**Relatório SDIS - *Enhacements***

***Sistemas Distribuídos***

***Mestrado Integrado em Engenharia Informática e Computação***


João Silva - up201305892@fe.up.pt
Pedro Castro - up201305337@fe.up.pt

## 3.2 Chunk Backup Subprotocol

Segundo a implementação da versão *default* do protocolo, a mensagem de PUTCHUNK deve ser processada por todos os peers que receberem a mensagem. Por causa desta implementação a *replication degree* poderá ser bastante superior à que é desejada, o que não é necessariamente um aspecto negativo mas pode levar a um exagerado número adicional de backups.

A forma de reduzir o numero de backups adicionais, mantendo sempre a *replication degree* igual ou acima do desejado foi implementada da seguinte maneira:

Um peer ao receber um PUTCHUNK, em vez de gravar logo como no protocolo original, vai esperar um tempo aleatório entre 0 e 400 ms. Durante este tempo vai contar quantas mensagens de STORED do dado chunk recebeu.
Se o este numero for igual ou superior ao minimo de *replication degree*, vai descartar o chunk e não o vai guardar porque significa que já foi gravado no mínimo o número de vezes necessárias. Se for inferior, vai guardar o chunk, enviar a mensagem de STORED e guardar o chunk.

## 3.3 Chunk Restore Subprotocol

No protocolo original, um peer que recebe um GETCHUNK e possui o dado chunk, espera um tempo aleatório entre 0 e 400 ms e ouve se recebe alguma mensagem CHUNK desse chunk. Se receber uma desta ultima, significa que alguém já enviou, não envia a mensagem, se não receber envia.

No entanto, este modo implica encher o multicast channel de packets com grande dimensão quando apenas um peer o deseja receber.

Resolvemos este problema da seguinte maneira:

Quando iniciamos o peer, para além dos canais multicast também criamos um canal TCP numa porta que esteja disponível no sistema.

Ao receber uma mensagem GETCHUNK, o peer vai funcionar parcialmente como no outro protocolo. Espera entre 0 e 400 ms e escuta por mensagens CHUNK para o mesmo chunk. Se após isto tiver recebido uma mensagem, não envia o chunk. Se não tiver recebido faz o seguinte:

1. Envia a mensagem original de CHUNK (isto é, a mensagem com o *body*) por TCP para o initiator-peer.
2. Envia uma mensagem CHUNK sem o body para o canal multicast para que os outros peers não enviem também o CHUNK. Desta forma não irá ser desperdiçado uma quantidade de dados enorme que o seria se o canal multicast fosse utilizado para este efeito.

Para o peer saber a porta para onde enviar o CHUNK com o body, o initiator-peer anexa como último argumento a porta a mensagem do GETCHUNK fica então com este formato “GETCHUNK <VERSION> <SERVERID> <FILEID> <CHUNKNO> <PORT> <CRLF> <CRLF>”

## 3.4 File Deletion Subprotocol

O Enhacement sugere arranjar uma forma de reduzir a quantidade de ficheiros “zombie” (ficheiros que foram apagados enquanto o peer estava desligado, portanto não recebeu a mensagem de DELETE).

O nossa solução a este problema consiste no seguinte:

Quando um peer recebe a mensagem de DELETE, envia uma nova mensagem, DELETECHUNK por cada chunk que enviou.

O peer que lancou o DELETE conta os DELETECHUNK que recebeu. Quando este receber mensagens DELETECHUNK de numero igual ao numero de *replication degree*, para a sua execução.
Se isto não acontecer, espera um determinado tempo(que vai sendo incrementado ao longo das várias iterações) e envia novamente o DELETE.

Deste modo teremos uma thread que tentará sempre remover estes ficheiros zombies, até receber confirmação que assim foi feito.

## 3.4 Space Reclaiming Subprotocol

Este Enhacement pede que se garanta que caso o BackupProtocol falhe no peer iniciador, arranjar uma forma de manter a *replication degree* minima. (“the peer that initiates the chunk backup subprotocol fails before finishing it”).

Nós decidimos resolver isto, passando a responsabilidade de manter a *replication degree* aos peers que já receberam os chunks em causa.

Na nossa implementação, passados 10 segundos depois de ter recebido o PUTCHUNK e se ter gravado o ficheiro, se a *replication degree* deste dado não for satisfatória, este peer, não o iniciador, vai se tornar responsavel por mandar novos PUTCHUNKS garantindo assim que é mantida a *replication degree minima.*

Se entretanto o ficheiro for apagado, os novos PUTCHUNKS não vão ser enviados.