Daniel Ribeiro Favoreto
Rafael Igor Athaydes Rios

Arquitetura mestre-escravo com Java RMI: Ataque de dicionário em mensagem criptografada

Vitória
20 de Junho de 2017

Introdução

Este trabalho apresenta uma arquitetura mestre-escravo utilizando a API Java RMI (Remote Method Invocation) 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 com uma solução centralizada (toda a computação realizada por um mesmo processo, sem uso de arquitetura cliente-servidor e com uma solução paralela e distribuída.

Este relatório é dividido em três 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 frente a implementação serial. 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 operações como registro e desregistro para escravos, além da operação do ataque de dicionário (attack). Escravos registram-se no mestre, utilizando o serviço de nomes para encontrá-lo e oferecem uma operação de ataque (startSubAttack). Clientes fazem requisições para o mestre, provendo o texto criptografado e alguma palavra presente no texto conhecida pelo cliente. O mestre faz requisições a seus escravos registrados, dividindo o dicionário entre cada escravo. Os escravos realizam o ataque de dicionário no texto criptografado, e para cada chave candidata encontrada, devem retornar ao mestre, por meio de uma operação oferecida pelo mestre (foundGuess), a chave candidata e a mensagem descriptografada.

Caso um escravo gere uma exceção durante sua execução, o mestre o retira de sua lista de escravos e repassa o trabalho que ainda falta ser realizado para algum escravo ocioso. Para que o mestre possa manter controle do trabalho realizado pelos escravos, a cada 10 segundos, cada escravo realizando uma requisição deve enviar ao mestre um checkpoint, ou seja, o índice da última palavra do dicionário já testada como chave candidata. Caso algum escravo falhe em enviar um checkpoint por mais de 20 segundos, este escravo é removido pelo mestre e o trabalho que ainda falta ser realizado é repassado para algum escravo ocioso. Esta estratégia permite que não seja necessário que o trabalho seja realizado do zero toda vez que algum escravo gere uma exceção.

A arquitetura mestre-escravo desenvolvida é composta pelas interfaces Master.java e Slave.java e pelas respectivas implementações das interfaces MasterImpl.java e SlaveImpl.java. A interface Master.java herda das interfaces SlaveManager.java (composta de métodos de gerência dos escravos, como operações de registro, checkpoint e foundGuess) e Attacker.java (que define a assinatura da operação de ataque de dicionário fornecida pelo mestre). A seguir tem-se as interfaces Master.java e Slave.java utilizadas neste trabalho de implementação:

```
public interface Master extends Remote, SlaveManager, Attacker {
        // o mestre é um SlaveManager e um Attacker
}
```

```
public interface Slave extends Remote {
        public void startSubAttack(
                        byte[] ciphertext,
                        byte[] knowntext,
                        long initialwordindex,
                        long finalwordindex,
                        int attackNumber,
                        SlaveManager callbackinterface)
        throws java.rmi.RemoteException;
}
```

Antes de rodar os escravos é necessário executar o mestre primeiramente, sendo que os escravos são executados passando-se um nome descritivo como argumento único. Em seguida, o cliente é executado passando-se como primeiro argumento o nome do arquivo com a mensagem criptografada e como segundo argumento a palavra conhecida dessa mensagem. Caso o cliente não encontre o arquivo com a mensagem criptografada, é gerado uma mensagem aleatória criptografada passando-se como terceiro argumento o tamanho dessa mensagem, sendo que quando não é especificado esse parâmetro, o cliente gera essa mensagem entre 1kByte e 100kBytes.

Foi implementada uma solução serial/sequencial para a comparação com a solução distribuída, incluída na classe ClientSequencial. Para utilizá-la, basta executar passando-se os mesmos argumentos do cliente da solução paralela.

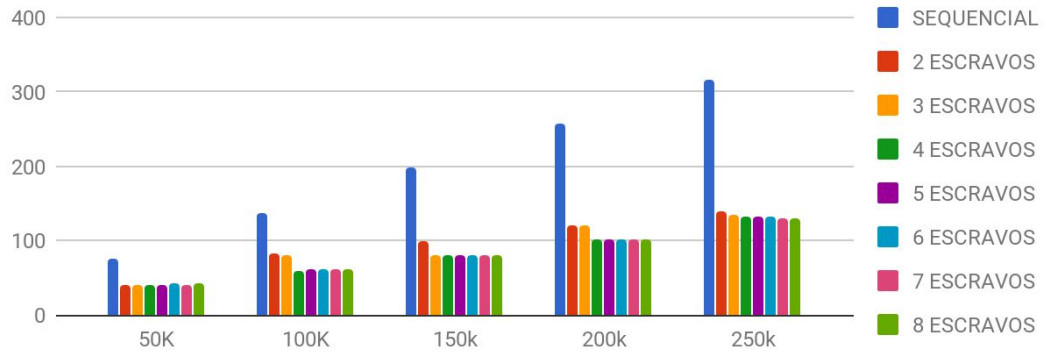
Para os cálculos dos tempos, foi usada a função System.nanoTime, disponível na API do Java.

```
long tempoInicial = System.nanoTime();

Guess[] chutes =mestre.attack(mensagemCriptografada,mensagemConhecida);

long tempoFinal = System.nanoTime();
```

Testes e análise de desempenho

Os tamanhos das mensagens criptografadas foram 50kBytes, 100kBytes, 150kBytes e 200kBytes. 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.

Comparação serial/sequencial versus paralela distribuída



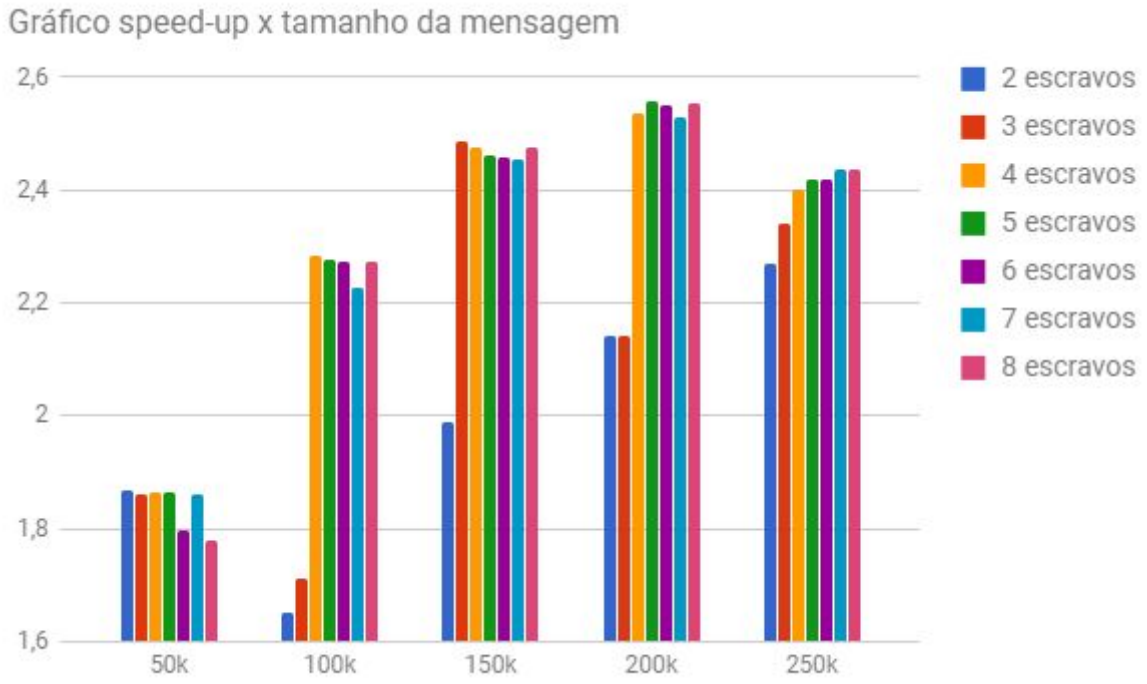
Analisando-se o gráfico tempo de resposta x tamanho da mensagem, nota-se que a abordagem sequencial tem desempenho inferior às abordagens distribuídas no quesito tempo de resposta e que todas as abordagens têm característica linear. O desempenho inferior da abordagem sequencial se deve à necessidade da mesma ter de percorrer todo o dicionário em busca de palavras-chaves candidatas, que é uma operação custosa considerando o tamanho do dicionário.

Já as abordagens distribuídas dividem o acesso ao dicionário com base nos índices que são repassados pelo mestre, diminuindo a quantidade de palavras-chaves que precisam testar, e como as "Guesses" que fornecem são independentes, não há necessidade de aguardar o fim do processamento de outros escravos.

De forma geral, a divisão em maior número de escravos resultou em menor tempo. Apesar disso, pode-se notar que a diferença, em segundos, entre uma abordagem com n-1 escravos e uma abordagem com n escravos para um mesmo tamanho de vetor tende a diminuir conforme n aumenta. Essa característica se deve

ao fato de que o acesso ao dicionário tem tempo reduzido com o número maior de escravos, mas conforme o número de índices cai, não existe muita diferença entre os tempos para n e n-1 escravos percorrerem os seus espaços do dicionário. A característica linear dos gráficos em questão se deve à relação linear do tempo de resposta com o tamanho do arquivo criptografado - quanto maior o arquivo, maior o tempo para decriptografá-lo e para determinar as palavras-chaves candidatas.

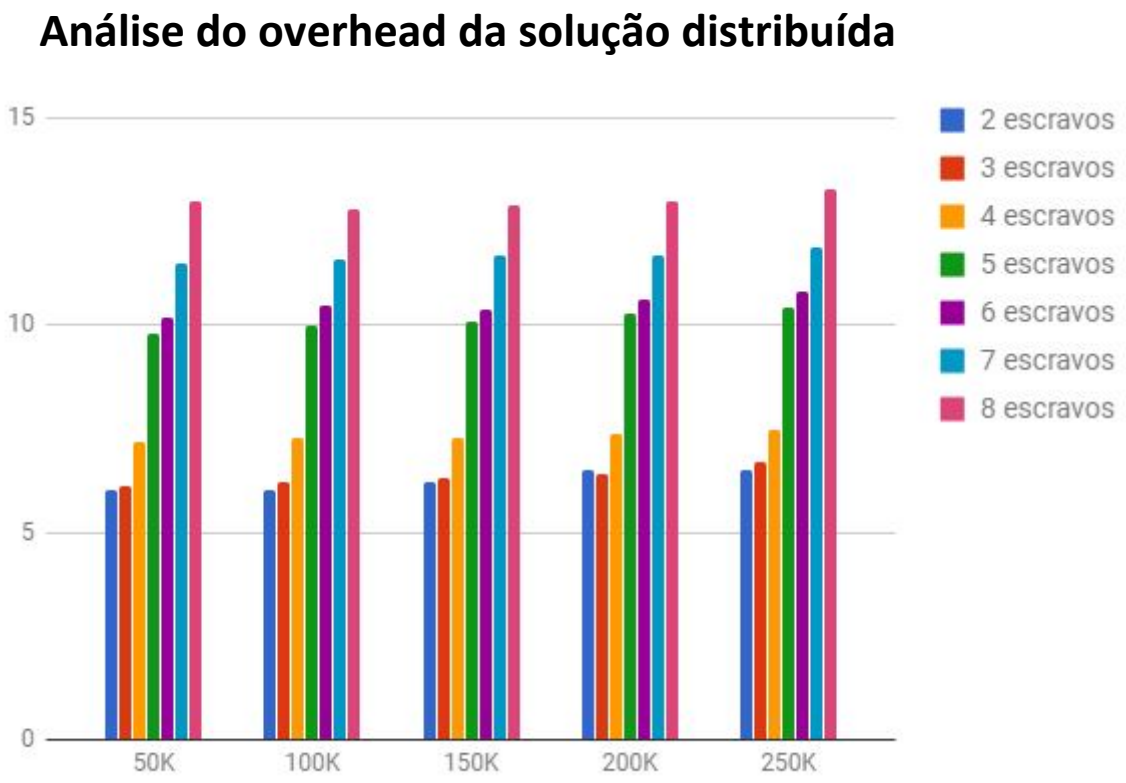
Análise do speed-up da solução distribuída



No que diz respeito ao speed-up, nota-se que há uma variação considerável para diferentes tamanhos. Quando o tamanho da mensagem é de 50kBytes, as soluções com 6 e 8 escravos obtiveram um speed-up menor em relação às outras soluções do mesmo teste, sendo que na média, os valores de speed-up desse teste foram menores, o que já era esperado devido a mensagem ser de apenas 50kBytes. Nos testes com 100kBytes, 200kBytes e 250kBytes nota-se que a solução com 2 e 3 escravos possuem speed-ups menores em relação às demais soluções dos mesmos testes. Nos testes realizados, observou-se um maior speed-up com o tamanho da mensagem igual a 200kBytes alcançando valores superiores a 2,5. No geral, isso permite concluir que o problema de ataque tem um ganho aumentado para os casos distribuídos, o que já tinha sido observado nos tempos do gráfico 1.

O overhead que impede os casos distribuídos de terem speed-up linear se deve a três fatores: a instabilidade da rede, a divisão dos índices e distribuição do

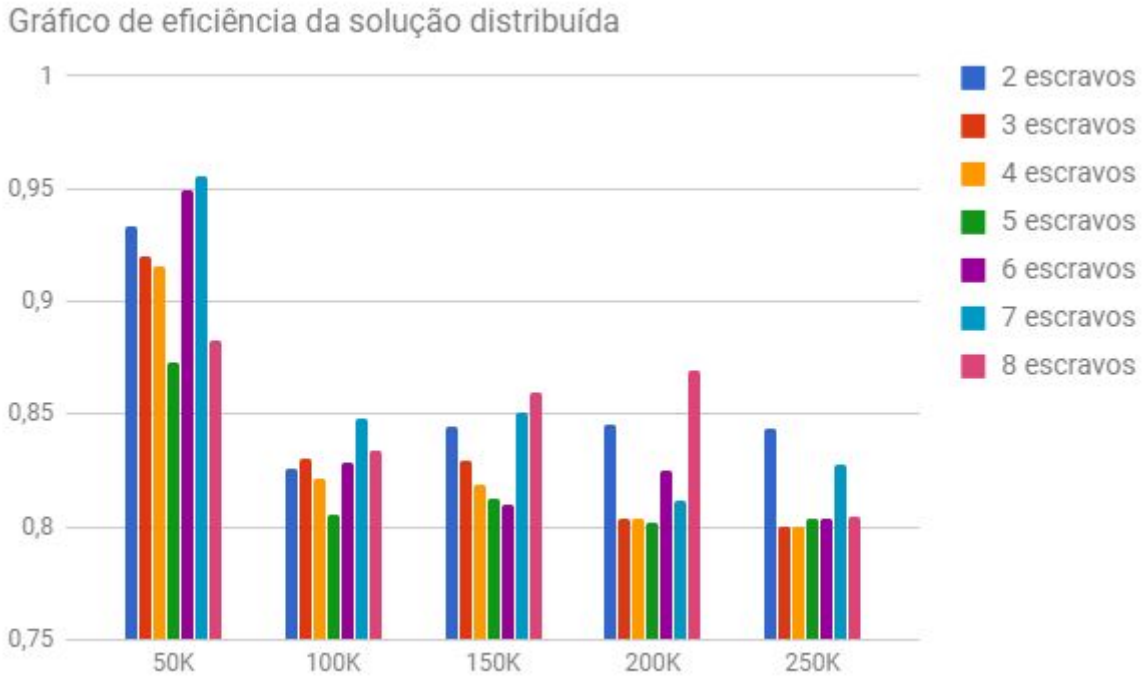
texto criptografado para cada escravo, e a serialização dos "Guesses" de cada escravo. Este último caso, em particular, é o mais provável responsável pelo tamanho do overhead, dado que cada Guess requer, além da passagem da palavra-chave encontrada, a serialização do texto decriptografado. Considerando um número elevado p de palavras-chaves para um determinado texto criptografado, seriam necessárias p serializações, uma para cada arquivo decriptografado. Com tamanhos maiores desses arquivos, a operação de serialização se torna cada vez mais custosa.



Os tempos de overhead apresentados são uma estimativa, que fornecem apenas uma ideia geral do comportamento geral do tempo de resposta. Para realizar a estimativa, considerou-se que toda comunicação pela rede e o envio dos arquivos criptografados e dos índices era realizada, mas não ocorria de fato o envio de "Guesses" pelos escravos. Assim, obteve-se o tempo gasto com as operações não-paralelizáveis do sistema. Como pode ser observado, o overhead de comunicação é relativamente pequeno se comparado ao tempo que é de fato gasto para realizar as operações de decriptografia pelos escravos, independente do número de escravos. Enquanto o overhead é da ordem de milissegundos, o tempo gasto para cada abordagem distribuída é da ordem de segundos.

Para um número maior de escravos, ele tende a aumentar, uma vez que há um número maior de vezes que precisa-se enviar os textos criptografados e os índices do dicionário.

Análise da eficiência da solução distribuída



Como pode ser observado, em geral, os casos distribuídos tendem ao mesmo nível de eficiência, que é próximo a 80%. Isso indica que o speed-up tem característica de linearidade quanto ao número de escravos. A explicação para esse fenômeno se dá porque o speed-up possui característica muito próxima da linear, mas devido à influência do overhead entre os períodos de execução (seja pelo envio de "Guesses" ou outros fatores), o tempo é acrescido de uma proporção que varia conforme a quantidade de escravos.

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, uma solução distribuída é mais rápida, o que permite concluir que o problema de ataque de dicionário em mensagem criptografada é uma operação muito cara para ser realizada de forma serial. Isso significa que, para o caso serial, a maior parte do tempo da operação é percorrendo cada palavra do dicionário e testando se cada uma é uma palavra-chave. Quando os índices das palavras são divididas para cada escravo, o espaço de busca diminui e, por consequência, a operação torna-se mais rápida, já que as respostas que cada escravo envia não dependem do processamento de outros escravos.