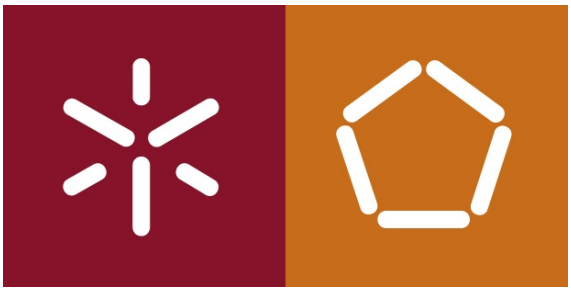


Universidade do Minho

Mestrado Integrado em Engenharia Informática

Interoperabilidade Semântica

Comunicação via HL7

Folha de Exercício FE02

Bárbara Cardoso (A80453)
João Vilaça (A82339)

5 de Março de 2021

Resumo

Vivemos inegavelmente numa sociedade profundamente digitalizada. A grande maioria dos bens e serviços que adquirimos e utilizamos estão ou estiveram de alguma maneira associados a um suporte informático, como, por exemplo, nos supermercados onde a gestão de stock de produtos é feita utilizando software para esse fim ou quando chegamos à caixa e é utilizado um software de faturação para registar a nossa compra. Como é óbvio, estes sistemas são profundamente dependentes entre si, ou seja, quando efetuamos uma compra de determinados produtos, é expectável que o stock desses produtos seja alterado para refletir a nossa compra, e é nesta situação que entre o conceito de Interoperabilidade Semântica, a capacidade destes sistemas partilharem dados e ter essas mesmas informações correctamente interpretadas pelo sistema receptor como no sistema transmissor.

Um caso particular, de grande complexidade, destes sistemas que operam em conjunto é, sem dúvida, na área da saúde. Todos os dias existem milhões de informações completamente distintas a circular entre estes sistemas, marcação de consultas, entrada no centro de saúde, início da consulta, informações e medições da consulta, exames e receitas derivadas dessa consulta, fim da consulta, pagamento da consulta e depois, realização dos exames marcados, geração dos relatórios sobre os exames, levantamento dos medicamentos, etc. E isto apenas no âmbito de uma consulta num centro de saúde, existe verdadeiramente um espectro gigante de tipos de dados que circulam diariamente nestes sistemas.

Assim, foi então essencial desenvolver um protocolo que garantisse a eficiente e eficaz comunicação de todos estes dados entre sistemas, o HL7, para que existisse um norma de base sobre a qual poderiam ser desenvolvidos estes sistemas de software tendo como certo e garantido o formato dos dados estes terão de gerar e enviar e ser capazes de receber.

Conteúdo

1 Introdução **3**

2 Ferramentas e Técnicas **4**
2.1 ZeroMQ (0MQ) 4
2.2 Programação por Eventos 4
2.3 Reenvio de mensagens 4

3 Análise de Performance **5**
3.1 Condições de rede 5
3.2 Criação de mensagens 6

4 Conclusão **7**

5 Bibliografia **8**

1 Introdução

Neste exercício, depois de implementado um sistema composto por 2 programas que operem em conjunto recorrendo a comunicação usando o formato HL7, pretende-se analisar a eficiência do programa desenvolvido verificando a capacidade do sistema sobre grandes cargas e a sua velocidade de resposta. Esta análise deverá ser feita para todos os devidos tipos de mensagens possíveis, apenas consideradas entregues à receção de um ACK, e tendo em conta a variabilidade das condições da rede, nomeadamente, perdas de pacotes e tempo de comunicação.

Com a análise e a evolução dos testes sobre o sistema serão feitas e documentadas todas as melhorias consideradas relevantes e discutidos, se ainda se mantiverem, os bottlenecks do sistema e possíveis, caso existam, soluções para os mesmos.

2 Ferramentas e Técnicas

2.1 ZeroMQ (0MQ)

Para melhorar o desempenho de cada aplicação isoladamente e do sistema como um todo, ao invés de se optar pela utilização simples de Sockets, que obrigariam a desenvolver toda a camada de comunicação, nem de WebSockets, que seriam extremamente ineficientes, escolhemos utilizar ZeroMQ. 0MQ é uma biblioteca de troca de mensagens assíncronas, desenvolvido para ser usado em ambientes distribuídos e concorrentes tal como é o nosso caso. Esta ferramenta oferece ainda funcionalidades de fila de espera de mensagens sem a necessidade do uso de um broker dedicado.

Sendo uma ferramenta pequena e leve oferece grande performance no envio e receção de grandes volumes de dados.

2.2 Programação por Eventos

Com o uso de 0MQ, é fácil encapsular toda a lógica de comunicação entre os serviços em Threads, permitindo paralelizar operações com eficácia de modo a poder processar as várias mensagens recebidas em simultâneo.

Deste modo, é possível aproveitar todo o poder computacional da máquina onde está a ser executada a nossa aplicação, diminuindo o tempo de processamento de cada pedido, servido cada utilizador mais rápido e consequentemente mais utilizadores.

2.3 Reenvio de mensagens

Com esta paralelizarão, e apesar de todas as vantagens que dela advêm, surge um problema, o tratamento do reenvio de mensagens perdidas.

Para isto, foi necessário arranjar uma solução, em que teríamos uma estrutura de dados para armazenar todas as mensagens enviados cujo ACK ainda não foi recebido. Isto significa que apesar de a adição de mensagens a esta estrutura ser uma operação atómica, a sua remoção, uma vez que corresponde à receção de uma mensagem, a de ACK, é efetuada num contexto de uma Thread e, por isso, uma operação concorrente.

Para ultrapassar este desafio, utiliza-se um ConcurrentHashMap de Java, que recorrendo à utilização de locks, garante que esta operação crítica é apenas efetuada por uma Thread de cada vez.

A utilização de um Map ou invés de uma List ou de um Set é também uma decisão justificada, porque tem um impacto muito significativo no performance do sistema, uma vez que todas as mensagens enviadas passam por esta estrutura. No caso de uma List, tanto a inserção como a remoção de elementos teriam uma complexidade de O(n), enquanto que, nos Map, esta complexidade é O(1), uma diferença neste caso de uso abismal.

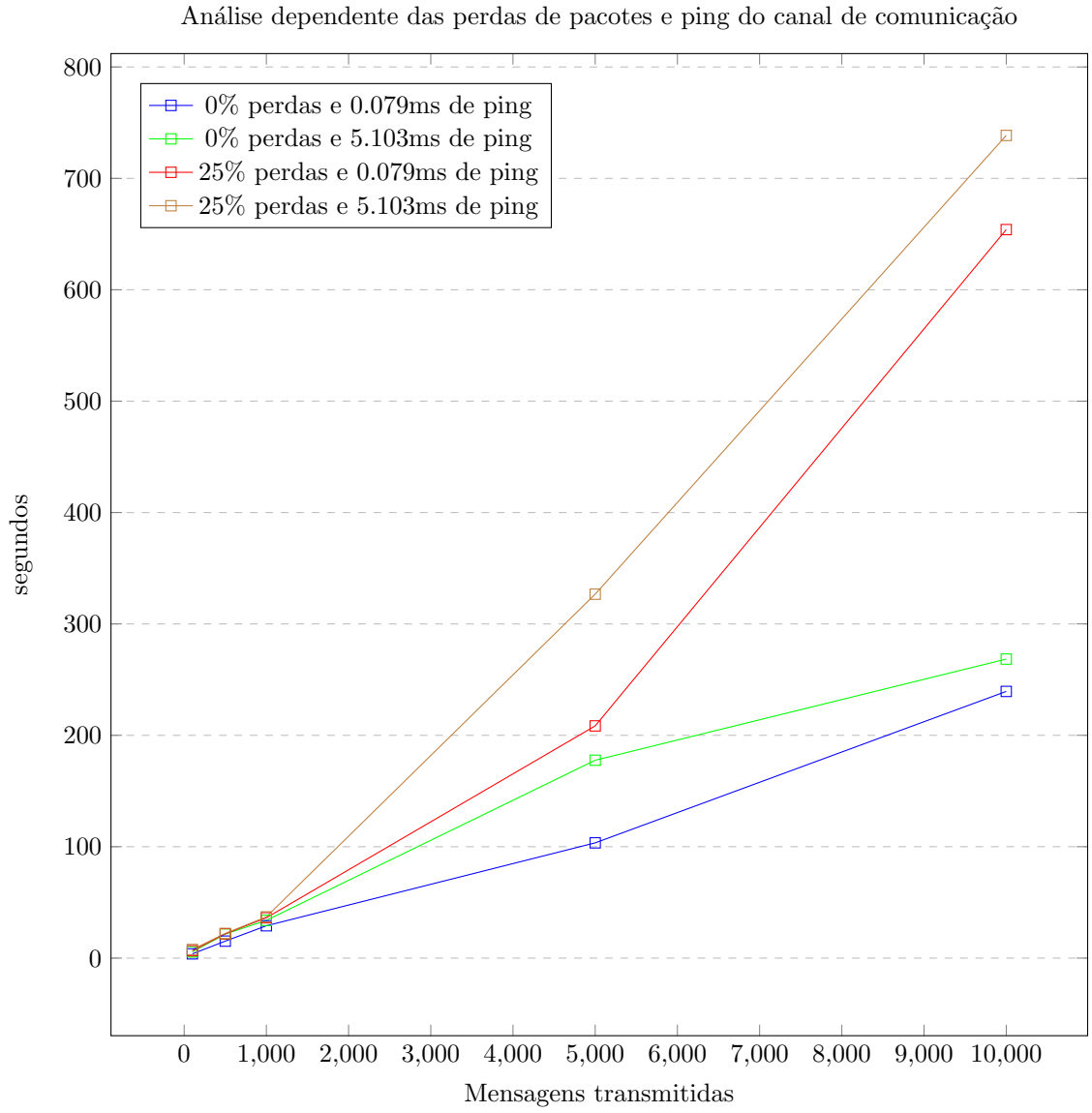
3 Análise de Performance

3.1 Condições de rede

Em seguida analisamos o tempo que o sistema demorar a criar a transmitir N mensagens em várias condições de rede até um total máximo de 10000 mensagens. Estas mensagens são criadas o mais rapidamente possível pelo serviço 'Doctor' e imediatamente enviadas para o serviço 'Clinic'.

O primeiro factor é a qualidade da ligação, em que são avaliados respectivamente uma rede Ethernet com perdas na ordem de 0% e uma rede Wi-Fi de má qualidade e muito pouco fiável com perdas de pacotes na ordem dos 25%.

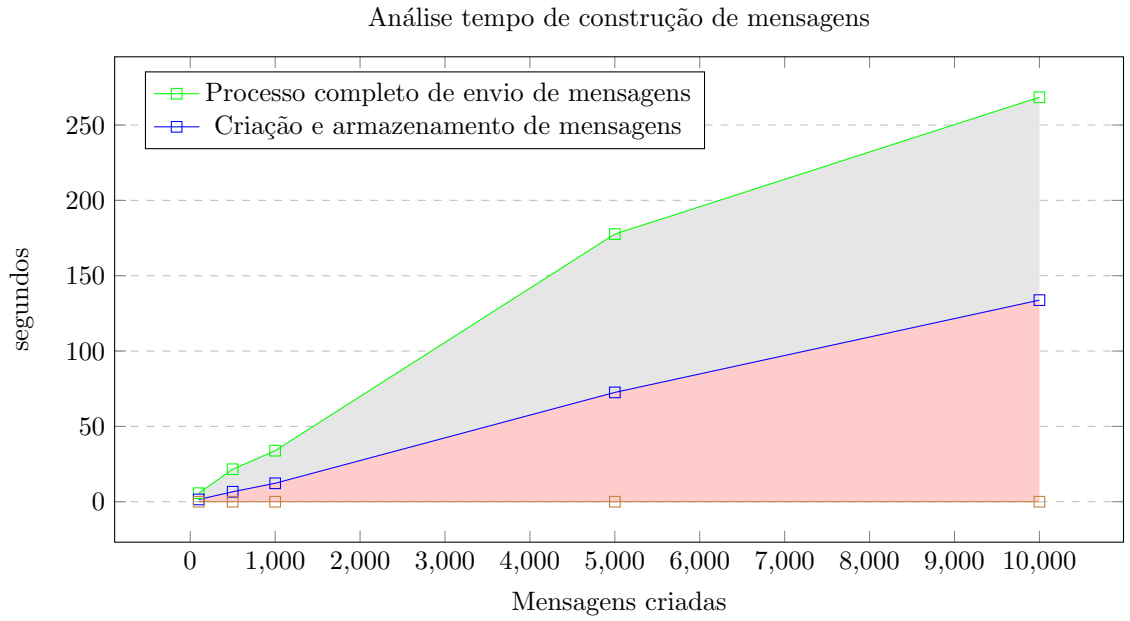
Outro factor, é a distância e consequente tempo de comunicação entre as máquinas onde se encontram a ser executados os serviços. Em primeiro lugar, uma rede local, onde o ping é, em média, 0.079ms, e, em segundo, dois computadores em redes diferentes mas geograficamente próximos (cerca de 10km), com ping médio de 5.103ms.

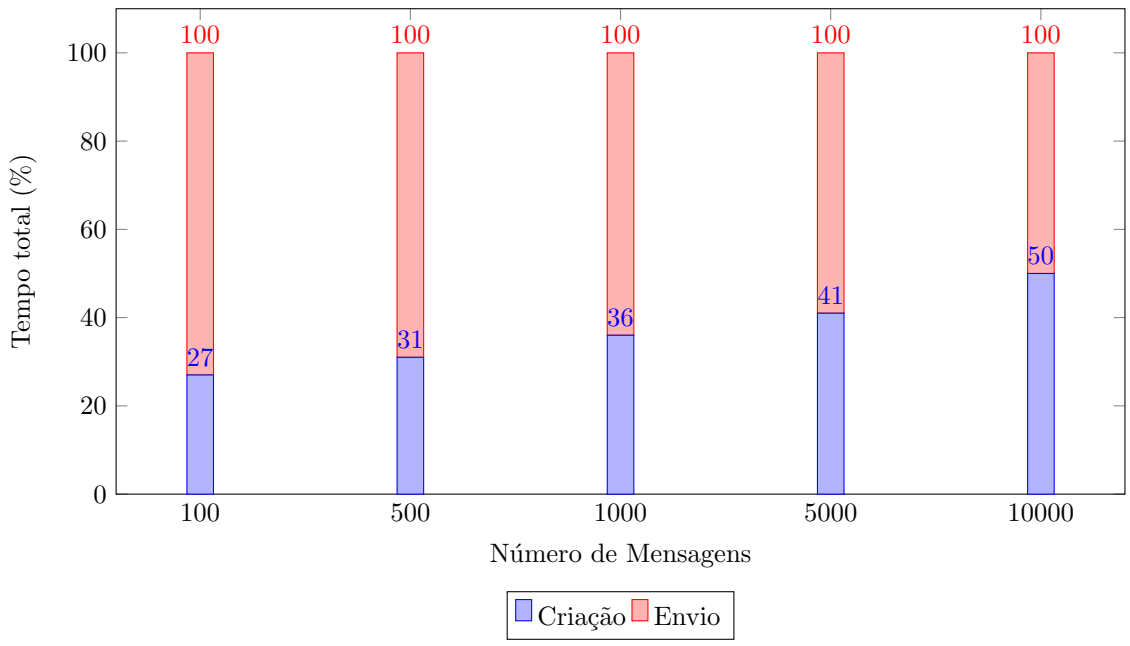


3.2 Criação de mensagens

Em condições de rede em que temos 0% perda de pacotes e 5ms de tempo de comunicação entre serviços, como descrito no gráfico antertior, verificamos que o tempo de comunicação de mensagens é bastante elevado. Mas aí é medida a duração de todo o processo desde a criação, armazenamento, envio e possíveis reenvios de mensagens casos os ACK não sejam recebidos.

Assim, é importante analisar qual desta percentagem de tempo é gasto apenas na construção e armazenamento das 10000 mensagens.





4 Conclusão

Feita a avaliação do sistema, foi possível corrigir alguns bottlenecks que limitavam a eficiência do mesmo e, assim, foi atingida uma melhoria significativa na performance quer na geração das mensagens HL7 quer na sua transmissão.

Apesar de existiram ainda imensas limitações estas são difíceis de resolver e muitas delas são relacionadas directamente com o protocolo utilizado. Ainda assim o sistema tem a capacidade de aguentar e processar mensagens sobre grande carga de utilização e mesmo sobre más condições de rede, de modo que para o utilizador final tudo aconteça sem precalços.

Esta alta disponibilidade foi apenas conseguida devido ao variado leque de técnicas e ferramentas que foram implementadas ao longo do desenvolvimento deste projeto.

5 Bibliografia

Referências

[1] The Free, Open and Best HL7 Parser and Library for Java
`https://hapifhir.github.io/hapi-hl7v2/`

[2] Interop: The Promise and Perils of Highly Interconnected Systems
John Palfrey, Urs Gasser. Basic Books; 1 edition (June 5, 2012)

[3] Spring Shell,
`https://projects.spring.io/spring-shell/`