U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

12 de abril de 2019

Sistemas Distribuídos

Relatório do Primeiro Projeto

(Mestrado Integrado em Engenharia Informática e Computação)

Grupo 7 da turma 4:

- Helena Montenegro up201604184@fe.up.pt
- Juliana Marques up201605568@fe.up.pt

I. Instruções de compilação / Execução

Para compilar, deve-se executar a instrução: ***javac *.java***. Para facilitar a compilação, incluímos o ficheiro ***"compile.sh"*** e ***"make.bat"*** que compilam o programa. Em Ubuntu, deve-se executar a instrução ***sh compile.sh***, e em Windows a instrução ***make.bat***. De seguida, inicia-se o registo RMI com a instrução: ***start rmiregistry***, em Windows, ou ***rmiregistry &***, em Ubuntu.

Para executar um *peer*, deve-se chamar a classe ***Peer*** com o seguinte formato:

```
java Peer <version> <peer_id> <remote_obj_name> <mc_addr>
<mc_port> <mdb_addr> <mdb_port> <mdr_addr> <mdr_port>
```

- ***version*** - "1.0" ou "2.0", conforme se queira executar a versão sem ou com melhorias do projeto, respetivamente;
- ***peer_id*** - refere-se ao identificador do *peer*;
- ***remote_obj_name*** - refere-se ao nome do objeto a ser usado pelo *peer* no registo RMI;
- ***mc_addr*** - endereço a ser usado pelo canal *multicast* de controlo;
- ***mc_port*** - *port* a ser usado pelo canal *multicast* de controlo;
- ***mdb_addr*** - endereço a ser usado pelo canal *multicast* do protocolo de ***"Backup";***
- ***mdb_port*** - *port* a ser usado pelo canal *multicast* do protocolo de ***"Backup";***
- ***mdr_addr*** - endereço a ser usado pelo canal *multicast* de controlo do protocolo de ***"Restore"***;
- ***mdr_port*** - *port* a ser usado pelo canal *multicast* de controlo do protocolo de ***"Restore"***.

Para executar cada um dos protocolos, deve-se chamar a classe ***TestApp*** do seguinte modo:

```
java TestApp <peer_ap> <sub_protocol> <opnd_1> <opnd_2>
```

- ***peer_ap*** - ponto de acesso ao *peer*, neste caso, este será a junção do *hostname* e do nome do objeto que se encontra no registo RMI, representativo do *peer* com o qual se quer comunicar (a junção é feita com os dois pontos ":"). Portanto, neste argumento coloca-se: < ***hostname*** > ***:*** < ***remote_obj_name*** >;
- ***sub_protocol*** - nome do protocolo que se quer iniciar (***BACKUP***, ***RESTORE***, ***DELETE***, ***RECLAIM***, ***STATE***, ***BACKUPENH***, ***RESTOREENH***, ***DELETEENH***);
- ***opnd_1*** - nome do ficheiro que se irá utilizar para correr os protocolos de ***"Backup"***, ***"Restore"*** ou ***"Delete"***, ou o espaço máximo reservado para guardar *chunks* no caso do protocolo de ***"Reclaim"***. No caso de ***"State"***, não existe este argumento;
- ***opnd_2*** - grau de replicação de um ficheiro para o qual se executa o protocolo de ***"Backup"***. Este argumento não existe para os restantes protocolos.

II. Concorrência

Para atingir a concorrência, foi utilizada a classe ***ConcurrentHashMap*** para representar a informação presente no ***Peer*** que poderá ser acedida por diversas *threads* ao mesmo tempo. Existem também várias *threads* com diferentes responsabilidades, que são geridas pelos *peers* utilizando a classe ***ScheduledThreadPoolExecutor***:

- ***Threads dos canais multicast: MCThread***, ***MRThread*** e ***MDBThread***, responsáveis por juntar o *peer* ao respetivo canal *multicast* e pela receção de mensagens. No canal de controlo (***MCThread***), esta é responsável também pelo envio de mensagens.
- ***Threads de receção de mensagens: ReceiveMessageMC***, ***ReceiveMessageMDR*** e ***ReceiveMessageMDB***, responsáveis pelas ações do *peer* em seguimento da receção de uma mensagem.
- ***Threads de envio de mensagens: MulticasterPutChunkThread*** e ***MulticasterChunkThread***, responsáveis pelo envio das mensagens de ***PUTCHUNK***, no caso do protocolo de ***"Backup"***, onde se deve repetir o envio da mensagem caso necessário (quando o número de ocorrências de um *chunk* é inferior ao seu grau de replicação desejado), e pelo envio das mensagens de ***CHUNK***, no caso do protocolo de ***"Restore".***
- ***Threads de execução de protocolos no initiator peer: DoReclaimThread*** e ***DoRestoreThread***, responsáveis pela execução do protocolo de ***"Reclaim"*** e do protocolo de ***"Restore"***, respetivamente, que foram separadas da classe do ***Peer*** devido à sua extensão.
- ***Outros Threads:***
  - ***ManageDataFilesThread***: responsável por escrever nos ficheiros que guardam informação sobre as ocorrências dos *chunks* e sobre os ficheiros para os quais o *peer* iniciou o protocolo de ***"Backup"***, sempre que ocorre alguma alteração a esta informação (ficheiros estes presentes na pasta *"data"*, dentro da pasta do *peer* respetivo).
  - ***SendDeleteThread***: responsável pelo envio da mensagem ***DELETE*** na melhoria do protocolo de ***"Delete"***, quando existem *peers* que contêm *chunks* do ficheiro recebido na mensagem e que ainda não a receberam.

III. Melhorias

Melhoria ao protocolo de "Backup"

A melhoria ao protocolo de ***"Backup"*** tem como objetivo diminuir o espaço em disco nos *peers*, reservado ao *backup* de ficheiros, tal como a sua atividade.

Para tal, quando os *peers* recebem a mensagem ***PUTCHUNK***, eles esperam um tempo aleatório entre 0 e 400 milissegundos antes de guardarem o *chunk* e enviarem a mensagem de ***STORED***. Sempre que um *peer* recebe uma mensagem ***STORED***, este guarda o identificador do *peer* responsável pela mensagem, tendo assim acesso a todos os *peers* que têm o *chunk* e podendo, deste modo, saber a sua ocorrência. Passado o tempo aleatório, os *peers* vão verificar se as ocorrências verificadas do *chunk* que querem guardar são superiores ou iguais ao grau de replicação do *chunk* e, se forem, o *peer* não guarda o *chunk*.

Melhoria ao protocolo de "Restore"

A melhoria ao protocolo de ***"Restore"*** tem como objetivo a substituição do uso de um canal *multicast* para receber um *chunk* cujo destino deveria ser só um *peer*, por uma conexão TCP.

Deste modo, ao enviar o pedido de um *chunk*, com a mensagem ***GETCHUNK*** modificada de modo a conter o endereço da máquina que corre o servidor e o *port* de serviço, o *initiator peer* abrirá uma conexão TCP usando a classe ***ServerSocket*** com o *port* recebido como argumento no início do *peer*, e esperará por uma resposta.

Um *peer* que receba a mensagem ***GETCHUNK*** e que contenha o *chunk* pedido iniciará uma conexão através da classe ***Socket***, usando o endereço IP e o *port* recebidos na mensagem. Para que apenas um dos *peers* envie o *chunk* ao *initiator peer* usando TCP, o *peer* que inicia a conexão envia uma mensagem para o canal de controlo do tipo ***SENTCHUNK***, informando os restantes *peers* que este se encarrega do envio do *chunk* pedido e que mais ninguém necessita de o enviar.

O formato da mensagem ***GETCHUNK*** modificada é:

```
GETCHUNK <Version> <SenderId> <FileId> <ChunkNo>
<ServiceAddress> <ServicePort> <CRLF><CRLF>
```

O formato da mensagem ***SENTCHUNK*** é:

```
SENTCHUNK <Version> <SenderId> <FileId> <ChunkNo>
<CRLF><CRLF>
```

Melhoria ao protocolo de "Delete"

A melhoria ao protocolo de ***"Delete"*** tem como objetivo certificar-se que um ficheiro será eliminado por todos os *peers*, inclusive aqueles que não estavam ativos quando foi solicitada a eliminação do ficheiro.

Para tal, sempre que o *initiator peer* envia esta mensagem, guarda o identificador do ficheiro a ser eliminado associado a uma lista que contém os identificadores dos *peers* que contêm *chunks* do ficheiro. Cada um dos *peers* que recebe a mensagem ***DELETE*** e elimina os *chunks* do ficheiro manda uma mensagem de ***DELETED***, que informa que o ficheiro foi eliminado com sucesso. O *initiator peer* recebe esta mensagem e elimina o *peer* que a enviou da lista de identificadores de *peers* que contêm *chunks* do ficheiro, eliminando também a entrada relativa ao ficheiro na estrutura de dados caso a lista fique vazia. Sempre que esta estrutura não estiver vazia, significa que existem *peers* que têm informação que não foi eliminada, pelo que o *initiator peer* irá enviar a mensagem de ***DELETE*** por cada um dos ficheiros existentes de 30 em 30 segundos.

O formato da mensagem ***DELETED*** é:

```
DELETED <Version> <SenderId> <FileId> <CRLF><CRLF>
```