Mestrado integrado em Engenharia Informática e Computação – Sistemas Distribuídos

Ana Amaral, up201303169@fe.up.pt

Pedro Câmara, up201304073@fe.up.pt

Distributed Backup Service - Melhorias

Melhoria do Subprotocolo *Chunk Backup*

Quando uma mensagem do tipo ***PUTCHUNK*** é recebida, o subprotocolo "ouve" os ***STOREDs*** e só guarda o *chunk* se não tiver sido atingido o nível de replicação desejado.

Implementação contida no ficheiro "PutChunkHandler.java":

SE replicação_atual < replicação_desejada **ENTÃO**

Guarda o *chunk*

FIM SE

Vantagens:

- Não requer quaisquer mensagens adicionais, mantendo assim a interoperabilidade

- Evita atividade nos *peers* desnecessária, pois não guarda cópias desnecessárias dos *chunks,* o que faz com que não seja necessário fazer *space reclaiming* tão cedo

Desvantagens:

- Não garante que guarde apenas o mínimo

Melhoria do Protocolo *Chunk Restore*

Os *chunks* são enviados por UDP *unicast* e só são enviados com a probabilidade de 1/número de replicações. Se receber um segundo pedido para esse mesmo *chunk* no espaço de 1 minuto, então envia por *multicast*.

Implementação contida no ficheiro "GetChunkHandler.java":

SE já recebeu pedido para mesmo *chunk* no espaço de 1 minuto **ENTÃO**

Envia *chunk* por *multicast*

SENÃO

N = número aleatório [0, replicação[

SE N = 0 **ENTÃO**

Envia *chunk* por *unicast*

FIM SE

FIM SE

Vantagens:

- Não requer quaisquer mensagens adicionais, mantendo assim a interoperabilidade

- Por enviar os *chunks* por *unicast*, evita processamento desnecessário nos outros *peers*

Desvantagens:

- Visto que funciona por estatísticas, pode acontecer que não seja enviado por nenhum *peer* e que seja enviado só depois por *multicast*

Melhoria do Subprotocolo *File Deletion*

O dono dos ficheiros apagados sempre que "ouve" um ***REMOVED*** ou ***PUTCHUNK*** de um *chunk* desses ficheiros, envia ***DELETE***.

Implementação contida no ficheiro "PutChunkHandler.java" e "RemovedHandler.java":

Na receção de um *REMOVED* ou *PUTCHUNK*:

SE fileID contido na lista de ficheiros apagados **ENTÃO**

Inicia subprotocolo de *DELETE*

Ignora mensagem

FIM SE

Vantagens:

- Não requer quaisquer mensagens adicionais, mantendo assim a interoperabilidade

- Reduz o número de ficheiros "*zombies"*

Desvantagens:

- Obriga à utilização de espaço para guardar a lista dos IDs dos ficheiros apagados

- Só elimina os ficheiros "*zombies*" aquando do *space reclaiming* por parte do *peers* com esses ficheiros

Melhoria do Subprotocolo *Space Reclaiming*

O *initiator* reinicia a contagem de replicação, envia o ***REMOVED*** e espera 1 segundo. Se a contagem de replicação não tiver atingido o mínimo desejado executa o protocolo de backup desse *chunk* para tentar garantir a replicação. Se mesmo assim não conseguir, envia um ***STORED*** para atualizar a replicação nos outros *peers*. Só apaga os *chunks* se conseguir garantir o nível de replicação desejado.

Implementação contida no ficheiro SpaceReclaiming.java:

Espaço_ocupado = Cálculo do espaço ocupado

Obtida lista dos *chunks* ordenada por sobre-replicação estimada

PARA CADA *chunk* dessa lista até Espaço_ocupado <= Espaço_desejado

Reinicia contagem de replicação

Envia *REMOVED*

Espera 1 segundo

SE nível de replicação atual < nível de replicação mínimo **ENTÃO**

Executa subprotocolo de backup para o *chunk*

FIM SE

SE nível de replicação atual >= nível de replicação mínimo **ENTÃO**

Espaço_ocupado = Espaço_ocupado – tamanho do *chunk*

Apaga *chunk*

SENÃO

Envia STORED

FIM SE

FIM PARA

Vantagens:

- Não requer quaisquer mensagens adicionais, mantendo assim a interoperabilidade

- Garante que um ficheiro nunca é apagado sem colocar o nível de replicação abaixo do mínimo

Desvantagens:

- Pode não ser possível fazer o *space reclaiming* devido a baixos níveis de replicação e ausência de *peers* que os garantam

- Processo mais lento do que simplesmente apagar e enviar *REMOVEDs*