Serviço de Backup Distribuído

Sistemas Distribuídos

[Relatório do primeiro projeto]

Grupo T1G02

Francisca Leão Cerquinho R. Fonseca – up201505791

Ventura de Sousa Pereira - up201404690

Melhoramentos

Além dos protocolos base do projeto, o grupo implementou ainda melhorias a nível do protocolo Backup.

Protocolo Backup

O protocolo Backup descrito no enunciado não garante replicação de um *chunk* com *replication degree* desejado, ocupando demasiado espaço de disco. Isto acontece porque os recetores da mensagem ***PUTCHUNK*** guardam o seu conteúdo em disco e esperam um tempo aleatório entre 0 e 400 ms para enviar a confirmação (***STORE*** *message*).

Com vista a melhorar este protocolo usamos uma "**ConcurrentHashMap<chunkname, contador>"**, sendo *chunkname* o nome de um *chunk* e *contador* o número de vezes que esse chunk foi guardado(*STORED)*. ConcurrentHashMap é uma tabela de hash mais rápida, que consome menos memória e permite associar *valores* a *chaves* facilmente, assim como *thread's*. É usada para associar o nome de cada *chunk* ao seu número de *chunks* guardados. Assim, na receção do *PUTCHUNK,* o valor do *replication degree* é comparado com esse valor. No caso de o grau de replicação ser maior ou igual a este valor, a escrita do conteúdo de um *chunk* é interrompida, caso contrário o *peer* envia a mensagem *STORED* após finalizar o processo.

Implementação da concorrência

Para aumentar a eficiência e a concorrência do programa, utilizamos *thread's*, que são pequenos processos que compartilham os mesmos recursos e endereçamentos de memória.

Cada mensagem lança uma *thread* de execução. Utilizamos *thread's* diferentes para processar as mensagens:

-Executors.newFixedThreadPool, cria um *pool* com número fixo de *thread's* e fila ilimitada de tarefas, evitando um novo encadeamento para processar cada mensagem.

-Executors.newScheduledThreadPool é um *ExecutorService* que permite agendar tarefas para executar após um atraso, ou para executar repetidamente com um intervalo fixo de tempo entre cada execução. As tarefas são executadas de forma assíncrona por um *thread* de trabalho e não pelo *thread* que entrega a tarefa ao *ScheduledExecutorService*.

Todos os subprotocolos usam um canal multicast, o canal de controle (**MC**). Alguns subprotocolos também usam um dos dois canais de dados multicast, **MDB** e **MDR**, que são usados para fazer backup e restaurar dados de fragmentos de arquivos, respetivamente. O canal de controle(**MC**), processa cada tipo de mensagem recebida pelo canal.