SDIS 2015/2016 – 2º Semestre
Projeto 1 – Serviço de Backup Distribuído

António Manuel Vieira Ramadas – up201303568
Rui Miguel Teixeira Vilares – up201207046

Turma 5 – Grupo 03
Sistemas distribuídos

Melhoramentos

Chunk backup subprotocol

Segundo a especificação do projeto, quando uma mensagem **PUTCHUNK** é recebida, guarda-se o *chunk*, espera-se um intervalo de tempo aleatório (entre 0 e 400 milissegundos) e envia-se uma mensagem **STORED**. É necessário evitar que o espaço destinado ao backup de ficheiros seja rapidamente atingido, devido à elevada replicação de *chunks* (grau de replicação maior que o desejado). A nossa melhoria não requer nenhuma mensagem especial e é compatível com a implementação pretendida do protocolo.

Assim, no final do tempo de espera aleatório (entre 0 e 400 milissegundos), confirma-se o grau de replicação atual. Caso esse grau de replicação seja maior ou igual ao número de replicação desejado, não há necessidade de guardar esse *chunk*. Caso contrário, ou seja, caso o número de replicação desejado não tenha sido ainda atingido, procede-se normalmente, guardando o *chunk*.

Esta melhoria permite que o espaço destinado ao backup não seja rapidamente esgotado. Além disso, procura atingir sempre o grau de replicação desejado.

Chunk restore subprotocol

Para recuperar um determinado *chunk*, é enviada uma mensagem **GETCHUNK** para o MC (*multicast channel*), com a especificação do *chunk*. Quando essa mensagem é recebida por um *peer* que contém uma cópia desse *chunk*, ele é enviado pelo *body* da mensagem **CHUNK**, através do canal MDR (*multicast channel restore*). Caso o *chunk* seja muito grande ou sejam envidados vários *chunks* para a rede, este protocolo não é desejável. Além disso, apenas um *peer* necessita de receber o *chunk* e é usado um canal *multicast* para isso.

Neste caso, para eliminar o problema e melhorar a solução, altera-se o protocolo e interage-se com os restantes *peers*. Assim, ao enviar a mensagem **CHUNK** para a rede, ela vai sem o body, permitindo assim aos *peers* tomar conhecimento. Em seguida, a mensagem **CHUNK** com o *body* é enviada diretamente para o *peer* que enviou a mensagem **GETCHUNK**. Ao mandar uma mensagem para a rede, cuja versão é superior à do protocolo normal, o id do emissor é substituído pela porta multicast desse mesmo emissor. O emissor do pedido, no início da comunicação, cria um novo canal *multicast* sem argumentos. Esse canal é usado para receber o *chunk*, sem recurso aos três canais *multicast* habituais.

Com esta solução, evita-se enviar o *chunk* com *body* a *peers* desnecessários e aproveitam-se os benefícios do canal *multicast*. Este subprotocolo é o único que modifica a especificação proposta inicialmente.

File deletion subprotocol

Quando um ficheiro é eliminado, os seus *chunks* devem também ser eliminados do serviço de backup. Quando um *peer* recebe a mensagem **DELETE**, deve remover todos os *chunk* pertencentes ao ficheiro especificado. Neste caso, não é esperada nenhuma mensagem de resposta. Pode-se enviar a mensagem o número de vezes que se quiser, de modo a garantir que todo o espaço que estava a ser ocupado pelos *chunks* seja libertado, independentemente da perda de algumas mensagens. O problema é que, quando se enviam essas mensagens, podem existir *peers* que não estão a ser executados e como tal, o espaço usado pelos *chunks* nesses *peers* nunca será reclamado.

Aqui, recorrendo a uma pequena alteração do protocolo, incluindo mensagens adicionais, podemos resolver esse problema. De cada vez que o *peer* se liga à rede, manda remover todos os ficheiros. Caso não recebe nenhuma mensagem **PUTCHUNK**, significa que é seguro apagar. Pelo contrário, se receber, responde à mensagem e não apaga.

Este melhoramento garante que quando um ficheiro é apagado, todos os seus *chunks* serão removidos, mesmo que algum *peer* não esteja em execução naquele momento.

Space reclaiming subprotocol

Na especificação do projeto, quando uma mensagem **REMOVED** é recebida por um *peer* que contem uma cópia local do *chunk* especificado, atualiza-se a contagem local desse *chunk*. Se essa contagem for menor que o nível de replicação desejado por esse *chunk*, o subprotocolo de backup inicia-se após a espera de um tempo aleatório (varia entre 0 e 400 milissegundos). Se durante essa espera, o *peer* receber uma mensagem **PUTCHUNK** para o mesmo *chunk*, deve voltar atrás e evitar iniciar outro subprotocolo. Este protocolo pode ter um problema, se o subprotocolo de backup falhar, o grau de replicação pode ser menor que o pretendido.

Uma melhoria que foi feita, de forma eficiente e compatível com a especificação proposta de todos os protocolos tem em conta as seguintes considerações: ao receber a mensagem **REMOVED**, o grau de replicação é decrementado, ficando abaixo do desejado; em seguida, é iniciado um protocolo backup, executado por outro qualquer *peer*; faz-se uma espera aleatória (variável entre 30 e 60 segundos); volta-se a verificar o grau de replicação e caso seja menor que o desejado, inicia-se o subprotocolo de backup.

Esta solução, compensa o facto de um *peer* falhar o subprotocolo de backup, porque espera um tempo significativamente grande. Além disto, o tempo de espera aleatório (variável entre 30 e 60 segundos) garante uma reduzida probabilidade de coincidência na iniciação da recuperação de dois *peers*.