Sistemas Distribuídos - Atividade 1: Serialização

André Filipe de Medeiros - 542104

December 2, 2024

1 Resumo

A presente atividade visa avaliar o impacto de diferentes protocolos de serialização no tamanho de mensagens trocadas entre processos (cliente e servidor) em Java. Analizaram-se cinco tipos de serialização: padrão do Java (com e sem customização), JSON, XML e Protobuf, em dois cenários: um com baixo volume de dados (requisição de compra) e outro com maior volume (agenda de contatos). Com auxílio do Wireshark, comparou-se o número de bytes enviados em cada contexto. Por fim, a serialização pelo protocolo Protobuf aparenta ser uma boa opção, pois apresentou eficiência em termos de compactação e pode ser utilizado em diferentes sistemas.

2 Discussão Técnica

2.1 Serialização

A serialização é um procedimento responsável por transformar um objeto ou estrutura de dados em um formato possível de ser armazenado ou transmitido. Nesse sentido, a serialização é fundamental para que sistemas diferentes se comuniquem no contexto de computação distribuída.

Atualmente, existem diferentes protocolos que implementam a serialização como a serialização padrão do Java, serialização customizada do Java, JSON, XML e Protobuf.

3 Metodologia

3.1 Código

Os códigos foram escritos na linguagem Java. Programou-se dois programas para cada protocolo e cenário. O primeiro programa é referente ao processo servidor, que foi executado no endereço *localhost* e porta 6789. O papel do processo servidor é de apenas receber a mensagem do processo cliente e imprimir a mensagem no terminal. Enquanto, no processo cliente, instanciam-se objetos e os enviam ao processo servidor.

Ao todo, criaram-se códigos a dez cenários:

$$N = 5\ (prot.\,de\,serializacao) \times 2\ (compra\,e\,agenda) = 10 \tag{1}$$

Os códigos estão disponíveis em: https://github.com/andredemedeiros/serializations.git

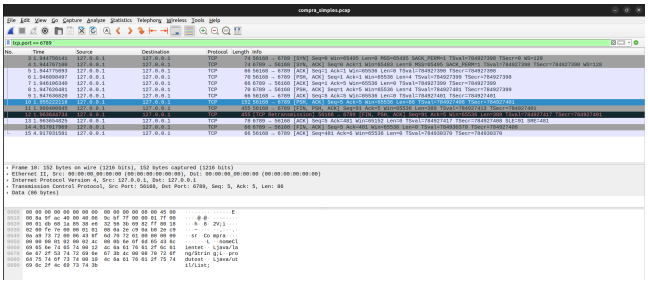
3.2 Captura de pacotes

A captura de pacotes ocorreu em três etapas. A primeira etapa foi referente a inicialização de captura de pacotes através do Wireshark. Na segunda etapa,

executaram-se os processos servidor e cliente, respectivamente, para que os objetos fossem serializados e transmitidos. Por fim, os dados da captura foram salvos em formato *.pcap* e estão disponíveis no repositório.

Figure 1: Captura de pacotes da serialização padrão no cenário de requisição de compra.

(a) Execução dos processos servidor e cliente com protocolo de serialização padrão do Java.



(b) Captura dos pacotes trocados pelos processos cliente e servidor com protocolo de serialização padrão do Java.

Aplicou-se o mesmo procedimento de captura de pacotes em todos cenários.

3.3 Cálculo de tamanho de pacotes

Os pacotes do cenário que simula uma requisição de compra, possui um volume consideravelmente menor em comparação com os pacotes do cenário que simula uma agenda de contatos. Por isso, no primeiro cenário, foi possível calcular manualmente (com arquivos os *.pcap*) a quantidade de *bytes* necessária para transmitir a mensagem após serialização. No seguindo caso, porém, utilizou-se o sotware tshark que automatiza a leitura e operação em arquivos *.pcap*.

4 Discussão dos Resultados

Após a escrita dos scripts em Java que implementam a serialização e de posse dos arquivos *.pcap* com os tamanhos dos pacotes transmitidos (em *bytes*) foi possível gerar gráficos de barras comparativos entre os protocolos em diferentes cenários.

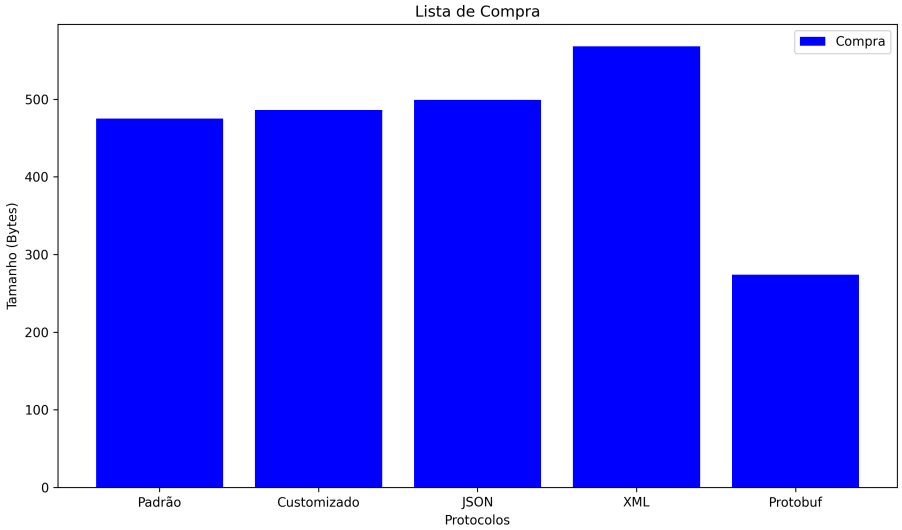


Figure 2: Tamanho das mensagens enviadas após serialização, cenário de requisição de compra.

Observa-se pela Figura 2 que o protocolo de serialização Protobuf apresenta uma eficiência considerável em termos de compactação, quando comparado aos outros protocolos.

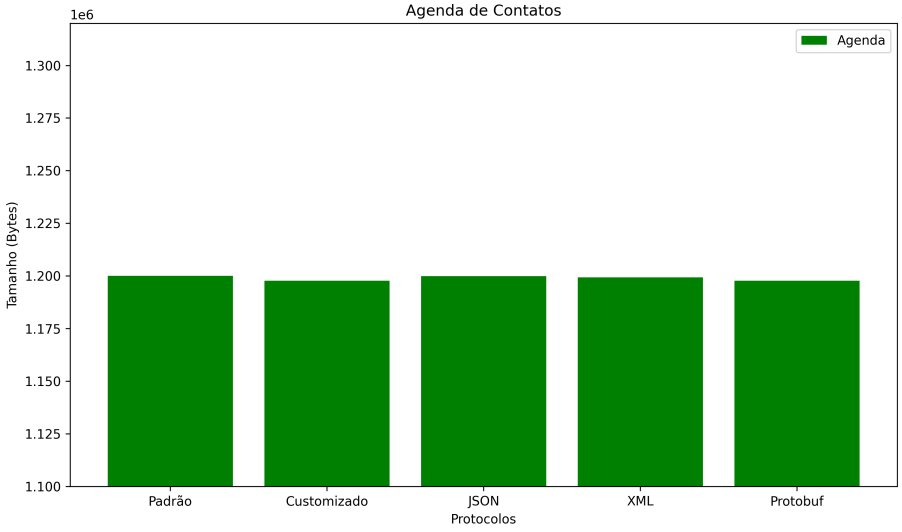


Figure 3: Tamanho das mensagens enviadas após serialização, cenário de agenda de contatos.

No segunda caso, Figura 3, porém, percebe-se que para mensagens que possuem imagens a discrepância percentual entre a eficiência dos protocolo de serialização é menor. O melhor protocolo de serialização nesse caso foi o customizado. Mesmo assim, o protocolo Protobuf também apresentou uma boa eficiência em comparação aos outros protocolos.

Categoria	**Compra**		**Agenda**	
	Valor (B)	**+/- Média (%)**	**Valor (B)**	**+/- Média (%)**
Padrão	475	-3.5%	1,199,998	0.07%
Custom	486	5.6%	1,197,660	-0.12%
JSON	499	8.4%	1,199,956	0.07%
XML	568	23.3%	1,199,362	-0.06%
Protobuf	274	-40.5%	1,197,724	-0.11%

Table 1: Tabela de cenários de lista de compra e de agenda de contatos com valores e comparativo cercentual

A Tabela 1 apresenta um comparativo resumido entre os protocolos de serialização em diferentes cenários. Além disso, pode-se comparar, em cada cenário, o percentual de eficiência de um protocolo utilizado em relação à média.

Conclui-se que os protocolos de serialização resultam em diferentes eficiências de compactação. Essas diferenças estão relacionadas ao protocolo e à maneria que a serialização foi implementada. Além disso, o melhor protocolo de serialização varia de sistema para sistema. No cenário de lista de compras, o Protobuf foi considerávelmente eficiente. Porém, no cenário de agenda de contatos, a serialização customizada foi mais eficiente. No entanto, em ambos casos, o Protobuf foi o único protocolo que apresentou eficiência maior que a média. Além disso, esse protocolo pode ser implementado em sistemas além do Java.