SDIS Valorizações Trabalho 1

T6G05

Valorização 1: Subprotocolo de *Backup*

Para evitar que o grau de replicação de um determinado *chunk* de um ficheiro seja superior ao desejado, o que poderá conduzir a uma ocupação de espaço de disco desnecessária em algum ou mesmo vários pares, foi adicionada uma alteração que, mantendo a interoperabilidade entre pares, evita estes problemas.

A alteração efetuada foi a seguinte: quando um par recebe uma mensagem PUTCHUNK e possui espaço em disco suficiente, o par cria o respetivo *chunk* com os dados mas não procede de imediato ao seu armazenamento em disco. Após a espera do intervalo de tempo aleatório para poder responder, esse par verifica se o grau de replicação desse mesmo *chunk* já é igual ou superior ao desejado, se for descarta os dados desse *chunk* e não envia resposta alguma, se não for procede ao envio da mensagem STORED. A verificação referida torna-se possível porque o *chunk* é adicionado inicialmente pelo par (sem ser guardado em disco) passando assim a "seguir" o estado desse *chunk,* e porque em cada receção de uma mensagem do tipo STORED procede ao incremento do grau de replicação do mesmo *chunk* se o possuir também.

```
if(f.getChunk(chunkNo).getCurReplicationDeg() >= f.getReplicationDeg()) {
       f.removeChunk(chunkNo);
       (…)
(Extraído da função run da classe MDB.java)
```

Valorização 2: Subprotocolo de *Restore*

Quando os ficheiros são muito grandes o protocolo utilizado não é muito eficaz, uma vez que estamos a inundar a rede desnecessariamente pois apenas um par deseja o *chunk* pedido.

Para melhorar este processo, foi adicionada uma versão especial, a versão "2.2" na qual sempre que um par envia uma mensagem do tipo GETCHUNK o "primeiro" que possua a *chunk* pedida a formular a resposta liga-se por UDP ao par que enviou a mensagem, aumentando assim a rapidez do envio dos dados e evitando "inundar" a rede com mensagens que não são relevantes para nenhum outro par exceto o que faz o envio da mensagem inicial. As mensagens de resposta do tipo CHUNK são na mesma enviadas de acordo com o protocolo possuindo como versão a "2.2", no entanto o campo de dados vai vazio servindo apenas para informar os

outros pares que já há outro par a enviar a mensagem CHUNK, que irá possuir o campo de dados com a informação respetiva, por UDP.

```
if(!BackupService.getVersion().equals(BackupService.getVersionEnhancement())) {
	UDP udp = new UDP();
	udp.start();
}
(Extraído da função askRestoreFile da classe MC.java)
```

Valorização 3: Subprotocolo de *Delete*

Se um par não estiver a correr no momento em que outro par envia uma mensagem do tipo DELETE pode ficar com *chunks* guardados em disco que já não irão ser mais pedidos, resultando assim num uso ineficiente do disco.

Quando um par pretende apagar um ficheiro, envia a mensagem DELETE para os outros pares, quando outro par recebe a mensagem e possui algum *chunk* do ficheiro especificado responde com a mesma mensagem de volta (aumentando a probabilidade de todos os pares atualmente na rede receberem a mensagem DELETE). Este envio de resposta permite ao par que iniciou a ação de *delete* de um ficheiro registar quantas mensagens do tipo DELETE recebe em relação ao ficheiro e assim saber quantos pares apagaram efetivamente os *chunks* desse ficheiro. Como o par tem informação acerca de quantos pares e quais pares guardaram inicialmente o ficheiro, poderá assim saber se poderá dar por terminada a ação de *delete* se estes números forem iguais. Se não forem iguais, o par irá periodicamente voltar a enviar a mensagem DELETE até atingir um certo limite de envios.

O par que pediu inicialmente o DELETE pode não ter recebido todas as repostas DELETE, podendo por isso acontecer que vai voltar a mandar mensagens DELETE apesar de todos os pares já terem apagado o ficheiro. Esta foi no entanto uma escolha do grupo para garantir que todos os pares recebiam a mensagem DELETE ao longo do tempo.

```
if (!(f.getCountDeleted() >= f.getHighestReplicationDeg())) {
	BackupStatusHandler.addPendentMessage(msg);
}
(Extraído da função askDeleteFile da classe MC.java)
```

Valorização 4: Subprotocolo de Space Reclaiming

Se um par não conseguir guardar o seu ficheiro com o grau de replicação necessário no início depois nunca iria conseguir ter o grau de replicação desejado.

Para evitar este problema, foi criada uma *Thread* na qual é verificada periodicamente o estado de replicação de cada *chunk* de um ficheiro, se for menor que o desejado tenta fazer *backup* novamente dessa *chunk* a fim de conseguir o estado de replicação desejado.

Por outro lado, as mensagens do tipo REMOVED são, tal como as mensagens do tipo de DELETE, enviadas periodicamente assumindo que desta forma todos a vão receber.

Todas as valorizações são compatíveis com a versão "1.0" do protocolo à excepção da valorização 2. A valorização 2 tem de ser ativada na interface, as restantes uma vez que são compatíveis com o protocolo "1.0" estão sempre a funcionar.