INF1406 – 2017.1

Exercício 3: Multiplicação Distribuída de Matrizes no modelo Produtor X Consumidor

1. Neste exercício o objetivo é incrementar o trabalho anterior, para que funcione numa arquitetura distribuída. A aplicação distribuída deve ser composta de servidores e clientes capazes de, em conjunto, multiplicar **conjuntos** de várias matrizes de tamanho fixo, N x N, realizando os cálculos independentes de forma paralela. O cálculo de matrizes é dado pela seguinte fórmula:

$$(AB)_{ij} = \sum_{k=1}^{m} A_{ik} B_{kj}.$$

2. Existem 4 programas ou "atores" no cenário do trabalho. **CADA UM DEVE TER SEU PROJETO ECLIPSE SEPARADO,** pois são programas independentes que podem ser executados em máquinas diferentes. Esses atores são chamados de:
   a. **Produtor** (deve existir apenas um)
   b. **Consumidor** (podem existir vários)
   c. **Servidor de Execução** (podem existir vários)
   d. **Configurador** (podem existir vários)
   e. Você deve assumir que podem existir vários Consumidores, Servidores de Execução e Configuradores e, portanto, tratar a concorrência de acordo, mas **no seu trabalho só precisa utilizar um de cada.**

3. O Produtor torna disponíveis conjuntos de matrizes a ser multiplicados. Para tanto, deve:
   a. O programa deve receber o seguinte parâmetro:
      i. O caminho para o arquivo "matrizes.txt".
   b. O arquivo "matrizes.txt" conterá nomes de arquivos de matrizes (cada arquivo destes, no mesmo formato do trabalho 2), um por linha. Em cada linha haverá ainda um número com a dimensão das matrizes e outro número com o número de matrizes que o arquivo contém. Exemplos de linhas:
      i. C:/inf1406/trab3/conjunto1.txt 3 5000
      ii. C:/inf1406/trab3/conjunto2.txt 28 2037
   c. O programa deve ter uma thread de leitura de arquivos que deve dormir durante 1minuto após ler um arquivo de matrizes e, em seguida, ler o próximo arquivo. Isso deve ser repetido até que não haja mais arquivos a ler.
   d. O programa deve exportar um objeto remoto RMI, implementando uma **interface remota** para uso dos Consumidores chamada "**Produtor**", que deve ter um método com a seguinte assinatura:

i. ConjuntoMatrizes obtemMatrizes() throws RemoteException;

e. O método “obtemMatrizes” retorna o conjunto de todas as matrizes de um dos arquivos lidos, para que um Consumidor possa realizar a multiplicação. Esse conjunto retornado deve ser removido do Produtor. Caso não haja nenhum conjunto, pode-se retornar *null*.
f. “ConjuntoMatrizes” deve ser uma **interface serializável** definida por você, no formato que achar melhor.
g. Como mencionado anteriormente, você deve assumir que vários Consumidores podem acessar o Produtor ao mesmo tempo e, portanto, o acesso concorrente deve ser tratado para evitar condições de corrida.
h. Por fim, o Produtor deve ainda exportar um segundo objeto remoto RMI, implementando uma **interface remota** de configuração de seu intervalo de tempo de leitura do arquivo “matrizes.txt”, chamada “**Configuracao**”, com o seguinte método:
    i. void aplicaIntervalo(int intervalo) throws RemoteException;
i. O método “aplicaIntervalo” serve para mudar o intervalo de leitura de arquivos de matrizes (que começa como 1 minuto por padrão) e será usado pelo Configurador. Leve em consideração se é necessário tratar concorrência na implementação deste método ou não.

4. O Consumidor é quem está interessado em realizar a multiplicação de uma sequência de matrizes, como no trabalho anterior. No entanto, assumiremos que a máquina onde o consumidor roda não é potente o suficiente para executar os cálculos em um tempo aceitável. Portanto, o consumidor deve paralelizar a execução dos cálculos, enviando tarefas independentes para um ou mais Servidores de Execução.
    a. O programa deve receber o seguintes parâmetros:
        i. O endereço IP do RMI Registry onde o Produtor está publicado.
        ii. A porta do RMI Registry onde o Produtor está publicado.
        iii. O endereço IP do RMI Registry onde o Servidor de Execução está publicado.
        iv. A porta do RMI Registry onde o Servidor de Execução está publicado.
    b. Quem deve criar a tarefa, ou seja, a classe que realiza o cálculo independente, é o Consumidor (essa classe que realiza o cálculo não deve existir no Servidor de Execução). Faça essa classe estender uma das interfaces aceitas por um ThreadPool.
    c. A tarefa deve ser passada por valor. Portanto, deve ser implementada como uma classe **serializável**. Ela deverá realizar a multiplicação de uma linha por uma coluna (como no trabalho anterior).

d. O Consumidor deve eternamente checar, a cada meio segundo, se há um conjunto de matrizes a ser multiplicado disponível no Produtor. Se houver, inicia o procedimento de multiplicação, que consiste em dividir a multiplicação em tarefas, enviá-las a Servidores de Execução, e somente então aguardar pelas respostas. Dica: ao invés de implementar objetos futuros remotos, utilize *callbacks* para retornar os resultados e semáforos para aguardar por eles.
e. O Consumidor deve, ao final da multiplicação de todas as matrizes de um conjunto, imprimir em um arquivo "resultados.txt" o resultado dessa multiplicação seguido de uma linha vazia e voltar a perguntar ao Produtor se há outro conjunto a multiplicar. O arquivo "resultados.txt" deve, ao final do programa, ter os resultados de todas as multiplicações de conjuntos de matrizes, com cada matriz resultado separada da próxima por uma linha em branco.
f. A execução de uma tarefa no Servidor de Execução deve ser feita de forma assíncrona. Sugestão: o Consumidor pode exportar um objeto distribuído RMI com uma interface chamada "**Callback**", com o seguinte método:
    i. void entregaResultado(Resultado resultado) throws RemoteException;
g. Esse objeto remoto da interface "Callback" **NÃO DEVE SER PUBLICADO NO RMI REGISTRY**, deve ser passado para o Servidor de Execução e utilizado após terminar de executar uma tarefa ou passado para a própria tarefa e utilizado por ela mesma antes de terminar (lembre que a tarefa executa no Servidor de Execução).
h. A classe "Resultado" deve ser passada por valor e portanto será uma classe **serializável**. Ela deve conter todas as informações necessárias para entregar um resultado ao Consumidor.
i. Você pode alterar a interface de entrega dos resultados se quiser, desde que esta ocorra de forma assíncrona.
j. O Consumidor deve exportar um outro objeto distribuído RMI com a mesma **interface** "**Configuracao**" do Produtor. Nesse caso, a interface servirá para configurar o tempo que o Consumidor espera para perguntar ao Produtor se existem novos conjuntos de matrizes a ser multiplicados. Novamente, avalie a questão de concorrência.

5. O Servidor de Execução simplesmente executa tarefas genéricas, das quais não tem conhecimento *a priori* da implementação.
    a. O programa deve receber o seguinte parâmetro:
        i. O número de *threads* a criar no seu *pool*.
    b. O programa deve exportar um objeto remoto RMI, implementando uma **interface remota** para uso dos Consumidores chamada "**Execucao**", que deve ter um método para a execução remota de uma tarefa. Sugestão de assinatura:
        i. void execute(Runnable tarefa) throws RemoteException;

c. Você pode alterar a assinatura desse método, se assim desejar.
d. Idealmente tarefas devem ser executadas em um *thread pool*, como no trabalho 2, mas também será aceita a gerência manual das *threads* se assim preferir.
e. A única interface que o Servidor de Execução precisa ter inicialmente em seu *classpath* é a "Execucao", que ele mesmo exporta. Todas as outras podem ser transferidas dinamicamente.

6. O Configurador é uma ferramenta interativa de configuração dinâmica dos tempos utilizados pelo Produtor e Consumidor.
    a. O programa deve receber o seguintes parâmetros:
        i. O endereço IP do RMI Registry onde o Produtor está publicado.
        ii. A porta do RMI Registry onde o Produtor está publicado.
        iii. O endereço IP do RMI Registry onde o Consumidor está publicado.
        iv. A porta do RMI Registry onde o Consumidor está publicado.
    b. Deve iniciar perguntando ao usuário se deseja alterar o intervalo do produtor ou do consumidor.
    c. Após a escolha do usuário, usa a interface "Configuracao" do serviço correspondente para realizar a configuração em tempo de execução.
    d. Por fim, pode terminar a execução ou voltar à pergunta inicial.

7. Responda à seguinte pergunta em seu email de entrega (ou adicione um arquivo com a resposta ao pacote):
    a. Você vê alguma dificuldade ou problema no uso do RMI Registry em geral? Imagine que quiséssemos executar um número indefinido de Produtores, Servidores de Execução, Consumidores e Configuradores.

8. Seu trabalho deve contemplar:

    - Quatro projetos Eclipse Java (ou um script Windows responsável por compilar corretamente o código e outro responsável por executar o código, para cada programa).

    - O código-fonte da sua solução. A linguagem utilizada deve ser Java.

    Observações:

    - A JVM a ser utilizada deve ser a 1.6 ou maior.

    - O trabalho deve ser feito em duplas.

- O prazo do trabalho será informado em sala de aula.

- Você deve enviar o seu trabalho por email para caeaugusto@gmail.com ou caugusto@inf.puc-rio.br .

- A apresentação será feita durante o horário de aula, na sala de aula. Caso não possa comparecer, você deve enviar o trabalho por email até as 17h do dia de entrega e agendar outro horário de apresentação (com perda de pontos).

- Cada dia de atraso incorre na perda de um ponto na nota. Cada dia de atraso na apresentação (entregando o trabalho até a data limite mas apenas não apresentando) incorre na perda de meio ponto na nota.

- Durante a apresentação, serão feitas perguntas para avaliar o seu entendimento do problema, da solução, dos erros de concorrência e suas soluções. A nota será baseada nas suas respostas e no fato dos seus programas funcionarem como especificado ou não.

- Não é objetivo do trabalho o estudo de algoritmos de multiplicação de matrizes. Portanto, qualquer solução para o problema será aceita, desde que realize os cálculos de forma paralela.

- Avalie a diferença no tempo total gasto de acordo com a quantidade de *threads*. Caso obtenha sempre resultados muito semelhantes, independente do número de *threads*, você precisará utilizar matrizes com N maior ou artificialmente gastar mais tempo em cada cálculo. Não utilize *Thread.sleep()* ou outra forma de bloqueio da *thread*, como *wait()*. Você pode incluir um *loop* finito, por exemplo.

- Não esqueça das modificações no comportamento das últimas versões de Java em relação ao *codebase*. Maiores informações aqui.