Daniel Ribeiro Favoreto
Rafael Igor Athaydes Rios

Ataque de Dicionário em Mensagem Criptografada com Middleware Orientado a Mensagens

Vitória
24 de Julho de 2017

Introdução

Este trabalho apresenta o uso de Middleware orientado a mensagens JMS e análise de desempenho em um cluster de computadores para realizar um ataque de dicionário de uma mensagem criptografada de forma paralela.

Para analisar o desempenho da arquitetura mestre-escravo desenvolvida para este tipo de aplicação, foram realizados testes em 2 cenários diferentes: 1 cenário A: com 3 máquinas, cada qual rodando 4 escravos (total 12 escravos) - 1 cenário B: com 3 máquinas, com 2 delas rodando 4 escravos e 1 delas rodando 8 escravos (total 16 escravos). Também foi realizado uma comparação com a implementação do primeiro trabalho.

Este relatório é dividido em 3 partes. Na primeira parte são discutidos detalhes da implementação realizada. Na segunda parte são mostrados os testes realizados e a análise de desempenho da arquitetura mestre-escravo em um cluster de computadores e comparação com o primeiro trabalho. A terceira parte apresenta as conclusões obtidas da análise de desempenho realizada.

Implementação

A arquitetura utilizada para o desenvolvimento da aplicação foi uma cliente-servidor com mestre-escravo. O mestre registra-se no serviço de nomes (RMI Registry), oferecendo a mesma interface do trabalho 1. Clientes fazem requisições para o mestre, provendo o texto criptografado e alguma palavra presente no texto conhecida pelo cliente.

O mestre delega o ataque a vários escravos colocando mensagens em uma fila JMS chamada “subAttackQueue”, os escravos então consomem mensagens dessa fila e realizam o ataque de dicionário no texto criptografado, e para cada chave candidata encontrada, devem retornar ao mestre, por meio da fila “guessQueue” a chave candidata e a mensagem descriptografada. A mensagem colocada na fila “subAttackQueue” contém o subAttack especificando a mensagem cifrada, o texto conhecido, o número do ataque, índice inicial e índice final:

```
sA = new SubAttack(ciphertext, knowntext, attackNum, initialI, finalI);
message = contextSubAttack.createObjectMessage();
message.setObject(sA);
producerSubAttack.send(subAttackQueue, sA);
```

Já a mensagem colocada na fila “guessQueue” contém uma “Guess” com a chave candidata, a mensagem e nome do escravo:

```
guessWord = new Guess();
guessWord.setKey(dicWord);
guessWord.setMessage(encrypted);
guessWord.setSlaveName(slaveName);
ObjectMessage message = contextGuess.createObjectMessage();
message.setObject(guessWord);
producerGuess.send(guessQueue,message );
```

Classes “SubAttack” e “Guess”:

```
public class SubAttack implements Serializable{

    private byte[] ciphertext, knowntext;
    private int attackNumber;
    private long initialindex,finalindex;
```

```
public class Guess implements Serializable {
        private String slaveName;
```

De forma que para sinalizar que um escravo terminou um sub-ataque, uma mensagem contendo um “Guess” vazio é criada:

```
guessWord = new Guess(); /
guessWord.setKey("");
guessWord.setSlaveName(slaveName);
```

A quantidade de palavras a ser testada por cada escravo foi definida da seguinte forma:

```
int wordsEach = (sizeDicti – (sizeDicti % m) )/m;
```

De forma que todas as palavras do dicionário serão procuradas por todos os escravos.

Para inicializar o mestre e os escravos, foram utilizados os seguintes comandos:

```
MasterImp <nomeDoMestre> <ipJMS>
SlaveImp <nomeEscravo> <ipJMS>
```

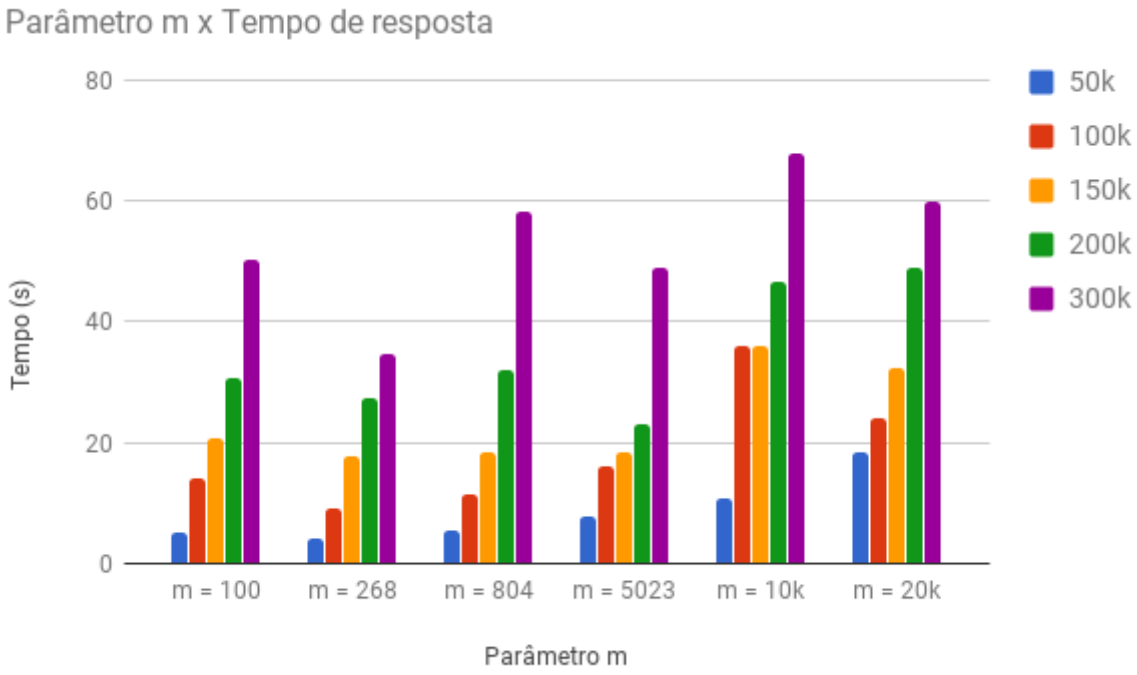
Testes e análise de desempenho

Os tamanhos das mensagens criptografadas foram 50kBytes, 100kBytes, 150kBytes, 200kBytes e 300kBytes. Para realizar testes uniformes para todos os casos considerados, mensagens para cada tamanho foram criptografadas e salvas previamente, juntamente com a palavra conhecida de cada mensagem. Para uma melhor precisão de resultados, cada teste para um tamanho de mensagem foi repetido três vezes e uma média aritmética dos tempos resultantes foi calculada e registrada. Foram utilizados os seguintes cenários:

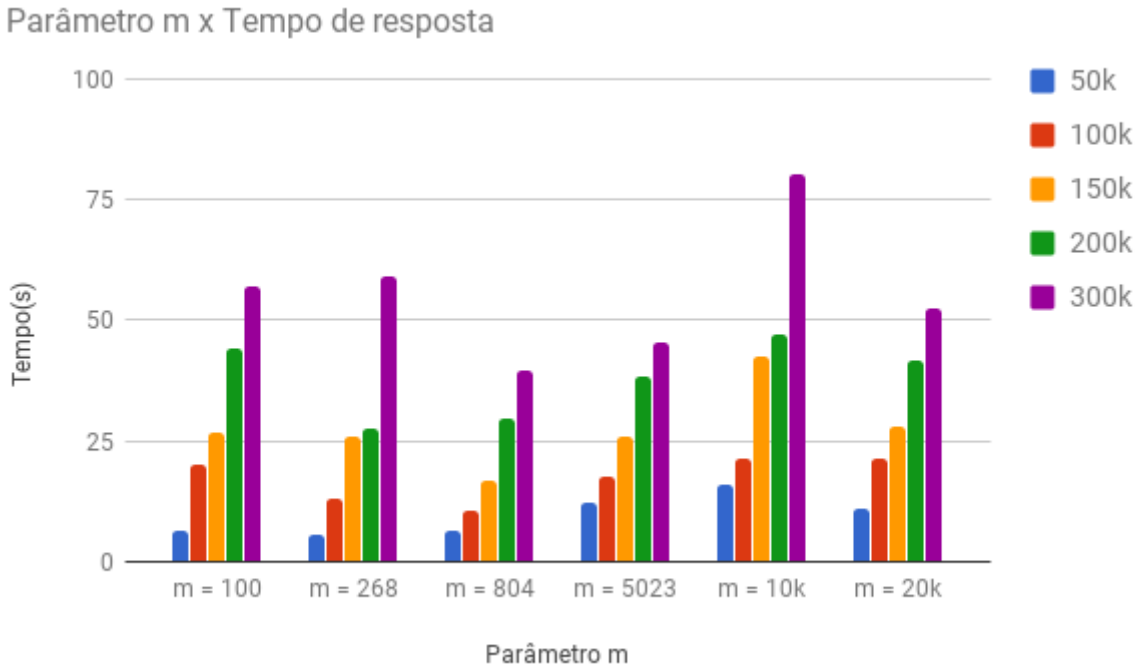
- Um cenário A: com 3 máquinas, cada qual rodando 4 escravos (total 12 escravos).
- Um cenário B: com 3 máquinas, com 2 delas rodando 4 escravos e 1 delas rodando 8 escravos (total 16 escravos).

Comparação parâmetro m x tempo de resposta

(Cenário A)



(Cenário B)



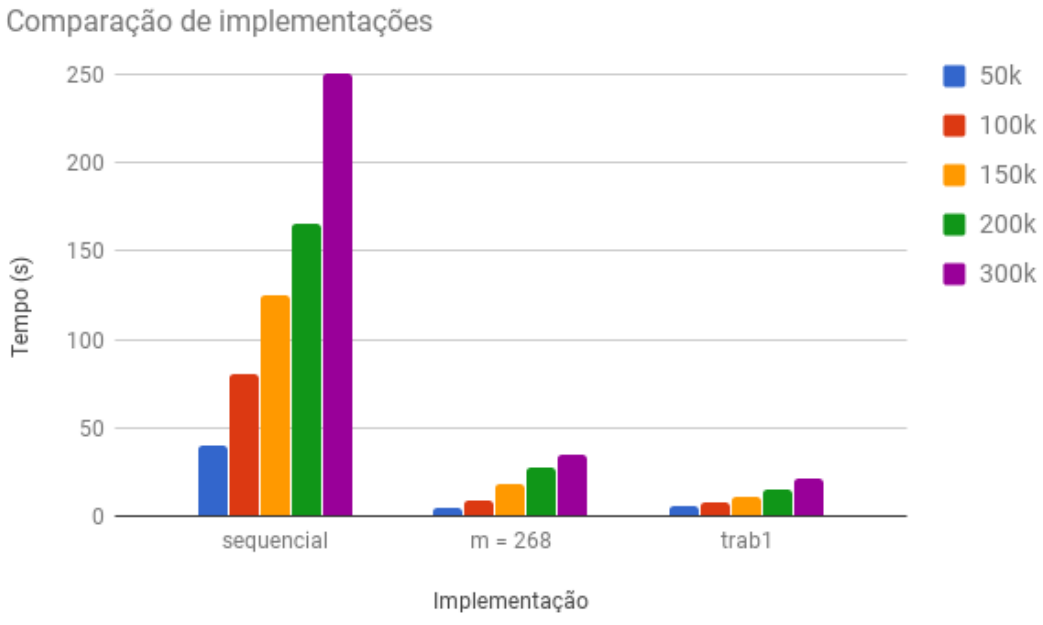
Testes foram realizados utilizando diferentes tamanhos de arquivos e diferentes valores do parâmetro **m,** nos permitindo observar que no Cenário A, quanto maior o tamanho de **m,** maior seria o tempo gasto para realizar o ataque, enquanto que no Cenário B, a partir de 10Kbytes o tempo gasto começou a reduzir.

Conforme o crescimento do parâmetro **m,** uma maior partição dos índices é feita, fazendo com que haja um maior tráfego na rede e gerando um desempenho inferior. Ademais, por algum motivo desconhecido, o JMS retirava alguns escravos fazendo com que em alguns testes estivessem rodando apenas 3 escravos em uma mesma máquina ao invés de 4 escravos, assim como no cenário B, na máquina com 8 escravos estivessem rodando 6. Acredita-se que isso tenha influenciado o desempenho de cada teste.

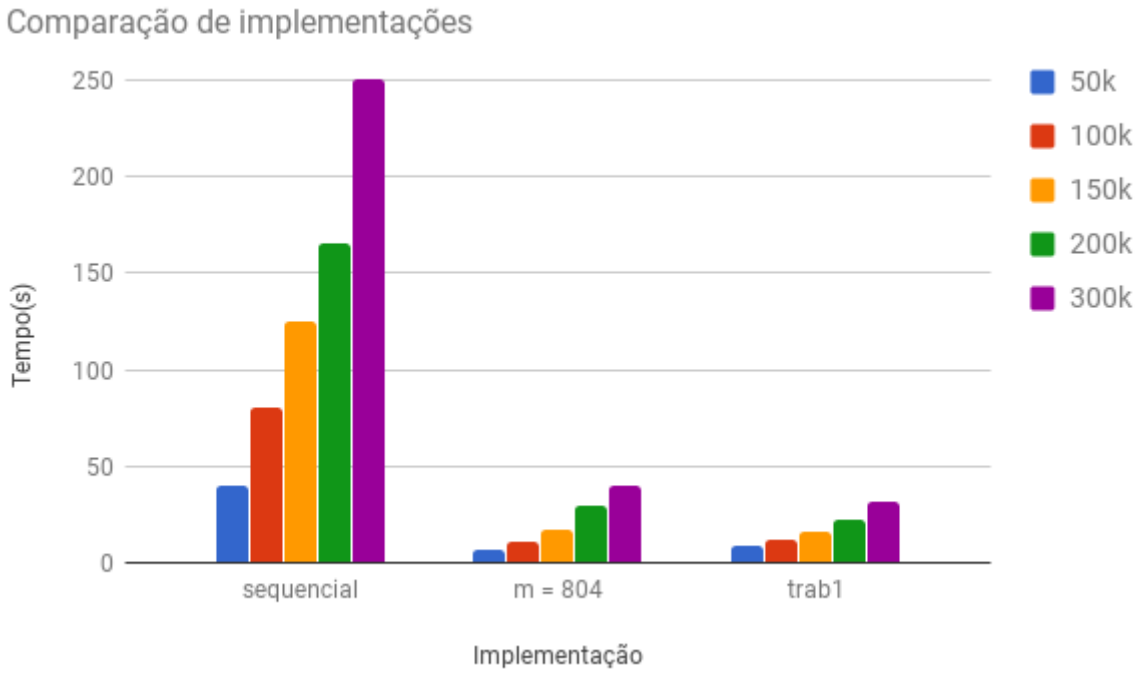
Entretanto, diferentes comportamentos foram observados principalmente com relação à escolha do **m** com melhor tempo. No cenário A, o melhor valor de **m** obtido foi de 268, enquanto que no cenário B, o melhor valor de **m** foi de 804.

Alguns ruídos nos testes foram detectados devido à instabilidade na rede do Labgrad, fazendo com que houvesse uma anormalidade no desempenho para alguns arquivos. Por exemplo, no cenário A, para m = 10k e arquivos de 100kBytes e 150kBytes, houve um aumento no tempo de resposta, assim como, no cenário B para tamanhos de 100kBytes e 300kBytes.

Comparação serial, m (ótimo) e trabalho 1 (Cenário A)



(Cenário B)



De acordo com o **m** escolhido em cada cenário, podemos observar que houve uma maior rapidez no cenário A tanto em relação ao trabalho 1 quanto em relação aos respectivos parâmetros **m** escolhidos.

Também é visível que o trabalho1 obteve o melhor desempenho em ambos os cenários. Em relação aos cenários, o cenário A se mostrou mais simples (devido a menos escravos) e com desempenho melhor em relação ao cenário B.

Testes de garantia de interoperabilidade foram realizados com a dupla Josias Oliveira e Vinícius Arruda.

Testes foram realizados em máquinas com processador Intel Core i5, 8GB memória RAM e sistema operacional Linux.

Conclusão

A partir dos testes realizados, verifica-se que para a aplicação implementada, a solução apresentada pelo trabalho 1 é mais rápida, além de permitir a recuperação de falhas, caso um escravo falhe por qualquer motivo. Em contrapartida, verifica-se que a implementação de um ataque de dicionário com Middleware orientado a mensagens é mais simples, devido o gerenciamento de escravos por parte do mestre ser resumido em colocar mensagens em uma fila, deixando de acordo com a disponibilidade de cada escravo em realizar o sub-ataque. Entretanto, a falta de checkpoint acaba por dificultar o tratamento de recuperação de falhas, não sendo possível identificar o escravo que falhou, o que é uma desvantagem em relação ao trabalho 1 em que era possível a recuperação de falhas. A principal vantagem observada na implementação do trabalho 1 em relação a este trabalho, é o desempenho geral, enquanto que a simplicidade de implementação deste trabalho talvez seja a principal vantagem em detrimento do desempenho.

A escolha do valor de **m** deu a entender que muito provavelmente uma escolha acima de 1k, piora o desempenho como observado nos testes. É bem possível que devido à interferência da rede, outros valores de **m** reduziriam o tempo observado, porém, por conta de fatores como falta de disponibilidade de horário do laboratório, a dupla entende que os testes realizados foram satisfatórios.