Universidade do Porto

Faculdade de Engenharia

FEUP

Serverless Distributed Backup Service

Sistemas Distribuídos

Mestrado Integrado em Engenharia Informática e Computação

Elementos do Grupo:

Catarina Ramos - up201406219 - up201406219@fe.up.pt
Inês Gomes - up201405778 - up201405778@fe.up.pt

10 de Abril de 2017

1. Enhancement do Chunk Backup Protocol

"This scheme can deplete the backup space rather rapidly, and cause too much activity on the nodes once that space is full. Can you think of an alternative scheme that ensures the desired replication degree, avoids these problems, and, nevertheless, can interoperate with peers that execute the chunk backup protocol described above?"

Para a implementação deste enhancement seguimos o seguinte raciocínio: um peer após ter recebido um pedido de *PUTCHUNK* para um determinado chunk e tendo todas as condições necessárias para o guardar, só o irá fazer se entretanto esse chunk tiver sido guardado um número de vezes inferior ao número da replicação desejada.

Sendo assim, para obtermos o número de vezes que o chunk foi guardado antes de o peer processar o pedido, temos o auxílio da classe *MessageRecord* associada ao peer. Esta classe regista determinadas mensagens que vão sendo recebidas pelos canais multicast. Neste caso, seria do nosso interesse obter o registo das mensagens do tipo *STORED* para aquele chunk associado a um determinado ficheiro. Este registo permite-nos obter a lista de peers que enviaram este tipo de mensagem. Tendo agora o número de peers, é só verificar se o número é maior ou igual ao número de replicação desejado. No fim, faz se um reset ao registo das mensagens *STORED* do *MessageRecord* para mais tarde numa outra tentativa de backup ser atualizado a lista de peers com o chunk.

Os peers que já tinham o chunk em backup enviam imediatamente a mensagem de *STORED* para o canal multicast. Desta forma, garante que quando um peer que não tenha o chunk esteja a ver o registo daquele chunk, consiga obter uma lista de peers o mais atualizada possível. Caso contrário, se o tempo de espera do peer for menor do que o de outro peer que já tenha o chunk, a mensagem de *STORED* chegaria após esse tempo e a verificação no registo não estaria de acordo com o número de replicação correto.

O enhancement do "Space Reclaiming Subprotocol" também contribui para garantir um nível de replicação dos chunks igual ao desejado, pelo menos, em caso de falha deste protocolo.

2. Enhancement do Chunk Restore Protocol

"If chunks are large, this protocol may not be desirable: only one peer needs to receive the chunk, but we are using a multicast channel for sending the chunk. Can you think of a change to the protocol that would eliminate this problem, and yet interoperate with non-initiator peers that implement the protocol described in this section?"

Para a implementação deste enhancement seguimos o seguinte raciocínio: se apenas um peer está interessado em receber o chunk, então podemos enviá-lo através de um canal "privado" entre o peer que tem o chunk e o peer que fará o restore. Mais tarde, a confirmação de recepção do chunk seria feita pelo canal multicast.

De modo a segurar a eficiente implementação deste enhancement, foram criados dois novos tipos de mensagens: *GETCHUNKENH* e *GOTCHUNKENH*, com as respetivas estruturas:

- GETCHUNKENH <version> <senderid> <fileid> <chunkno> <address> <port> <CRLF><CRLF>
- GOTCHUNKENH <version> <senderid> <fileid> <chunkno> <CRLF><CRLF>

A mensagem *GETCHUNKENH* é a mensagem que substitui a *GETCHUNK* e contém adicionalmente o endereço e a porta de quem a envia. Esta informação será usada para criar um canal de comunicação privado entre os dois peers intervenientes, aquele que faz o pedido e aquele que responde. Este canal privado é instanciado pela classe *DatagramListener* que, assim como a thread *MulticastListener*, processa num ciclo infinito a recepção de mensagens e contém ainda métodos para o envio de mensagens. Sendo assim, após o envio de uma mensagem *GETCHUNKENH* o peer cria uma instância da classe *DatagramListener* e executa-a. Por outro lado, após a recepção deste tipo de mensagens, o peer cria também uma instância desta classe e executa-a, caso tenha o chunk que foi pedido e após um tempo aleatório, se não houve a confirmação de envio do chunk (pelo registo do *MessageRecord*), envia a mensagem do tipo *CHUNK* pelo canal privado.

A mensagem *GOTCHUNKENH* serve como mensagem de confirmação da recepção do chunk por parte do peer que efetuou o pedido. Isto é necessário porque, como a informação de envio do chunk é feita pelo canal privado e não pelo multicast, os outros peers não conseguem detectar a transmissão desta informação, o que levaria a enviarem, desnecessariamente, os chunks pelo canal privado, podendo "entupir" o peer que espera a recepção. Esta mensagem quando é recebida por um peer apenas é registada no *MessageRecord* como se fosse uma mensagem do tipo *CHUNK*.

3. Enhancement do File Deletion Subprotocol

"If a peer that backs up some chunks of the file is not running at the time the initiator peer sends a DELETE message for that file, the space used by these chunks will never be reclaimed. Can you think of a change to the protocol, possibly including additional messages, that would allow to reclaim storage space even in that event?"

De forma a implementar este enhancement, decidimos implementar duas novas mensagens, que serão transmitidas sempre que o peer iniciar.

O objetivo deste enhancement passa por enviar uma mensagem do tipo *GETINITIATOR*, para o canal multicast, por cada fileId diferente dentro do seu diretório de chunks guardados. Cada peer, ao receber este tipo de mensagem verifica se, no seu mapeamento dos ficheiros cujo backup foi iniciado, existe algum ficheiro com o fileId igual ao recebido na mensagem. Se sim, o peer deve então responder com uma mensagem do tipo *INITIATOR* pelo canal multicast.

O peer que enviou a mensagem *GETINITIATOR* fica à espera da mensagem *INITIATOR*. Se receber, significa que não foi enviada nenhuma mensagem do tipo *DELETE* enquanto o peer não estava ativo. Se após o tempo de espera não tiver recebido a mensagem do tipo *INITIATOR* significa que já não existe o backup desse ficheiro, e os seus chunks devem ser eliminados.

Estes dois novos tipos de mensagens têm a seguinte estrutura:

- *GETINITIATOR* <version> <senderid> <fileid> <CRLF><CRLF>
- *INITIATOR* <version> <senderid> <fileid> <CRLF><CRLF>

4. Enhancement do Space Reclaiming Subprotocol

"If the peer that initiates the chunk backup subprotocol fails before finishing it, the replication degree of the file chunk may be lower than that desired. Can you think of a change to the protocol, compatible with the chunk backup subprotocol, that could tolerate this fault in an efficient way? Try to come up with a solution that works in both cases of execution of the chunk backup subprotocol, i.e. both when a chunk is being backed up for the first time and when a copy of the chunk is deleted."

De forma a implementar uma solução viável para o tipo de problemas descritos acima, optamos por criar uma tarefa agendada de tempo em tempo no peer para a verificação de chunks com um nível de replicação abaixo do desejado. Para cada um destes chunks, nestas condições, é iniciado o *ChunkBackupProtocol* para aquele chunk se, após um tempo aleatório, não tiverem recebido mensagens do tipo *PUTCHUNK* para aquele chunk em específico. Se o *ChunkBackupProtocol* for iniciado, o peer também manda uma mensagem de *STORED* para o multicast a avisar que ele próprio também tem o chunk em backup.

Esta solução permite tentar repor o nível de replicação desejada de um chunk, que por alguma razão está em défice. Este défice pode se dever a uma falha no backup de um ficheiro, após o reclaim num peer ou até mesmo quando, atualmente, no canal multicast, o número de peers é insuficiente para a replicação desejada do chunk.