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. Evidenciou-se que a serialização pelo protocolo Protobuf foi a mais eficiente em termos de compactação.

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:

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

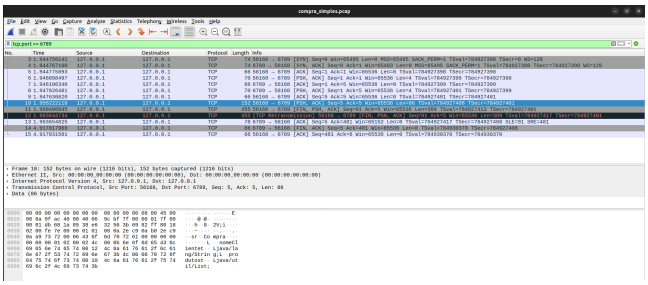
Os códigos estão disponíveis em: https://github.com/andredemedeiros/serializations.git

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.

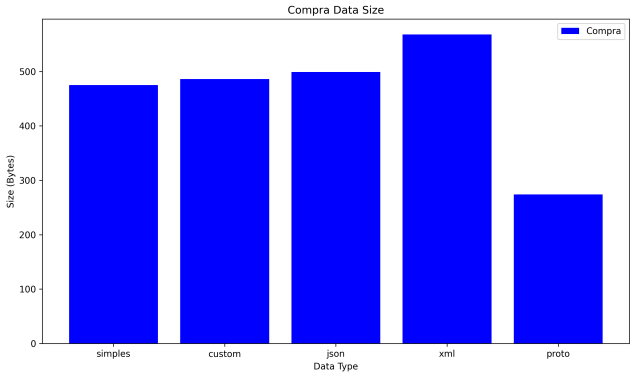
3.3 Cálculo de tamanho de pacotes

Os pacotes do cenário que simula uma requisição de compra, possui um volume considerávelmente 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*.

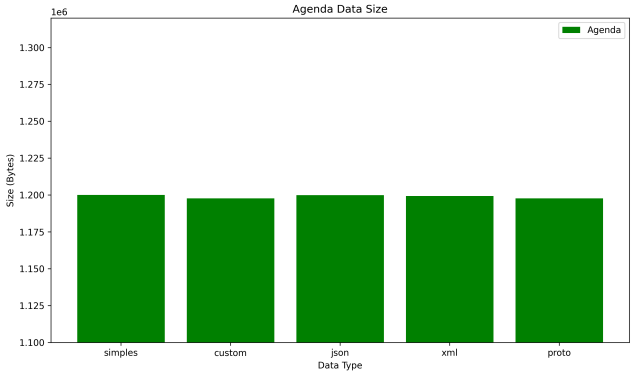
4 Discussão dos Resultados

Após a escrita do *script* em C++ que amplia e reduz (em dpi) uma imagem através de interpolação bilinear, submetemos a Figura 2a a uma redução de 1250 dpi a 100 dpi e chegamos à Figura 2b. Depois, através de interpolação bilinear, ampliamos de volta a Figura 2b de 100 dpi a 1250 dpi e a Figura **??** é o resultado.

Figure 2: 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.

Aplicamos o mesmo raciocínio a outras duas imagens escolhidas: imagem do coqueiro e imagem da chuva.

Conclui-se que, em determinadas condições, a interpolação bilinear é um ótimo método para interpolar pixels a uma imagem. Em especial, quando a diferença de intensidade entre pixels vizinhos é considerável, a interpolação é uma boa estratégia para aumentar a densidade de pixels (dpi). Além disso, por ser uma média ponderada, a interpolação bilinear acaba suavizando a diferença de intensidade entre pixels de uma imagem.

5 Código Fonte em C++