UNIVERSIDADE DE SÃO PAULO

ACH 2147 - Desenvolvimento de Sistemas de Informação Distribuídos

EXERCÍCIO PROGRAMA

REMOTE PROCEDURE CALLS (RPC)

Docente: Prof. Dr. Daniel de Angelis Cordeiro

Grupo:

Leonardo Colman Lopes **9875490**

Rafael Galrão Silveira **3495012**

São Paulo
06 de julho de 2018

SUMÁRIO

1. INTRODUÇÃO

1.1. OBJETIVO

O objetivo deste trabalho foi avaliar e comparar o desempenho de implementações diferentes de mecanismos de chamadas de procedimentos remotos (RPC - Remote Procedure Call). A comparação foi baseada no tempo em que cada RPC levava para executar operações básicas (detalhamento na seção 2 e subseções).

1.2. RPCs

Na computação distribuída, uma chamada de procedimento remoto (RPC) ocorre quando um programa de computador faz com que um procedimento (sub-rotina) seja executado em um espaço de endereço diferente (normalmente em outro computador em uma rede compartilhada), codificado como normal (local), sem que o programador codifique explicitamente os detalhes da interação remota. Ou seja, o programador escreve essencialmente o mesmo código, seja a sub-rotina local ou remota.

De modo geral, os passos seguidos por um RPC são:

- O cliente chama seu stub. É uma chamada de procedimento local, com os parâmetros colocados na pilha da maneira normal.
- O stub do cliente empacota os parâmetros em uma mensagem e faz uma chamada de sistema para enviar a mensagem. Empacotar os parâmetros é chamado de “marshalling”.
- O sistema operacional local do cliente envia a mensagem da máquina do cliente para a máquina do servidor.
- O sistema operacional local na máquina do servidor passa os pacotes recebidos para o stub do servidor.
- O stub do servidor desempacota os parâmetros da mensagem. Desempacotar os parâmetros é chamado de “unmarshalling”.
- Finalmente, o stub do servidor chama o procedimento do servidor. A resposta traça os mesmos passos na direção oposta.

2. MÉTODOS, MATERIAIS E PROCEDIMENTOS

2.1. RPCs UTILIZADOS

2.1.1. gRPC

Por pré-determinação, o RPC padrão utilizado para comparações será o gRPC. O gRPC é um sistema de chamada de procedimento remoto (RPC) de código aberto desenvolvido inicialmente no Google. Ele usa HTTP/2 para transporte, Protocol Buffers como a linguagem de descrição da interface e fornece recursos como autenticação, fluxo bidirecional e controle de fluxo, ligações bloqueantes ou não-bloqueantes e cancelamento e tempos de espera. Ele gera conexões inter-plataformas entre cliente e servidor para vários idiomas.

2.1.2. Apache Avro

O Avro é um RPC e um framework de serialização de dados desenvolvido dentro do projeto Hadoop do Apache. Ele usa JSON para definir tipos de dados e protocolos e serializa os dados em um formato binário compacto. Seu uso principal é no Apache Hadoop, onde pode fornecer tanto um formato de serialização para dados persistentes, quanto um formato de conexão para comunicação entre nós do Hadoop, quanto de programas clientes para os serviços do Hadoop.

O Avro utiliza esquemas. Quando os dados do Avro são lidos, o esquema usado ao escrevê-lo está sempre presente. Isso permite que cada dado seja escrito sem overheads por valor, tornando a serialização rápida e pequena. Isso também facilita o uso com linguagens de script dinâmicas, já que os dados, juntamente com seu esquema, são totalmente auto-descritivos.

Quando os dados do Avro são armazenados em um arquivo, seu esquema é armazenado junto, para que os arquivos possam ser processados posteriormente por qualquer programa. Se o programa que lê os dados espera um esquema diferente, isso pode ser facilmente resolvido, uma vez que ambos os esquemas estão presentes.

Quando o Avro é usado no RPC, o cliente e o servidor trocam os esquemas no handshake da conexão. (Isso pode ser otimizado para que, na maioria das chamadas, nenhum esquema seja realmente transmitido.) Como tanto o cliente quanto o servidor possuem o esquema completo do outro, a correspondência entre os mesmos campos nomeados, campos ausentes, campos extras etc. pode ser facilmente resolvida.

2.2. MÉTODOS

A análise foi feita em duas etapas. Primeiro, cada RPC foi avaliado individualmente em 2 cenários:

1) Quando cliente e servidor estão em uma mesma máquina
2) Quando cliente e servidor estão em máquinas diferentes

Para cada RPC, foi avaliado o tempo gasto para cada uma das operações básicas descritas abaixo:

a) Chamada de uma operação sem argumento e sem valor de retorno (void).
b) Chamada de uma operação com **1** argumento do tipo *long* e valor de retorno *long* (singlelong)
c) Chamada de uma operação com **8** argumentos do tipo *long* e valor de retorno *long* (eightlongs)
d) Chamada de uma operação com argumentos do tipo *String* e valor de retorno *String*. Os tamanhos da String variaram em potência de 2, ou seja, 1, 2, 4, 8, 16, 32, 64, 128, 256, 512 e 1024 caracteres (string**x**, onde **x** varia de acordo com a potência de 2).
e) Chamada de uma operação com argumento do tipo complexo e valor de retorno do tipo complexo (foo).

Nosso tipo complexo consistiu de um objeto com um identificador do tipo *long*, uma *String* de título, até 6 textos de body, cada um com até mil caracteres, de 1 a 20 autores. Além disso, cada autor possui um identificador do tipo *long*, uma *String* de nome e uma *String* de sobrenome.

Modelo em Proto:

```
13  message Foo {
14      int64 id = 1;
15      string title = 2;
16      repeated string body = 3;
17      repeated Author authors = 4;
18  }
19
20  message Author {
21      int64 id = 1;
22      string name = 2;
23      string surname = 3;
24
25  }
```

2.3. MATERIAIS

Os experimentos foram executados em um PC comum e em um servidor (Amazon EC2) da Amazon Web Services (AWS), ambos descritos detalhadamente nas próximas seções.

2.3.1. Hardware

LOCAL:

CPU

- AMD FX(tm)-8300 Eight-Core Processor

Memória

- 8 MB Cache L2
- 8 MB Cache L3
- 4 x 4GB DIMM DDR3 Synchronous 1600 MHz (Totalizando 16GB de RAM)

AMAZON EC2:

CPU

- 1 vCPU de processador Intel Xeon de Alta frequência (infelizmente, AWS não provê informações mais específicas)

Memória

- 1 GB de RAM

2.3.2. Sistema Operacional

LOCAL:

- Ubuntu 18.04 LTS

AMAZON EC2:

- Ubuntu 16.04 LTS

2.3.3. Rede

Para os casos em que cliente e servidor estão em máquinas diferentes, utilizamos internet Ethernet conectada diretamente no roteador da Vivo Fibra 100mbps.

2.3.4. Bibliotecas e Linguagens de Programação Utilizadas

Utilizamos a IDE IntelliJ (JetBrains) Ultimate e a linguagem de programação Kotlin para o desenvolvimento dos algoritmos.

Bibliotecas:

- Java JDK 1.8 (apenas para classes do Protobuf/Avro, geradas por outras bibliotecas)
- Kotlin Standard System Library versão 1.2.50
- Plugin Gradle - Google Protobuf versão 0.8.6
- Google API GRPC Common Protos versão 0.1.9
- GRPC-Netty (Servidor) versão 1.5.0
- GRPC Protobuf versão 1.5.0
- GRPC Stub versão 1.5.0
- Apache Avro versão 1.8.2
- Apache Avro IPC versão 1.8.2
- Protoc (Compilador de Arquivos Proto) versão 3.3.0
- Protoc-Gen-GRPC-Java (Compilador de Proto para GRPC Java) versão 1.5.0

2.4. PROCEDIMENTOS

Cada experimento foi repetido 100 (cem) vezes, sendo o menor e o maior valor descartados. As médias e desvios-padrão foram calculados diretamente através do algorítmo e estão descritas na seção 3.1.

Para o cálculo do desvio-padrão, utilizamos a seguinte fórmula:

$$s_N = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \overline{x})^2},$$

Onde: $\{x_1, x_2, x_3, \ldots, x_i\}$ são os valores observados, $\bar{x}$ é a média dos valores observados e N é o tamanho da amostra.

Também utilizamos o cálculo de Coeficiente de Variação (Desvio Padrão Relativo) para verificar a dispersão dos valores, ou seja, para saber se nosso desvio-padrão era "alto" ou não:

$$Cv = \frac{s}{\bar{x}}$$

Onde *Cv* é o coeficiente de variação, *s* é o desvio-padrão e $\bar{x}$ é a média dos valores encontrados.

3. RESULTADOS E DISCUSSÕES

3.1. RESULTADOS

Os resultados dos experimentos (médias e desvios-padrão) estão descritos nas tabelas abaixo. Todos os tempos foram medidos em milissegundos.

Tabela 1: gRCP com cliente e servidor na mesma máquina

	Média (ms)	Desv Pad	CV
void	3.235	1.215	0.376
singlelong	2.827	1.217	0.430
eightlongs	2.612	0.879	0.337
string1	2.327	0.892	0.383
string2	2.204	0.707	0.321

string4	2.184	0.876	0.401
string8	2.112	0.870	0.412
string16	1.980	0.883	0.446
string32	1.918	0.783	0.408
string64	1.980	0.848	0.428
string128	1.827	0.825	0.452
string256	1.827	1.132	0.620
string512	2.000	1.148	0.574
string1024	1.724	0.846	0.491
foo	3.306	2.539	0.768

Tabela 2: gRCP com cliente e servidor em máquinas diferentes

	Média (ms)	Desv Pad	CV
void	182.684	4.612	0.025
singlelong	178.745	3.659	0.020
eightlongs	196.908	2.316	0.012
string1	177.265	2.594	0.015
string2	196.592	1.684	0.009
string4	196.776	2.003	0.010
string8	199.816	4.444	0.022
string16	178.704	3.359	0.019
string32	196.806	1.842	0.009

string64	177.255	1.976	0.011
string128	181.265	4.595	0.025
string256	178.163	3.208	0.018
string512	181.531	4.557	0.025
string1024	177.857	2.665	0.015
foo	187.092	2.522	0.013

Tabela 3: Avro com cliente e servidor na mesma máquina

	Média (ms)	Desv Pad	CV
void	1.133	0.658	0.581
singlelong	0.939	0.486	0.518
eightlongs	0.969	0.457	0.472
string1	0.969	1.100	1.135
string2	0.878	0.381	0.434
string4	0.602	0.505	0.839
string8	0.510	0.552	1.082
string16	0.480	0.495	1.031
string32	0.429	0.490	1.142
string64	0.571	0.548	0.960
string128	0.561	0.491	0.875
string256	0.520	0.495	0.952
string512	0.510	0.495	0.971
string1024	0.551	0.512	0.929
foo	1.561	1.668	1.069

Tabela 4: Avro com cliente e servidor em máquinas diferentes

	Média (ms)	Desv Pad	CV
void	177.224	3.132	0.018
singlelong	177.153	2.099	0.012
eightlongs	177.031	1.791	0.010
string1	196.796	2.679	0.014
string2	176.806	4.447	0.025
string4	176.316	2.398	0.014
string8	176.510	3.077	0.017
string16	176.939	1.896	0.011
string32	189.908	4.806	0.025
string64	177.184	2.304	0.013
string128	196.622	1.308	0.007
string256	196.520	1.235	0.006
string512	187.194	2.468	0.013
string1024	186.684	1.730	0.009
foo	181.735	11.231	0.062

3.2. DISCUSSÕES

Algo que se nota ao olharmos diretamente para os dados das tabelas acima, é que quando cliente e servidor estão na mesma máquina o tempo entre o envio e a resposta é muito menor.

Fica evidente que uma conexão local, em termos de tempo de resposta, é muito mais eficiente que conectar-se a uma outra máquina, especialmente considerando que o nosso servidor da AWS está localizado em Ohio, leste dos EUA.

Com relação aos dois tipos de RCP, ao compará-los obtivemos:

Gráfico 1: gRPC versus Avro quando cliente e servidor estão na mesma máquina

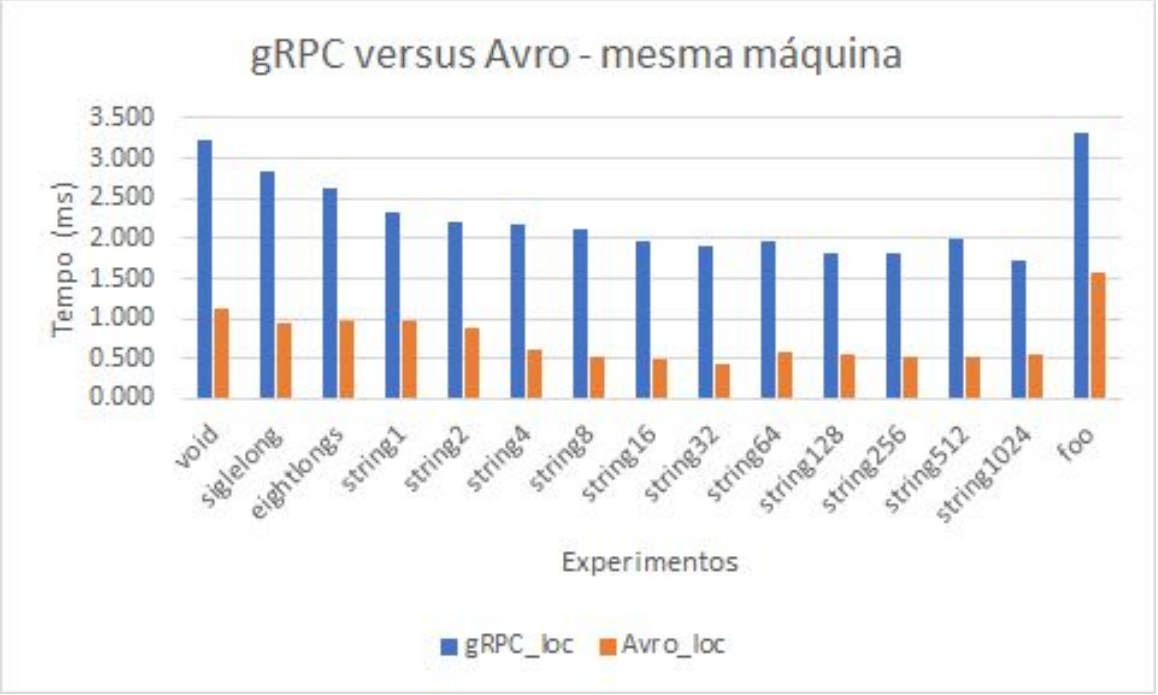
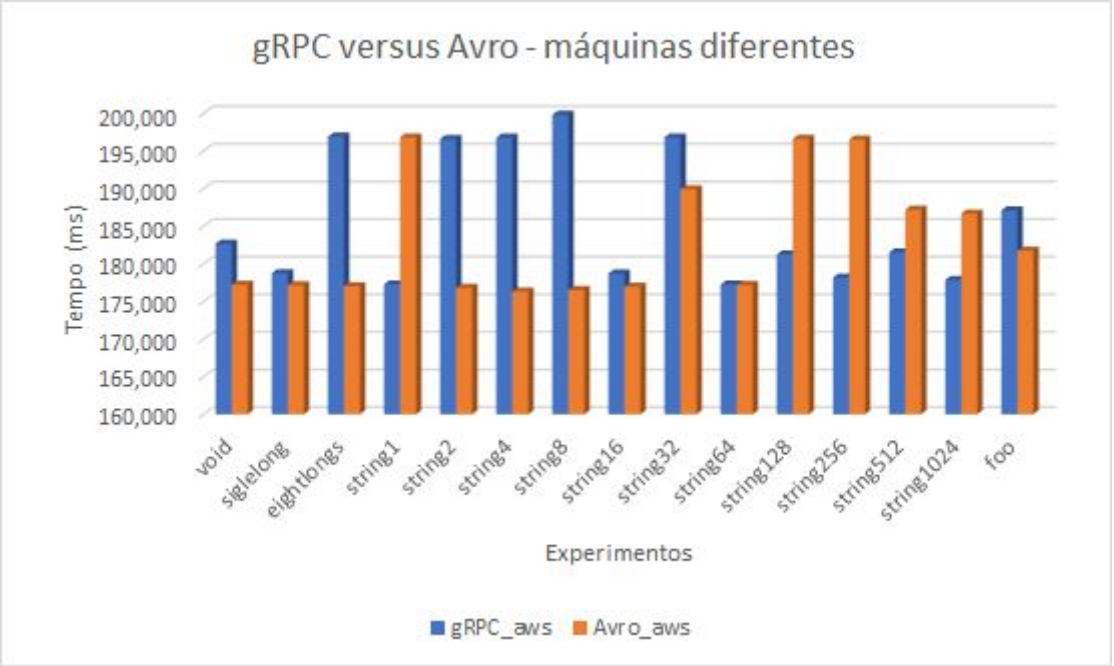


Gráfico 2: gRCP versus Avro quando cliente e servidor estão em máquinas diferentes



Nota-se que, quando cliente e servidor estão na mesma máquina, ambos os RPCs apresentam um alto desvio-padrão (Tabelas 1 e 3). Isso se deve, provavelmente, aos baixos tempos observados, ou seja, os tempo são tão pequenos que qualquer variação gera um grande desvio.

Além disso, com cliente e servidor na mesma máquina, o Avro foi mais eficiente, com tempos de envio e retorno mais baixos que o gRPC (Gráfico 1). Considerando uma média geral, o Avro foi aproximadamente 4x mais rápido que o gRPC.

Já com cliente e servidor em máquinas diferentes, observamos que os RPCs se comportam de forma diferente, dependendo do experimento, conforme os dados do Gráfico 2:

- Nos experimentos com void e longs, o Avro foi, em média, mais eficiente que o gRCP
- Nos experimentos com strings, quando são maiores (string128, string256, string512 e string1024) o gRPC foi, em média, mais eficiente que o avro.
- Já nos experimentos com strings menores, o Avro parece ser mais eficiente (string2, string4, string8, string16 e string32)
- Curiosamente, quando a string é pequena (string1), nota-se que o gRPC é mais eficiente do que o Avro.

- Finalmente, com relação ao objeto complexo (foo), O Avro também foi mais eficiente que o gRCP. Esse último comportamento é estranho para nós, dado que o nosso tipo complexo possui várias Strings com tamanho maior do que 1024. Acreditamos que isso ocorre pois o Protobuf/gRPC leva muito tempo para compactar a mensagem em uma forma mais eficiente de enviar à rede, e não ao tempo de trânsito da mensagem em si.

4. CONCLUSÕES

Os experimentos executados mantém um padrão, ou seja, foram executados numa mesma máquina ou nas mesmas máquinas em uma mesma rede. Isso eliminou possíveis interferências externas e nos permitiu encontrar resultados mais confiáveis.

Como nosso servidor encontra-se em Ohio - EUA, fica bem evidente que a distância geográfica influencia muito no tempo de resposta de um RPC, dado que a mensagem precisa viajar até lá e a resposta viajar de volta. Escolhemos um servidor mais distante pois isso nos permitiu ver com maior clareza a diferença de desempenho entre os dois RPCs durante os experimentos.

O desvio-padrão dos experimentos executados localmente é alto, mas isso pode ser explicado pela medida de tempo, feita em milissegundos. Como essa unidade de tempo é pequena (em potência de 10), qualquer variação causaria um desvio alto. Portanto, foi de grande importância repetir os experimentos e descartar os valores extremos para que o conjunto de dados apresentasse maior significado.

De acordo com os resultados, o Avro tem melhor desempenho geral, tanto em redes locais quanto à distância. Ficou atrás somente em strings com tamanhos maiores, o que nos faz pensar que o ganho de performance do Avro faz com que, em momentos específicos, ele perca dos outros RPCs em desempenho. Dito de outra forma, o Avro é, em média, melhor do que o gRPC, perdendo apenas em casos específicos.

5. REFERÊNCIAS

Arpaci-Dusseau, Remzi H.; Arpaci-Dusseau, Andrea C. (2014), *Introduction to Distributed Systems* (PDF), Arpaci-Dusseau Books.

gRCP - A high performance, open-source universal RPC framework - Website disponível em: https://grpc.io/ - Acesso em: 04-Jul-2018.

Apache Avro 1.8.2 Documentation - Website disponível em http://avro.apache.org/docs/current/ - Acesso em: 04-Jul-2018.

Saporta, Gilbert (2006). Probabilités – Analyse des Données et Statistiques. Paris: Éditions Technip. p. 279 – 280. 622 páginas

Infraestrutura Global da AWS - Informações disponíveis em: https://aws.amazon.com/pt/about-aws/global-infrastructure/ - Acesso em 04-Jul-2018.