2º Trabalho

Processamento Paralelo e Distribuído 2019/01
Ataque de Dicionário em Mensagem Criptografada com Middleware Orientado a Mensagens

Prof. João Paulo Andrade Almeida

7 de junho de 2019

O objetivo deste trabalho é praticar programação paralela usando *middleware* orientado a mensagens JMS e realizar análise de desempenho em um *cluster* de computadores. O trabalho inclui a comparação desta implementação com a implementação do primeiro trabalho (baseada em *Request-Response* com Java RMI).

O trabalho deverá ser realizado em duplas, de preferência com a mesma composição dos grupos no primeiro trabalho de implementação. Plágio (cópia entre diferentes grupos) não será tolerado, e grupos com trabalhos copiados (mesmo que parcialmente) receberão nota mínima no trabalho como um todo.

A organização do seu código, clareza e detalhamento dos comentários é um critério importante de avaliação. Documente *bugs* conhecidos ("*known bugs*") no seu relatório. (*Bugs* conhecidos documentados são melhores do que *bugs* desconhecidos e não documentados.)

Inclua o código fonte e o relatório (PDF) em um arquivo `.zip` nomeado com `ataque-dicionario-jms` seguido dos nomes dos componentes do grupo separados por hífen. (Exemplo: `ataque-dicionario-jms-JoseSilva-AntonioVieira.zip`). Envie o arquivo para jpalmeida@inf.ufes.br com o nome do arquivo no campo "subject". O relatório impresso deve ser entregue no escaninho do professor no Departamento de Informática. **O prazo para entrega é dia 8 de julho de 2019. Não haverá prorrogação.**

1 Implementando um ataque de dicionários com a arquitetura mestre-escravo

Será empregada a mesma arquitetura mestre-escravo do primeiro trabalho de implementação. O mestre oferecerá a mesma interface para os clientes. Assim como no primeiro trabalho, use o registry para registrar e achar o mestre, com o nome "mestre".

A interação com os escravos, porém, será diferente neste segundo trabalho:

Altere o mestre para que delegue o ataque a vários escravos usando mensagens em uma fila (*queue*) JMS. O mestre receberá as respostas dos escravos através de uma segunda fila.

Os escravos deverão consumir mensagens da fila denominada "SubAttacksQueue". Cada mensagem descreverá o sub-ataque que pode em princípio ser efetuado por um escravo qualquer. O formato dessa mensagem deve ser especificado pelo grupo e documentado no relatório.

Note que o mestre não sabe o número de escravos disponíveis, e, dessa forma, a partição dos índices do dicionário entre os vários escravos será feita de uma forma diferente do

primeiro trabalho. Neste caso, o número de palavras a ser testada por cada escravo (*m*) é variável e deverá ser objeto de estudo e ajuste do seu grupo. (Naturalmente, a partição dos índices do dicionário deve resultar na busca do dicionário por completo, então, considerando *d* o número de palavras no dicionário, há *d*/*m* sub-ataques.)

Um escravo deve consumir e realizar apenas um sub-ataque por vez, apenas consumindo um próximo sub-ataque quando o ataque anterior terminar.

Ao achar uma chave candidata, cada escravo informará esta chave ao mestre através de uma segunda fila (compartilhada entre todos os escravos e de nome "GuessesQueue".) Ao terminar o sub-ataque, o escravo deverá sinalizar o fim do sub-ataque através desta mesma fila.

Documente claramente no relatório os formatos das mensagens trocadas entre mestre e escravos, indicando também se cabeçalhos JMS são utilizados e como.

A Figura 1 representa o padrão arquitetural sendo aplicado na relação entre mestre e escravo (o *Competing Consumers* pattern). Quando um escravo consome uma mensagem oriunda do mestre, outros escravos não podem consumir a mesma mensagem.

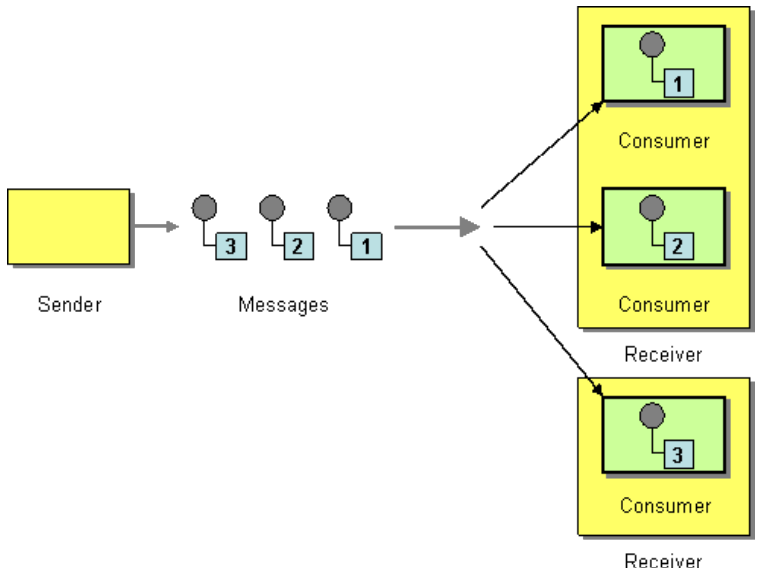

Figura 1 Competing Consumers Pattern[1]

Diferentemente da primeira implementação, **não** haverá *checkpoint* de escravos, ou outro mecanismo de recuperação de falhas.

Inclua no relatório todos os comandos utilizados para inicializar o registry, o cliente, o mestre e os escravos.

Inclua arquivos de teste utilizados (na faixa de 1KB a 50KB ao menos) e descreva os testes do seu relatório.

2 Programa Cliente

O programa cliente deve seguir as especificações do primeiro trabalho de implementação.

[1] http://www.enterpriseintegrationpatterns.com/patterns/messaging/CompetingConsumers.html

3 Mestre

O mestre deve imprimir uma mensagem na saída padrão a cada chave candidata recebida dos escravos; incluir o nome do escravo nesta mensagem e a chave candidata. O mestre também deve imprimir uma mensagem na saída padrão ao fim de um sub-ataque.

4 Análise do Desempenho

Faça uma análise do tempo de resposta observado pelo cliente ao requisitar um ataque.

Use 2 configurações de teste:

- 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).

Para cada uma dessas configurações, crie gráficos para análise, usando (pelo menos):

- Diferentes tamanhos de mensagem criptografada

    (crie um gráfico do tempo de resposta x tamanho da mensagem)

- Diferentes valores para o parâmetro *m* na faixa de 100 a 40000.

Usando os resultados de sua análise, ajuste o valor de *m* para minimizar o tempo de resposta observado pelo cliente.

Compare o desempenho (tempo de resposta) desta implementação com *m* escolhido com a implementação do primeiro trabalho, nas 2 configurações de teste acima.

Documente: tipo de máquinas usadas, sistema operacional, memória disponível, configuração de rede. Se necessário, use o laboratório de graduação.

O grau de detalhamento da análise é um fator importante para a avaliação (inclua outros gráficos e dados relevantes para a sua análise).

O que você pode concluir a partir da análise? Inclua suas conclusões no relatório.

5 Infraestrutura

Você deverá usar a plataforma Glassfish (Full Platform), versão 5.0, que pode ser baixada em: (Atenção, não funciona com JDK 9, ... requer JDK 8.)

https://javaee.github.io/glassfish/download

no Linux, use:

```
wget http://download.oracle.com/glassfish/5.0/release/glassfish-5.0.zip
```

E descompacte o zip em um diretório qualquer.

Para iniciar o servidor (o comando ‘asadmin’ fica no diretório bin do glassfish):

```
asadmin start-domain
```

As filas deverão ser criadas com o seguinte comando (com o arquivo glassfish-queues.xml que será disponibilizado na página da disciplina e descrito na seção seguinte):

```
asadmin add-resources glassfish-queues.xml
```

6 Descritor de Recursos para o Glassfish

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE resources PUBLIC "-//GlassFish.org//DTD GlassFish Application Server 3.1
Resource Definitions//EN" "http://glassfish.org/dtds/glassfish-resources_1_5.dtd">
<resources>
  <admin-object-resource enabled="true" jndi-name="jms/SubAttacksQueue"
          object-type="user" res-adapter="jmsra" res-type="javax.jms.Queue">
    <description/>
    <property name="Name" value="SubAttacksQueue"/>
  </admin-object-resource>

  <admin-object-resource enabled="true" jndi-name="jms/GuessesQueue"
          object-type="user" res-adapter="jmsra" res-type="javax.jms.Queue">
    <description/>
    <property name="Name" value="GuessesQueue"/>
  </admin-object-resource>

</resources>
```