Sistemas Distribuídos

Relatório do Primeiro Projeto

Grupo T1G09:
Bernardo Belchior
Maria João Mira Paulo

Introdução

Além dos protocolos base do projeto, o grupo implementou ainda melhorias ao nível dos protocolos de *Backup*, *Restore*, *Delete* e *Reclaim Space*.

Protocolo de *Backup*

O protocolo de ***Backup*** não garante a replicação de um *chunk* com *replication degree* desejado, o que pode ser responsável pela ocupação de demasiado espaço de disco, desnecessariamente. Isto acontece porque os recetores da mensagem *PUTCHUNK* guardam o seu conteúdo em disco e esperam um período aleatório entre 0 e 400 ms para enviar a confirmação.

A melhoria deste protocolo foi implementada invertendo a ordem de operações, ou seja, quando é recebida uma mensagem *PUTCHUNK*, o *peer* espera entre 0 e 400 ms para começar a escrever o *chunk* para o disco. Contudo, antes de do fazer, ele verifica que o grau de replicação atual é maior ou igual ao desejado e, nesse caso, aborta a escrita do conteúdo do *chunk*. Caso o *replication degree* seja menor, o peer escreverá em memória não-volátil, enviando a mensagem STORED imediatamente após finalizar este processo.

Com vista a melhorar a eficiência do protocolo, o grupo utiliza o bloqueio da estrutura de dados que guarda o *replication degree* atual de cada *chunk* no processamento de cada mensagem *PUTCHUNK* e no processamento de cada mensagem *STORED*.

Protocolo de *Restore*

O protocolo de ***Restore*** pode tornar-se ineficiente quando se tratam de grandes *chunks*, com grande número de *bytes*, uma vez que embora apenas um *peer* esteja à espera de receber esse *chunk*, como a mensagem *PUTCHUNK* é enviada por *multicast*, todos os *peers* que se encontrarem ligados à rede irão receber essa mensagem, desnecessariamente.

Desta maneira, um *peer* com protocolo 1.1 pode estabelecer uma conexão TCP directamente com o *peer* que pediu esse *chunk*, desde que este também possua o mesmo protocolo. Assim, o *peer* que inicia o *Restore*, ao possuir o protocolo 1.1, além de preparado para receber qualquer mensagem que possa ser recebida no *Recovery Channel*, cria também um *Serversocket*

responsável por aceitar pedidos que possam ser enviados pela rede. O *peer* que recebe a mensagem *PUTCHUNK,* caso não possua esse *chunk* na sua própria rede, descarta. Caso contrário, se possuir o protocolo melhorado, a mensagem de *CHUNK* é enviada mas desta vez através de um socket diretamente para o *peer* que enviou a mensagem.

Desta maneira, apenas o *peer* que pediu o *chunk* irá receber a mensagem, o que torna este protocolo bastante mais eficaz.

Para a criação do *socket* foi usada a porta do *multicast channel*, uma vez que é possível coexistirem protocolos TCP e UDP na mesma porta. O endereço IP usado pelo peer que envia a mensagem *PUTCHUNK* é o IP do peer remoto, conseguido através da função *getAddress* do *DatagramPacket* recebido.

Protocolo de *Delete*

O protocolo de ***Delete*** foi melhorado no sentido de prevenir que um um peer guarde na sua rede *chunks* de um ficheiro já apagado. Isto é, se um *peer* que fez *backup* de certos *chunks* de um ficheiro não está ativo no momento em que o ficheiro é apagado e são enviadas as mensagens de *DELETE* nunca se vai aperceber que está a ocupar espaço desnecessário na sua própria rede.

Desta forma, a melhoria implementada baseou-se no uso do mecanismo de *lease.* Este processo funciona oferecendo uma "licença de uso" a cada *peer* para cada ficheiro existente na rede. Esta licença tem uma duração de estabelecida e, quando esta acaba, o *peer* necessita de a renovar. Isto é feito através da mensagem *GETLEASE*. Se o remetente receber uma mensagem *LEASEOK*, a licença do ficheiro será renovada e poderá ser mantido. Caso a resposta ultrapasse um *timeout* pré-estabelecido, o remetente assume que o ficheiro foi apagado da rede e removê-lo-á. Só podem enviar uma mensagem *LEASEOK* os *peers* que guardam um *chunk* do ficheiro pedido.

Protocolo de *Reclaim Space*

O protocolo de ***Reclaim Space*** não está preparado para ser interrompido. Por exemplo, um *peer* que tenha iniciado *backup* de um ficheiro, se, por alguma razão, for forçado a parar, provocará um erro e o ficheiro não será guardado com sucesso. Para resolver este problema, foi implementada uma solução para permitir que um *peer* que inicie um *backup*, seja capaz de completar tarefas que possam ter ficado incompletas.

Peers com protocolo 1.1 possuem uma estrutura de dados, denominada *incompleteTasks*, capaz de armazenar tarefas incompletas, ou seja, *chunks* que estão à espera para serem guardados. Sempre que é iniciado o protocolo de *BACKUP de um* ficheiro*,* todos os *chunks* pertencentes a este são adicionados às *incompleteTasks* e, sempre que é recebida a mensagem *STORED* o *chunk* é retirado desta. Um *peer*, ao ser iniciado é responsável por completar, uma a uma, as tarefas que possam ter ficado incompletas.

Da mesma forma, sempre que é recebida uma mensagem *REMOVED, um peer capaz* de iniciar backup deste, antes de enviar a mensagem de *PUTCHUNK*, adiciona-o às *incompleteTasks.* Assim, é garantido que mesmo que esse *peer* seja forçado a terminar, quando é iniciado, voltará a enviar as mensagens de *PUTCHUNK* e, desta maneira, garante-se que o *replication degree* do ficheiro sejam sempre igual ou maior que o desejado.