Projeto 3 - MC833 - Programação em Rede de Computadores

Prof. Dr. Edmundo Roberto Mauro Madeira

Eduardo Moreira Freitas de Souza - RA: 166779

1. Introdução

O objetivo deste projeto foi o desenvolvimento de uma aplicação cliente-servidor, através da API de Java RMI (Java Remote Method Invocation), que se utiliza do método TCP.

O servidor armazena um banco de dados com entradas geradas aleatoriamente, que representam pessoas com as seguintes informações:

- Email;
- Senha de acesso;
- Nome completo;
- Residência;
- Formação Acadêmica;
- Habilidades;
- Experiência;

2. Servidor

O servidor foi feito Java, com acesso ao banco de dados MongoDB[1] através da biblioteca MongoDB Java Driver[2]. Assim que recebe uma requisição de um cliente para se conectar à interface remota através da porta especificada, se mantém conectado através de uma implementação do protocolo TCP, que conecta a implementação da classe do servidor com a interface do cliente.

Em seguida, encontra-se com sete possíveis operações:

(1) Listar todas as pessoas formadas em um determinado curso;
(2) Listar as habilidades dos perfis que moram em uma determinada cidade;
(3) Acrescentar uma nova experiência no próprio perfil;
(4) Dado o email do perfil, retornar sua experiência;
(5) Listar todas as informações de todos os perfis;
(6) Dado o email de um perfil, retornar suas informações;
(7) Desconecta em qualquer momento do servidor;

Cada uma dessas operações foram implementadas como métodos em uma classe que é instanciada no servidor e cada uma retorna uma string com as informações desejadas.

3. Estrutura de dados armazenados no servidor

A estrutura de dados do MongoDB foi montada através de um docker-compose, que popula o banco com um arquivo .json, no seguinte formato:

```
[
	{
	'Nome Completo' : { 'nome' : 'Lucas', 'sobrenome' : 'Teixeira Amaral' },
```

```
     'Email' : 'lucas.amaral@globo.com',
     'senha' : 'plaintext1',
     'Residencia' : 'Pereiras',
     'Formacao Academica' : 'Psicologia',
     'Habilidades' : 'Parapsicologia',
     'Experiencia' : [
          {
               'Local' : 'Sefit',
               'Cargo' : 'Estagiario'
          },
          {
               'Local' : 'Bolotus',
               'Cargo' : 'Psicologo'
          }
     ]
     },
     …
]
```

Esse objeto JSON é lido pelo servidor em Java como um Objeto genérico em uma lista de resultados de uma determinada consulta - lista essa chamada de Cursor.

Após uma possível concatenação entre resultados, o cliente recebe um objeto json, fruto da consulta requisitada pela sua operação, no seguinte formato:

```
Sua opcao: 1
Escreva o curso que deseja saber: Psicologia
{ "Nome Completo" : { "nome" : "Lucas", "sobrenome" : "Teixeira
Amaral" } }{ "Nome Completo" : { "Nome" : "Pedro", "Sobrenome" :
"Ducati Amaral" } }{ "Nome Completo" : { "Nome" : "Pedro",
"Sobrenome" : "Ducati Amaral" } }{ "Nome Completo" : { "nome" :
"Lucas", "sobrenome" : "Teixeira Amaral" } }
```

4. Implementação do servidor RMI

Para a comunicação entre o cliente e o servidor foi utilizada a API RMI de Java, que fornece um simples método para que torne possível o cliente executar métodos de um servidor remoto. Desse modo, é criada uma interface, chamada `ProcessRequestInterface`, que estende a interface `Remote` nativa do java, definindo os métodos para serem chamados remotamente. Os métodos estão listados a seguir, em ordem das tarefas enumeradas na Introdução:

```
public interface ProcessRequestInterface extends Remote {
```

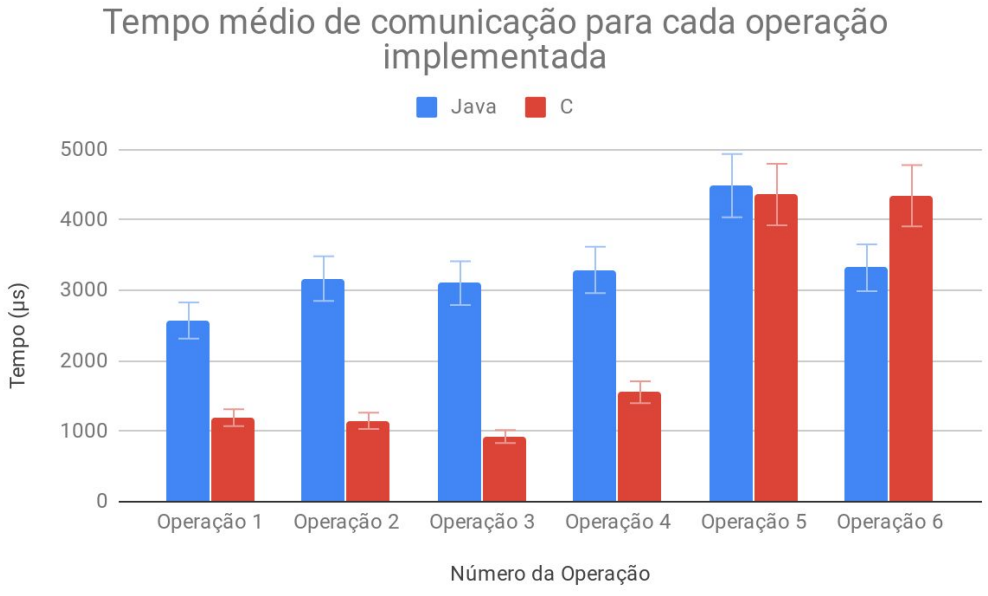
```
	public String list_name_course(String course) throws
RemoteException;
	public String list_hab_city(String city) throws
RemoteException;
	public String add_exp(String email, String work_location,
String job) throws RemoteException;
	public String list_exp_email(String email) throws
RemoteException;
	public String list_all() throws RemoteException;
	public String list_all_email (String email) throws
RemoteException;
}
```

Essa classe é instanciada e realizada o vínculo com a *socket* correspondente em uma outra classe chamada Servidor, que apenas executa tal preparação para a requisição do cliente - deixando, então, seus métodos disponíveis para tal.

5. Tempo

O tempo foi coletado tanto por parte do servidor quanto por parte do cliente, em etapas distintas do fluxo do programa.

No servidor, o primeiro tempo salvo é logo após que recebe a mensagem, e o segundo, logo quando envia a mensagem - ou seja, o tempo de processamento dos dados para o envio. No cliente, o tempo é contabilizado entre enviar a mensagem e receber uma resposta, ou seja, o tempo de espera do processamento e do envio do pacote - em ambos os sentidos. Abaixo segue as médias do tempo de 20 execuções em microsegundos das operações entre o servidor o cliente, com máquinas em uma mesma rede para o servidor em Java, implementando a API RMI e o servidor em C, usando o protocolo TCP:



Como é possível observar, o tempo médio de comunicação para o servidor implementado em Java é, em média, um pouco maior que o dobro do implementado em C. Isso reflete o alto *overhead* que é adicionado ao utilizarmos uma camada a mais na comunicação, por conta da API. Porém, o tempo de desenvolvimento da aplicação é reduzido enormemente, visto que no servidor em C é necessário implementar a formatação do pacote a nível de aplicação.

Nota-se que a sexta operação em C mostrou uma média de tempo maior que o de Java, provavelmente por conta de um maior tráfego na rede em um período de testes diferente, visto que a rede utilizada foi a eduroam, e esta tem uma velocidade altamente volátil, dependendo do número de alunos conectados no instituto.

6. Conclusão

Como é possível observar, a mesma aplicação implementada em diferentes linguagens podem apresentar grandes diferenças de desempenho e tempo necessário para serem implementadas. É necessário levar em consideração a fácil implementação do Java, através da API de RMI, caso o tempo disponível para o desenvolvimento desta seja pequeno e velocidade de conexão não seja uma característica que impacte na interação do usuário com o programa. Caso um maior controle de dados e uma velocidade maior seja necessário, é recomendável adotar a abordagem através da linguagem C, que permite um encapsulamento melhor dos dados, bem como uma conexão mais rápida entre o cliente e o servidor, tendo em vista o maior tempo empenhado no seu desenvolvimento, visto que não há um *middleware*, abstraindo parte da camada de comunicação.

7. Bibliografia

[1] https://www.mongodb.com/
[2] https://mongodb.github.io/mongo-java-driver/
[3] https://docs.oracle.com/javase/7/docs/technotes/guides/rmi/hello/hello-world.html