PUCRS

Pontifícia Universidade Católica
do Rio Grande do Sul

Programação Distribuída
Trabalho 2

Escola Politécnica – Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
CEP 90619-900 – Porto Alegre – RS – Brasil

Bernardo de Cesaro
bernardo.cesaro@acad.pucrs.br

Resumo. *Este artigo tem como objetivo apresentar os resultados obtidos a partir do desenvolvimento do segundo trabalho da disciplina de Programação Distribuída. Será mostrado todos os aspectos da implementação e o que foi levado em consideração durante o desenvolvimento. Ao final, é apresentado os resultados obtidos, assim como eventuais dificuldades encontradas.*

1 Introdução

O objetivo deste trabalho é a implementação de um sistema P2P básico, que deve ser organizado como uma arquitetura centralizada, onde o controle de toda a aplicação (lógica e estado) é concentrado em um *web service*.

2 Problema

Na questão de como deve ser desenvolvido a aplicação, alguns pontos foram levantados pelo enunciado deste trabalho prático. Por exemplo, os peers devem se registrar no servidor para poderem realizar a troca de arquivos entre peers. Isso deve ser feito por meio de uma REST. Durante o registro, cada peer informa seus recursos disponíveis e para cada arquivo, o peer fornece ao servidor uma string ou o nome do arquivo e sua hash. O servidor então associa cada recurso em uma estrutura de dados, onde cada recurso possui associado o IP do peer onde está o recurso e sua hash.

Os peers podem solicitar uma lista de recursos (nomes dos arquivos / strings de identificação, IPs dos peers que contém os recursos e hashes) ao servidor ou um recurso específico. Isso também deve ser feito por meio de uma REST.

Ao solicitar um recurso ao servidor, o peer recebe a informação sobre sua localização (outro peer) e deve então realizar essa comunicação diretamente com o mesmo. Sobre a comunicação entre os peers, a mesma deve ser realizada com sockets.

O servidor é responsável por manter a estrutura da rede de overlay. Para isso os peers devem consumir uma REST do servidor a cada 5 segundos. Caso um peer não fizer duas solicitações seguidas a um servidor, o mesmo é removido.

3 Solução

O código da aplicação está devidamente comentado, mas a seguir será apresentado uma breve explicação sobre cada método/classe desenvolvidos para solucionar o problema deste trabalho.

3.1 Server

3.1.1 *registerNewClient*

Adiciona um novo cliente e os seus respectivos recursos à uma lista de clientes registrados e envia esta confirmação então ao cliente.

3.1.2 *removeClient*

Remove um cliente da lista.

3.1.3 *checkConnectedClients*

Verifica se existem clientes inativos depois de um determinado tempo. Caso o cliente esteja inativo após x tempo, ele é removido da lista de clientes.

3.1.4 *filterResourcesFromClient*

“Pega” todos os recursos que possuem um ip ou uma porta diferente.

3.1.5 *sendResourceList e sendResourceInfo*

Dois métodos relacionados a relação entre o cliente e os recursos. O primeiro método é destinado a enviar ao cliente a lista atual de recursos existentes. Já o segundo é destinado a enviar informações sobre determinado recurso ao cliente.

3.1.6 *refreshClient*

Reinicia o *timer* de inatividade do cliente.

3.2 Client

Na classe cliente, grande parte da exibição/ações entre arquivo/cliente é implementado. Alguns dos principais métodos do cliente são:

3.2.1 *sendFileList*

Envia uma lista com os nomes, *hashes* e tamanhos do arquivo para o servidor.

3.2.2 *handleResourceRequest*

Salva as informações do arquivo e envia uma mensagem ao dono do arquivo pedindo pela transferência do arquivo.

3.2.3 *handleFileSection*

Salva a parte do arquivo recebida e envia uma mensagem de confirmação ao outro *peer.*

3.3 FileReader

3.3.1 *hashFile*

Neste método será feito o cálculo do hash de um único arquivo, utilizando md5.

3.3.2 *createFile*

Escreve o arquivo na pasta de download.

3.4 Network

Classe responsável pela conexão entre os peers e pela manipulação do tamanho que cada arquivo será enviado entre eles. Nesta implementação, a ideia foi dividir o arquivo enviado em pacotes/fragmentos, de tamanho máximo de 44KB. Cada pacote é enviado apenas quando o peer recebe o seu antecessor.

4 Estrutura Lógica

O programa foi escrito em JavaScript. Ou seja, o ponto de entrada para iniciar a aplicação é executar o seguinte comando:

npm install

Para iniciar um servidor, execute o comando:

npm start -- --server

Para iniciar um cliente, execute:

npm start -- --files=<folder path> --address=<server address> -- port=<server port>

5 Dificuldades Encontradas

Dentre as dificuldades encontradas durante a realização deste trabalho, o principal problema foi desenvolver toda essa aplicação sozinho. Em outras palavras, foi custoso ter que desenvolver algumas partes lógicas do programa sem ter outra pessoa dando uma visão externa sobre a implementação.

Acredito também que a forma utilizada neste trabalho em relação a uma REST poderia ter sido melhor abordada.