior desvio padrão.

Para ambos os mecanismos, apesar de as médias de tempo de execução em máquinas diferentes aparentarem diminuir conforme aumenta o número de argumentos do método, não é possível afirmar essa hipótese com certeza, pois o desvio padrão para os cenários com tempo mais elevado é alto, especialmente para os métodos que apresentaram a maior média de tempo de execução. Ter um desvio padrão alto pode indicar a ocorrência de oscilações de rede durante as execuções, resultando em uma maior dispersão nos dados obtidos. Sendo assim, o que se pode concluir neste teste é que, para ambos os cenários e mecanismos, receber 2, 4 ou 8 argumentos do tipo long não impactou de forma significativa o tempo de execução.

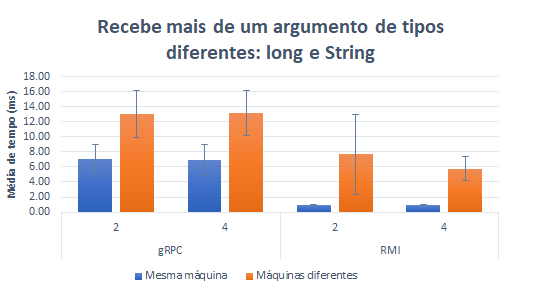
****

**Figura 7:** Testes em procedimentos que recebem mais de um argumento do tipo long

Ao analisar os resultados obtidos para os métodos que recebem mais de um argumento de tipos diferentes, também não foram observadas diferenças expressivas, conforme pode ser observado na Figura 8. O gRPC continuou com tempo de execução médio próximo a 7ms no cenário em que cliente e servidor estavam na mesma máquina e de 13ms em máquinas diferentes, enquanto no RMI manteve-se o tempo de execução médio próximo a 1ms no cenário em que cliente e servidor estavam na mesma máquina, e em torno de 6ms em máquinas diferentes.

Ao comparar esses resultados com os dos métodos com 2 e 4 argumentos do mesmo tipo, apresentados na Figura 7, observa-se que no cenário em que cliente e servidor estavam na mesma máquina não houveram diferenças significativas no tempo de execução, mantendo-se as médias em torno dos mesmos valores. No cenário com máquinas diferentes, por sua vez, em ambos os mecanismos observou-se em geral uma diminuição da média de tempo de execução. No gRPC, a média diminuiu de 16.78ms e 18.20ms (para 2 e 4 argumentos, respectivamente) para 13.10ms e 13.22ms; já no RMI, a média diminuiu de 10.89ms e 4.89ms (para 2 e 4 argumentos, respectivamente) para 7.67ms e 5.78ms. Neste último caso, comparando-se métodos com execução remota via RMI com 4 argumentos de mesmo tipo e tipos diferentes, a média de tempo de execução dos métodos com tipos diferentes foi de fato maior.

Infelizmente, não foram realizados no presente trabalho testes com métodos que recebem 8 argumentos de tipos diferentes, inviabilizando uma comparação equivalente com os resultados obtidos e apresentados nas Figuras 7 e 8. Porém, até onde nos é possível analisar, não foi observada uma diferença significativa na média de tempos de execução considerando-se métodos com mais de um argumento de mesmo tipo e tipos diferentes, considerando-se testes com 2, 4 e 8 argumentos. As diferenças mais significativas foram observadas no cenário em que cliente e servidor estavam em máquinas diferentes, no qual também se apresentou um desvio padrão mais alto e, portanto, indicando uma possível oscilação na rede e maior dispersão dos resultados. Como os testes foram realizados com um número baixo de argumentos, não é possível concluir uma associação direta entre o aumento do número de argumentos e o aumento da média de tempo de execução. Caso outros testes fossem realizados com um número maior de argumentos, é possível que essa associação se mostrasse de maneira mais evidente.

****

**Figura 8:** Testes em procedimentos que recebem mais de um argumento dos tipos long e String

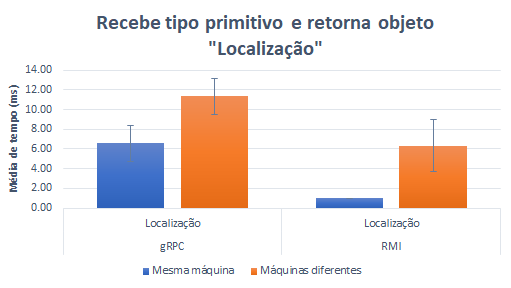
Em comparação com os resultados obtidos com a entrada e saída de apenas um argumento, apresentados no item (A), novamente não pode-se afirmar que houveram diferenças expressivas em relação ao tempo de execução de métodos com mais de uma variável, independente se o método recebe argumentos do tipo long ou string.

Para ambos os testes discutidos nesta seção, ao comparar os mecanismos gRPC e RMI em termos de desempenho, pode-se observar que os métodos do RMI foram executados mais rapidamente nos dois cenários considerados - mesma máquina e máquinas diferentes.

1. **Métodos que retornam tipos complexos**

O quinto e último aspecto sob o qual o gRPC e o RMI foram avaliados refere-se ao teste de operações que retornam um tipo complexo, que se refere a um objeto instanciado de uma classe específica. Para este teste, foi criada a classe “Localizacao”, que possui em sua estrutura três atributos, dois do tipo int chamados *latitude* e *longitude*, e um do tipo string chamado *mensagem*. Os únicos métodos da respectiva classe referem-se ao construtor e aos métodos de acesso público aos dados (*getters* e *setters*).

Para essa análise, implementou-se no servidor um método chamado **LocMsg**, que recebe dois argumentos int e uma string e retorna um objeto complexo do tipo “Localizacao”, sendo os atributos do objeto correspondentes aos valores recebidos como argumento. Os resultados obtidos podem ser observados na Figura 9.

****

**Figura 9:** Testes em procedimentos que recebem tipo primitivo e retornam objeto

Novamente, em nenhum dos cenários foi identificada uma diferença significativa no tempo de execução. Em relação ao gRPC, no cenário em que cliente e servidor estão na mesma máquina, a média de tempo de execução foi 6.56ms, e em máquinas diferentes 11.33ms. Já no caso do RMI, no cenário em que cliente e servidor estão na mesma máquina, a média de tempo de execução foi em torno de 1ms, enquanto em máquinas diferentes foi 6.33ms.

As características observadas na Figura 9 não diferem de maneira significativa do que foi apresentado no demais testes com argumentos que retornam o tipo string, long e int: em geral, a média de tempos de execução não aumentou significativamente ao se considerar um tipo complexo. Esse resultado era, de certa forma, esperado ao se utilizar como teste uma classe que contempla um número baixo de atributos do tipo int e string - na prática, o objeto apenas encapsulou atributos de tipos já considerados anteriormente. Caso se considerasse na classe “Localizacao” um maior número de atributos de tipos mais complexos, como por exemplo, ArrayList ou LinkedList, possivelmente a média de tempos de execução se mostraria diferente em relação aos demais testes.

No mecanismo gRPC em especial, o fato de a média de tempo de execução não se alterar em relação aos demais métodos analisados já esperado: a estrutura do mecanismo implica que mensagens de tipo primitivo devam ser sempre encapsuladas por um tipo complexo, usando-se um bloco de código do tipo *message* no arquivo .proto. Na prática, no caso do gRPC, este e todos os outros métodos realizados usando-se tipos primitivos consistiram em um objeto sendo passado e/ou recebido como argumento.

Infelizmente, nesta seção só foram realizados testes retornando-se um objeto como argumento e, portanto, as análises realizadas só são aplicáveis neste cenário. Testes futuros podem ser realizados considerando-se métodos que recebem tipos complexos como argumento, visando-se verificar se o mesmo comportamento é observado.

Ao comparar os mecanismos gRPC e RMI em termos de desempenho, pode-se observar que os métodos do RMI foram executados mais rapidamente nos dois cenários considerados - mesma máquina e máquinas diferentes.

1. **Considerações finais**

Neste relatório, foram apresentados os resultados de uma avaliação de desempenho conduzida sob dois mecanismos de chamada de procedimento remoto: gRPC e Java RMI. São três os objetivos dessa avaliação: *(i)* entender aspectos e estruturas de programação que podem impactar no desempenho de chamadas remotas; *(ii)* entender e comparar os custos de desempenho associados à execução de procedimentos remotos, estando o cliente e servidor na mesma máquina e em máquinas diferentes; e *(iii)* comparar o desempenho dos dois mecanismos em análise, visando-se verificar qual deles mostrou-se ser mais eficiente.

Para cumprir com esses objetivos, foram implementados trinta métodos com diferentes combinações de parâmetros de entrada e saída, os quais permitiram-nos analisar e comparar o desempenho de chamadas remotas considerando-se argumentos do tipo int, long, string e Localizacao, uma classe criada para testar tipos complexos. Os métodos foram implementados de maneira equivalente em ambos os mecanismos, e foram executados considerando-se cliente e servidor na mesma máquina e em máquinas diferentes.

De maneira geral, a média de tempo de execução dos métodos em máquinas diferentes foi maior em todos os testes, sendo no mínimo 3ms a mais do que nos testes em máquinas iguais. Essa diferença evidencia a influência da latência da rede no tempo de execução de chamadas remotas em computadores distintos. O desvio padrão dos dados obtidos também mostrou ser superior em testes com máquinas diferentes, mostrando que oscilações na rede podem ter influenciado os tempos de execução obtidos nesse cenário.

Comparando-se o desempenho do gRPC com o RMI, observa-se que as chamadas remotas usando-se RMI foram executadas mais rapidamente em praticamente todos os testes. Uma justificativa para essa diferença pode estar relacionada ao fato de que a estrutura do gRPC usa o protocolo HTTP para as requisições de chamada remota, tornando o processo de empacotamento e desempacotamento de parâmetros em mensagens (*marshalling* e *unmarshalling*)mais complexo. Além disso, chamadas remotas no gRPC não podem ser feitas passando-se tipos primitivos como argumento. Nestes casos, deve-se criar no arquivo .proto um bloco de código do tipo *message* para definir uma classe com variáveis do tipo int ou long. Dessa forma, os ganhos com a simplicidade dos tipos primitivos acabam sendo perdidos ao se utilizar uma classe com tipos primitivos como atributo.

O único caso em que a média de tempo de execução com o RMI foi superior ao gRPC refere-se ao método voidString para o tamanho 102.400, em que o cliente não manda nenhum argumento (void) e o servidor retorna uma string de 102.400 caracteres. Nesse teste, o tempo de execução usando-se RMI foi 3% maior do que com gRPC. Em outro método similar, stringVoid, para o teste em que o cliente manda uma string de 102.400 caracteres e não recebe nenhum argumento (void), o tempo de execução usando gRPC foi apenas 7% maior do que com RMI, diferindo significativamente de testes com outros tamanhos em que a diferença entre os tempos de ambos os mecanismos chegou a ser superior a 100%.

Com esses resultados, observa-se que a diferença na média de tempos de execução entre o gRPC e RMI em testes com strings a partir de 1.024 caracteres parece diminuir. Como os testes foram realizados apenas com strings de no máximo 102.400 caracteres, estudos mais aprofundados são necessários para evidenciar essa tendência e identificar exatamente em qual tamanho os tempos começam a convergir.

Dessa maneira, até onde os dados permitem-nos concluir, o desempenho do RMI mostrou-se superior para métodos que usam argumentos do tipo int, long, tipos complexos como “Localizacao” e strings de até 1.024 caracteres. No entanto, à medida em que se utilizam strings de tamanho significativamente maior nas chamadas remotas, a diferença no desempenho de ambos os mecanismos parece diminuir consideravelmente, com o gRPC mostrando-se ser superior para chamadas em que o cliente recebe uma string muito grande do servidor.

Apesar das diferenças em termos de desempenho entre ambos os mecanismos, os comportamentos observados em relação a estruturas e tipos exercitados foram os mesmos. Em relação aos tipos int e long, os resultados obtidos mostram que não há diferença significativa em termos de desempenho considerar int ou long como tipo de argumento em chamadas remotas. Nos testes em que servidor e cliente estavam em máquinas diferentes, a média de tempo de execução de métodos com int foi ligeiramente maior do que com long; porém, nestes mesmos casos, o desvio padrão observado também foi maior, o que indica uma maior dispersão dos dados e, possivelmente, maior influência da latência da rede nos resultados obtidos.

Também não foi identificada uma diferença expressiva entre métodos que passam e recebem argumentos de mesmo tipo, passam um argumento de determinado tipo e recebem void, ou mandam void e recebem um argumento de determinado tipo. Métodos que não mandam nem recebem argumento (void) tiveram seu tempo de execução próximo àqueles que fazem uso de argumentos do tipo int ou long, mostrando-se não haver grandes diferenças entre não passar nada ou passar argumentos do tipo int ou long em chamadas remotas.

Em relação ao tipo string, foi observado um aumento da média do tempo de execução de chamadas remotas ao se utilizar strings a partir de 256 caracteres. Para strings de 1 a 128 caracteres, crescer em tamanho ainda não se mostrou ser tão significativo a ponto de afetar o desempenho da chamada remota. No entanto, a partir de 256 caracteres, observou-se que a curva de crescimento já se tornara um pouco mais acentuada, impactando mais significativamente no tamanho e, consequentemente, no tempo de execução. Essa hipótese pôde ser confirmada ao se realizar testes com strings de 102.400 caracteres, nos quais se observou um crescimento bastante significativo na média de tempos de execução.

O desempenho de chamadas remotas em testes com strings de tamanho superior a 256 caracteres foi mais afetado quando cliente e servidor estão em máquinas diferentes. Esse resultado sugere que, quanto maior o tamanho da string, maior o tempopara que os dados saiam da máquina do servidor e cheguem à máquina do cliente e, portanto, maior o tempo para conclusão da chamada remota.

Ao se comparar o método em que o cliente envia uma string de no máximo 102.400 caracteres e não recebe nada como resultado (void), com o método em que o cliente não manda nenhum argumento (void) e recebe uma string de também no máximo 102.400 caracteres, observou-se que a média do tempo de execução foi superior no segundo caso, indicando que receber uma string de tamanho grande custou mais do que enviá-la. Esse resultado já era esperado pois, ao receber *void* e retornar uma string de determinado tamanho, o método deve executar um processamento para criar a string com um número específico de caracteres, o que afeta o tempo de execução.

No entanto, independente do tempo de processamento, a curva de crescimento da média do tempo de execução em máquinas iguais para diferentes se mostrou muito mais acentuada no primeiro método do que no segundo. Esse resultado pode indicar que o custo para um servidor mandar uma string de tamanho grande para um cliente é alto, independentemente se ambos estão na mesma máquina ou em máquinas diferentes, ao passo que o custo para um cliente mandar uma string de tamanho grande para o servidor mostrou-se ser consideravelmente mais baixo se ambos estiverem na mesma máquina.

Para testes com mais de um argumento do mesmo tipo ou tipos diferentes, não foi observada uma diferença significativa na média de tempos de execução considerando-se testes com 2, 4 e 8 argumentos. As diferenças mais significativas foram observadas no cenário em que cliente e servidor estavam em máquinas diferentes, no qual também se apresentou um desvio padrão mais alto e, portanto, indicando uma possível oscilação na rede e maior dispersão dos resultados. Em comparação com os resultados obtidos com a entrada e saída de apenas um argumento, novamente não foram identificadas diferenças significativas, mostrando-se não haver grandes diferenças entre passar 1 ou 8 argumentos. Caso outros testes fossem realizados com um número maior de argumentos, é possível que essa associação fosse mais evidente.

Por fim, para testes com métodos que retornam tipos complexos, as características observadas também não diferem de maneira significativa do que foi apresentado no demais testes com argumentos que retornam o tipo string, long e int: em geral, a média de tempos de execução não aumentou significativamente ao se considerar um tipo complexo.

Os resultados obtidos e apresentados neste relatório podem ter sido influenciados pela forma com a qual as execuções foram medidas, pelo fato de a comunicação entre cliente e servidor ter sido estabelecida através de uma rede wireless e, portanto, ser mais suscetível à latência da rede no momento da execução dos testes, e pela metodologia conduzida para filtrar os dados e desconsiderar os valores extremos nos resultados.

Neste contexto, estudos futuros podem ser conduzidos considerando-se um maior número de execuções dos métodos considerados, visando-se obter valores mais representativos para as médias. Além disso, para testes em que o cliente e o servidor estão em máquinas diferentes, seria mais apropriado estabelecer a comunicação usando-se uma rede dedicada cabeada ao invés da rede wireless, para que a interferência da latência da rede tivesse menos impacto nos resultados. Por fim, testes adicionais com strings de tamanhos superiores a 1.024 caracteres, número de argumentos superior a 8 e recebendo-se objetos como argumento podem ser realizados para confirmar as tendências observadas no presente trabalho.

1. **Referências bibliográficas**

[1] Andrew S. Tanenbaum and Maarten van Steen. 2017. Distributed Systems (3rd Edition). Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[2] gRPC Basics - Java. Disponível em: <<https://grpc.io/docs/tutorials/basic/java.html>>.

[3] Java Remote Method Invocation API (Java RMI). Disponível em: <<https://docs.oracle.com/javase/7/docs/technotes/guides/rmi/>>.