EP1 - Sistemas P2P

SISTEMAS DISTRIBUÍDOS

Nome: Igor Carvalho de Oliveira
RA: 11201920763 | **Turma:** A2N
Apresentação: Link para Loom
Github: ufabc-distributed-system

Introdução

O objetivo desse trabalho é criar uma rede P2P (Peer-to-peer) que realize a transferência de arquivos de vídeos com mais de 1GB utilizando RMI e TCP como protocolo de comunicação. A aplicação não terá a arquitetura 100% P2P pois teremos um servidor de controle parecido semelhante ao Napster a fim de guardar os IPs e portas dos peers que estão conectados com a rede. A motivação deste trabalho é compreender a dificuldade de projetar uma aplicação P2P, visto que existem vários desafios que precisam ser resolvidos.

Descrição do sistema P2P

Este relatório descreve as funcionalidades e o uso do servidor central implementado em Java, que utiliza RMI (Remote Method Invocation) para transferência de arquivos entre peers remotos. O servidor central desempenha um papel fundamental no gerenciamento e compartilhamento de arquivos na rede. Ele permite que os peers remotos se conectem, pesquisem por arquivos disponíveis, realizem transferências de arquivos e atualizem os arquivos armazenados no servidor. O servidor oferece uma interface remota chamada FileTransfer, que define os métodos disponíveis para interação com os peers remotos.

Funcionalidades

Este relatório descreve as funcionalidades e o uso do servidor central implementado em Java, que utiliza RMI (Remote Method Invocation) para transferência de arquivos entre peers remotos. O servidor central desempenha um papel fundamental no gerenciamento e compartilhamento de arquivos na rede. Ele permite que os peers remotos se conectem, pesquisem por arquivos disponíveis, realizem transferências de arquivos e atualizem os arquivos armazenados no servidor. O servidor oferece uma interface remota chamada FileTransfer, que define os métodos disponíveis para interação com os peers remotos.

FUNCIONALIDADES DO SERVIDOR

- Registrar Peer (**join**)
  - Os peers podem se juntar ao servidor central para compartilhar seus arquivos. Utilizando o método `join` na interface *FileTransfer*, um peer informa seu nome, nome do arquivo que possui, seu endereço IP e porta. Desta forma, essas informações ficam guardadas no servidor e qualquer outro peer consegue consultá-las se for necessário.
  - No momento de cadastro, o servidor central verifica se o peer já está registrado e, caso não esteja, adiciona-o à lista de peers registrados, juntamente com o nome do arquivo que possui.
- Pesquisar de arquivos em outros Peers (**search**)
  - O servidor central permite que os peers procurem por arquivos disponíveis em outros peers. Através do método `search` na interface *FileTransfer*, os peers podem enviar uma solicitação de pesquisa informando o nome do arquivo desejado.
  - O servidor realiza a busca em sua lista de peers registrados e retorna uma lista de peers que possuem o arquivo solicitado.
- Atualizar os arquivos do Server (**update**)

    - Essa foi a única funcionalidade que não foi implementada. A expectativa era que o servidor central permita que os peers atualizem arquivos armazenados nele. A funcionalidade de atualização seria realizada através do método `update` na interface *FileTransfer*.
- Receber arquivos enviados por outros Peers (**receiveFile**)
    - O servidor central é capaz de receber arquivos enviados por outros peers. Ao utilizar o método `receiveFile` na interface *FileTransfer*, um peer pode enviar um arquivo para ser armazenado no servidor central.
- Realizar download de Arquivos (**downloadFile**)
    - A expectativa é que um peer remoto baixe um arquivo específico de outro peer remoto. O método `downloadFile` na interface FileTransfer é responsável por essa funcionalidade. Porém, tive dificuldades em implementar o código e a solução ficou incompleta.
    - Basicamente, o peer requisitante deve fornecer o nome do peer de origem, o nome do arquivo, além do endereço IP e porta para estabelecer a conexão o peer detentor do arquivo. Por sua vez, este pior decide se vai liberar o download para o peer requisitante.
    - O código foi projetado para que a transferência entre peers não dependa do servidor. Portanto, a comunicação entre eles é feita utilizando TCP.

Requisitos do Sistema

REQUISITOS FUNCIONAIS

- **RF-01:** Servidor de controle rodando
- **RF-02:** Registrar peer no servidor de controle
- **RF-03:** Procurar peer no servidor de controle
- **RF-04:** Download do arquivo que um peer estiver servindo
- **RF-05:** Servidor procurar dado na sua estrutura de dados
- **RF-06:** Servidor registrar dados na sua estrutura de dados

REQUISITOS NÃO-FUNCIONAIS

- **RNF-01:** Facilidade em executar o código para entender conceitos básicos sobre p2p.

Implementação do Projeto

O servidor central atua como um intermediário que facilita a comunicação e o gerenciamento dos arquivos compartilhados. Segue um resumo do que foi implementado:

1. Primeiramente, os peers têm a capacidade de se conectar ao servidor central, registrando-se e tornando seus arquivos disponíveis para outros peers. Essa funcionalidade permite que a rede seja expandida à medida que novos peers se juntam.
2. O servidor central oferece a capacidade de pesquisa, permitindo que os peers remotos encontrem arquivos específicos em outros peers conectados à rede. Isso agiliza o processo de localização de arquivos desejados, eliminando a necessidade de pesquisar individualmente em cada peer.
3. Os peers podem solicitar arquivos específicos de outros peers através do servidor central. O servidor atua como um intermediário, recuperando o arquivo solicitado do peer correspondente e fornecendo-o ao solicitante. Isso simplifica o processo de transferência de arquivos e melhora a eficiência na rede.
4. O servidor central permite que os peers enviem arquivos uns aos outros. Quando um peer deseja enviar um arquivo para outro peer, ele pode usar o servidor central como intermediário. O arquivo é enviado ao servidor, que o encaminha para o peer de destino. Isso

garante que os arquivos sejam transferidos de forma confiável e eficiente entre os peers remotos.

O desenvolvimento do sistema em si foi feito utilizando a técnica orientação a objetos em que sua estrutura foi dividida em 2 classes principais: Peer e Server. A classe Peer efetua as seguintes ações: upload de arquivos para o server, e download de arquivos, enquanto a classe Server armazena todos os dados da rede.

O peer é iniciado ao chamar o método `start()`, que realiza as seguintes etapas:

1. Obtém o endereço IP local do peer e a porta disponível para escuta de conexões.
2. Inicia um servidor de socket para receber conexões de outros peers.
3. Inicia uma thread para lidar com as conexões recebidas.
4. Obtém uma referência para o registro RMI (Remote Method Invocation) do servidor central.
5. Obtém uma referência para o objeto remoto `FileTransfer` no servidor.
6. Cria um objeto `Scanner` para receber as entradas do usuário.
7. Exibe um menu de opções para o usuário, permitindo que ele escolha entre JOIN, SEARCH, DOWNLOAD ou Sair.
8. Com base na opção escolhida pelo usuário, o código interage com o servidor central e realiza as ações correspondentes, como se juntar à rede, pesquisar por arquivos ou fazer o download de um arquivo.
9. O código também contém tratamentos de erro para lidar com situações excepcionais, como arquivos não encontrados ou falhas na conexão.

Além disso, o código possui métodos auxiliares, como `handleRequest()` e `processRequest()`, que são responsáveis por lidar com as requisições recebidas de outros peers e processá-las de acordo com a escolha do usuário.

O sistema foi desenvolvido utilizando a linguagem Java junto com tecnologia de sockets e o protocolo de transporte TCP. Vale ressaltar que foi feita uma adaptação no código para permitir a comunicação direta entre os peers, sem a necessidade do servidor central como intermediário. Primeiramente, foi necessário ajustar a lógica de inicialização dos peers para obter o endereço IP e a porta em que cada peer está executando. Isso foi feito utilizando as classes `InetAddress` e `ServerSocket` para obter o endereço IP local e selecionar uma porta disponível. Em seguida, foi criado um mecanismo de troca de mensagens entre os peers, por meio da implementação de sockets TCP. Utilizando a classe `Socket`, os peers conseguem estabelecer conexões diretas uns com os outros, permitindo a troca de informações e a realização de operações, como busca e download de arquivos.

Em resumo, foram adicionadas duas funcionalidades principais nessa adaptação: a busca por arquivos e o download. Para a busca, cada peer pode enviar uma requisição de pesquisa diretamente a outros peers, que respondem informando se possuem o arquivo desejado. Já para o download, o peer requisitante pode estabelecer uma conexão com o peer que possui o arquivo e realizar o download diretamente a partir dessa conexão. O servidor de controle utiliza uma tabela hash (Dicionário em Java) e a tecnologia de Threads também foi utilizada para que vários Peers possa se conectar de forma paralela com o Servidor de Controle.

Materiais de Consulta

- Utilizado para direcionar a arquitetura do projeto: https://github.com/brunoqs/try-p2p
- Utilizado para consultar a sintaxe da linguagem Java: https://chat.openai.com/chat