Sistemas Distribuídos

Network File-System

Marco Pontes — Nuno Azevedo

up201308000@fc.up.pt — up201306310@fc.up.pt

14 de Novembro de 2016

1 Abstract

O objetivo deste projeto é a implementação de um sistema de ficheiros em rede (*Network File-System).* A comunicação cliente - servidor (utilizador - sistema de ficheiros virtual) assenta na tecnologia *Java RMI*[1]. O *Network File-System* suporta várias unidades de armazenamento assim como vários clientes.

2 Introdução

Um *Network File-System* é um sistema de ficheiros em rede constituído por uma ou mais unidades físicas de armazenamento, estando disponíveis para o cliente poder guardar os seus ficheiros. Este serviço tem como principal função criar um sistema virtual sobre servidores físicos com o objetivo de armazenar ficheiros de forma distribuída, em computadores interligados por rede.

De forma a aumentar o espaço disponível no *Network File-System*, pode ser necessário a inserção de mais do que uma unidade de armazenamento mas isso deve ser transparente ao cliente, pois este não tem de se preocupar com a localização real dos seus dados. Com isto, é necessário um servidor que unifique as unidades de armazenamento.

O servidor de meta-dados cria uma camada virtual sobre as unidades de armazenamento de forma a que virtualmente exista apenas um sistema de ficheiros. Sendo assim, este servidor é considerado o núcleo do sistema devido a fazer a ligação entre todos os componentes.

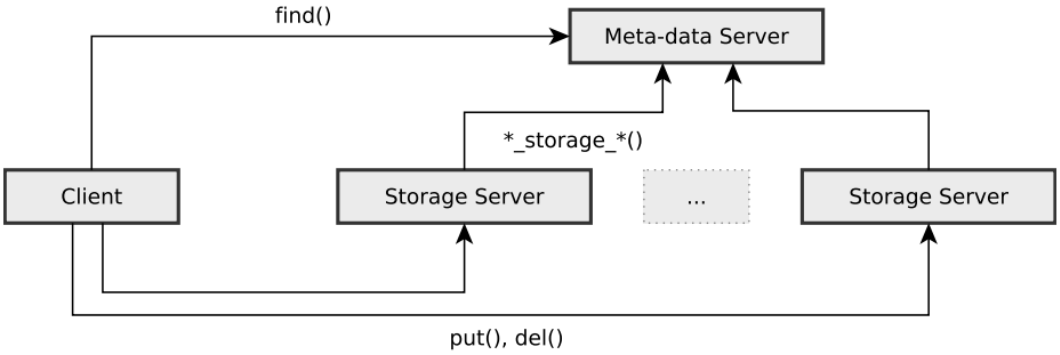
A comunicação entre o cliente, o servidor de meta-dados e as unidades de armazenamento é feita através da tecnologia *Java RMI* sendo este um mecanismo que permite a invocação de métodos residentes em diferentes máquinas virtuais *Java* (*JVM*), permitindo assim ao cliente a manipulação dos ficheiros no servidor remotamente.

[1] `https://docs.oracle.com/javase/tutorial/rmi/`

3 Descrição

Após todos os componentes estarem conectados e em sincronia, podemos ver um sistema de ficheiros como o representado na figura 1.

Figura 1: Interação entre o cliente (*Client*), o servidor de meta-dados (*Meta-data Server*) e as unidades de armazenamento (*Storage Server*).



Origem: `http://www.dcc.fc.up.pt/~rmartins/aulas/sd1617/trabI/trabI.pdf`

Como referido em cima, pode-se confirmar nesta figura que o servidor de meta-dados *(Meta-data Server)* tem as informações de todas as unidades de armazenamento *(Storage Servers)*.

Do ponto de vista do utilizador, este tem uma vista sobre um sistema de ficheiros de tal forma uniforme, opaco ao conjunto de unidades de armazenamento que o constituem, que tornam a navegação sobre o *Network File-System* semelhante a um sistema de ficheiros local.

Quanto à implementação, sempre que o cliente *(Client)* precisa de aceder a um ficheiro, tem de contactar o servidor de meta-dados para ficar a conhecer a localização real do ficheiro. Depois disso, pode aceder diretamente à unidade de armazenamento correspondente para efetuar as operações que desejar.

Quando o cliente fizer alguma alteração, a unidade de armazenamento onde essa alteração ocorrer fica responsável por informar o meta-dados do evento para este atualizar as suas informações.

3.1 Metodologia

Para o desenvolvimento do projeto usamos o *IDE IntelliJ IDEA*[2] de forma a tornar a programação mais rápida e eficiente, além de que nos ajudou imenso a conhecer os vários métodos disponíveis nas bibliotecas usadas durante o desenvolvimento do projeto.

Pela facilidade de trabalhar em grupo, de ter registos de toda a atividade do projeto, como também nos permitir ter o código guardado de forma segura, usamos a ferramenta *Git*, neste caso o *GitLab*[3] para manter o projeto privado gratuitamente.

Quanto à divisão das tarefas do projeto, optamos por repartir os métodos de cada classe entre nós, de modo a que cada um conseguisse perceber um pouco de cada componente (*Client, Meta-data Server e Storage Server*).

3.2 Interfaces/API Usada

- ***Storage Interface*** - Permite que o cliente interaja com um *Storage Server* através dos seguintes métodos:
  - ***create()*:** Cria um novo diretório/ficheiro, ou atualiza um ficheiro já existente.
  - ***del()*:** Apaga um diretório/ficheiro. No caso do diretório conter sub-diretórios/ficheiros, este método apaga todos as entradas recursivamente.
  - ***get()*:** Retorna o conteúdo do ficheiro da localização indicada na forma de *array* de *bytes*.
- ***Meta-data Interface***
  - Métodos invocados pelos *Storage Servers* - Permitem que um *Storage Server* comunique com o servidor de meta-dados informando de todas as alterações que ocorrem:
    - ***addStorageServer()*:** Existência de uma nova unidade de armazenamento.
    - ***delStorageServer()*:** Remoção de uma unidade de armazenamento.
    - ***addStorageItem()*:** Criação de um novo diretório/ficheiro.
    - ***delStorageItem()*:** Remoção de um diretório/ficheiro.
  - Métodos invocados pelo *Client* - Permitem fornecer ao cliente informações acerca dos ficheiros armazenados nos vários *Storage Servers*:
    - ***find()*:** Mostra em qual das unidades de armazenamento um determinado item está localizado.
    - ***lstat()*:** Lista todos as entradas num determinado diretório ou, no caso do argumento dado ser um ficheiro, mostra o ficheiro se ele existir.
    - ***checkExists()*:** Verifica se um determinado diretório/ficheiro existe.

[2] `https://www.jetbrains.com/idea/`
[3] `https://gitlab.com/`

* ***isDir()*:** Verifica se uma determinada entrada é um diretório.
* ***isFile()*:** Verifica se uma determinada entrada é um ficheiro.

3.3 Classes Principais e Suas Funcionalidades

- **StorageServer:** Representa uma unidade de armazenamento responsável pela gestão de ficheiros em disco. Antes de iniciar, verifica se já existem ficheiros no seu diretório local de forma a já os incluir no novo servidor. Efetua as alterações nos diretórios ou ficheiros tal como criar, remover ou alterar e, apenas no caso de ficheiros, enviar ficheiros para o cliente.
- **MetaDataServer:** Responsável pela ligação entre todos os *Clients* e o *Storage Servers*. Regista todas as alterações das unidades de armazenamento, incluindo a inserção ou remoção de novas unidades. Fornece ao cliente informações sobre os diretórios/ficheiros existentes nos vários *Storage Servers*.
- **Client:** Esta classe proporciona uma interface para o utilizar interagir com o sistema de ficheiros virtual. Conecta-se ao *Meta-data Server* apenas para verificar e obter informações sobre o conteúdo do sistema de ficheiros. Quanto à gestão dos seus ficheiros, o cliente conecta-se aos vários *Storage Servers* para criar e remover diretórios, ou, criar, alterar, remover e mover ficheiros.
- **FSTree:** Classe que representa a árvore do sistema de ficheiros virtual utilizada no *Meta-data Server*. Está capacitada para retornar as suas informações, como o nome, tipo, diretório parente e informações sobre os nós filhos de um determinado nó na árvore. Também permite adicionar e remover entradas na árvore, ou seja, criar e remover diretórios e ficheiros.
- **Stat:** Desempenha a função de transportar as informações de um determinado nó do sistema de ficheiros, como o seu nome e os seus filhos no caso de ser um diretório, para o cliente, quando este invoca um método para listar as informações de um determinado diretório/ficheiro.

3.4 Principais Problemas Encontrados e Soluções Propostas

- **Exceções:** Tivemos algumas dificuldades em lidar com todas as exceções lançadas pelos *Storage Servers* e *Meta-data Server* e ter um tratamento adequado para cada uma delas, de modo a que o sistema esteja protegido contra todas as falhas. Por vezes tivemos de propagar exceções lançadas pelo *Meta-data Server*, para um *Storage Server* e esse ter de propagar novamente para um *Client*.
- **Caminhos:** Implementámos uma função para fazer *parsing* de todo o tipo de caminhos, sejam eles absolutos, relativos ou até mesmo contendo a sintaxe ".." para indicar o diretório parente. Esta função apesar de não ser nada complexa, demorou algum tempo até funcionar sem problemas, pois devido ao número de possibilidades havia sempre algum caso de falha.

- **Unidades de Armazenamento:** Um dos problemas que nos acompanhou desde o início da implementação foi como lidar com um número dinâmico de *Storage Servers*. Quando o cliente se liga ao sistema de ficheiros não tem qualquer conhecimento sobre como este está constituído, e para realizar alguma alteração usando *Java RMI*, é necessário o *stub* da *interface* de cada *Storage*. A solução passou por utilizar o *Metadata Server* como o fornecedor do *hostname* de cada *Storage* através da função *find()* para depois conseguir fazer um *lookup()* no registo do *RMI* com esse *hostname* e assim conectar-se aos diversos *Storage Servers*.

- **Transferência de Ficheiros:** Relativamente a esta ação, a dificuldade que sentimos foi como conseguir transferir ficheiros de qualquer tipo, pois numa implementação inicial o nosso programa apenas suportava a transferência de ficheiros de texto, pois utilizávamos *strings* para representar o conteúdo dos ficheiros. Para transferir por exemplo imagens este método não funcionou, portanto após uma pesquisa mais aprofundada verificámos que a melhor solução seria transferir o ficheiro através de um *array* de *bytes*, para que assim fosse possível transferir qualquer tipo de ficheiro.

4 Conclusão

De uma forma geral, conseguimos implementar um sistema de ficheiros em rede totalmente funcional, capaz de responder a todos os pedidos de um utilizador.

Não tivemos oportunidade de testar o projeto num ambiente distribuído, no entanto, tivemos sempre a preocupação de a implementação estar preparada para tal, por isso cremos que não existirão muitos problemas para tornar o sistema de ficheiros funcional ao ser composto por diferentes servidores.

A realização deste trabalho permitiu o aprofundamento do nosso conhecimento em vários temas. Nomeadamente, para perceber como um sistema de ficheiros em rede funciona, apesar de que num nível básico, e a forma como realiza a comunicação entre os seus componentes usando o mecanismo *Java RMI*. Permitiu também aprender um pouco mais sobre a programação orientada a objetos.

Consideramos que o tema do trabalho foi adequado para testar a aplicação desta tecnologia e nos deu bastantes ideias para a colocar em prática numa situação real.