SSC0904 - Sistemas Computacionais Distribuídos

Docente: Rodolfo Ipolito Meneguette

Marcus Vinícius Teixeira Huziwara 11834432

Projeto Final

Sistema de Arquivos Distribuídos

São Carlos - SP
2023

Introdução

O objetivo do projeto foi elaborar um sistema distribuído para armazenamento de arquivos. Neste relatório serão apresentados os componentes do sistema, os protocolos de comunicação, as operações sobre os arquivos, a persistência do sistema, a autenticação dos usuários e a replicação dos arquivos. Assim como serão apresentadas as instruções para o uso do sistema.

Arquitetura

O sistema desenvolvido possui uma arquitetura similar ao do Hadoop Distributed File System (HDFS), onde há um servidor mestre e os minions. O mestre é responsável por conter os metadados sobre os arquivos e os minions armazenam os blocos de dados dos arquivos do cliente.

Comunicação

Para a comunicação entre os componentes foi utilizada a biblioteca RPyC do Python que implementa chamadas de procedimento remoto, assim como arquiteturas HDFS utilizam protocolos RPC customizados sobre o TCP.

Funcionamento

- **Gravar (Put)**

O cliente passa os blocos para o minion e este minion passa o dado para o próximo minion. Sendo o nó mestre responsável por alocar os blocos e o minion que irá receber o arquivo.

- **Ler (Get)**

O mestre fornece a lista de blocos e suas localizações do arquivo para o cliente tentar a leitura, se ela falha em um minion, o cliente tenta no próximo minion.

Implementação

- **client.py**
  Funções:
  1) get: chama a função read do master que retorna o dicionário com os blocos do arquivo caso ele exista. Para cada bloco, o cliente tenta conectar com o minion (usando a função rpyc.connect()) que o possui e realiza a leitura com a função get do minion.

  2) put: recebe como parâmetros, o path para o arquivo e o nome dele, calcula o tamanho do arquivo (com os.path.getsize()) e chama a função write do master com o nome e o tamanho. A função write retorna a lista de blocos e com essa lista o cliente inicia a conexão com o primeiro minion na lista e chama a função put do minion para iniciar a escrita.

  3) delete: requisita o dicionário de blocos do arquivo através da função read do master, em seguida, conecta e chama a função delete para cada minion que contém um dos blocos.

  4) authenticate: função para ler e verificar a autenticidade do usuário a partir do arquivo users.conf, o usuário tem 3 tentativas para inserir o nome do usuário e 3 para acertar a senha.

  5) main: chama a função authentic, se passar, inicia a conexão com o master e requisita o comando a ser realizado (put, get, exit). Para o comando get é necessário especificar o nome do arquivo e para o comando put é necessário o caminho do arquivo e seu nome.

- **master.py**
  Variáveis:
  1) file_block: dicionário que armazena os blocos de cada arquivo.
  2) block_minion : dicionário que armazena em que minions está cada bloco.
  3) minions: dicionário que contém as portas dos minions.
  4) block_size: inteiro com o tamanho dos blocos.
  5) replication_factor: inteiro como o fator de replicação.

  Funções:
  1) exposed_read: busca os blocos do arquivo e retorna lista com os ids do blocos e os endereços deles (o minion que contém).

2) exposed_delete: retorna um dicionário com os ids dos blocos e os minions que o contém, e deleta os dados do arquivo em file_block e block_minion.

3) exposed_write: calcula o número de blocos e chama função alloc_blocks para alocá-los.
4) alloc_blocks: gera o id dos blocos (com o uuid.uuid1()), escolhe aleatoriamente os minions que receberam cada bloco (com o random.sample()) e registra nos arquivos file_block e block_minion. Retorna um dicionário com os ids e seus minions. Caso ele não encontre minions para realizar a alocação, ele retorna -1.

- **minion.py**
  Funções:
  1) exposed_put: recebe o id do bloco, o conteúdo dele e os minions seguintes que também vão recebê-lo. No diretório do minion, ele cria o arquivo do bloco e escreve seu conteúdo. Se há mais minions para receber o bloco, é chamada a função forward().

  2) exposed_get: recebe o id do bloco, acessa o arquivo do bloco e retorna seu conteúdo.

  3) exposed_delete: recebe o id do bloco e remove ele do diretório do minion.

  4) forward: recebe o conteúdo do bloco com seu id e a lista de minions a receber esse bloco, ele realiza o rpyc,connect com o primeiro minions da lista e chama o put dele passando como parametro o id do bloco, o conteúdo e o resto da lista de minions.

- **users.conf**
  Arquivos com os nomes de usuários e suas senhas.

- **minion0, minion1, minion2**
  Diretórios de cada minion com os blocos dos arquivos.

- **block_minion.json e file_block.json**
  Arquivos com os metadados dos arquivos e dos seus blocos, garantem a persistência dos dados no master.

- **files**

Arquivos txt que serão armazenados nos minions.

- **clear.py**
  Código auxiliar para limpar diretórios e arquivos .json

Como rodar

> Iniciar os nós minions informando a porta e o diretório:

```
python3 minion.py 8000 ./minion0
python3 minion.py 8001 ./minion1
python3 minion.py 8002 ./minion2
```

> Iniciar o nó master:

```
python3 master.py
```

> Iniciar o cliente:

```
python3 client.py
```

> Informar o usuário no cliente:

```
username: Marcus
```

> Informar senha do usuário:

```
password: m123
```

> Escolher comando:

```
Selecione o comando[get, put, delete, exit]: put
```

> Informar o path do arquivo:

```
Informe o arquivo: ./files/fut.txt
```

> Nomear o arquivo:

```
Informe o nome do arquivo: fut

          #### Arquivo é registrado ####
```

> Inicia novamente o cliente:

```
python3 client.py
```

> Informar o usuário no cliente:

```
username: Marcus
```

> Informar senha do usuário:

```
password: m123
```

> Escolher comando:

```
Selecione o comando[get, put, delete, exit]: get
```

> Informa o nome do arquivo:

```
Informe o arquivo: fut

    #### Arquivo é printado na tela ####
```

> Inicia novamente o cliente:

```
python3 client.py
```

> Informar o usuário no cliente:

```
username: Marcus
```

> Informar senha do usuário:

```
password: m123
```

> Escolher comando:

```
Selecione o comando[get, put, delete, exit]: delete
```

> Informa o nome do arquivo:

```
Informe o arquivo: fut

    #### Arquivo deletado ####
```

> Para encerrar os minions e o master, basta realizar o comando Ctrl + C.

Vídeo da apresentação:
https://drive.google.com/file/d/1yhO2tc6xKUREs187V8MKdRHx70GOk81N/view?usp=sharing