

## 

## **Melhoramentos ao Serviço de Backup Distribuído**

Relatório

Sistemas Distribuídos

3º ano do Mestrado Integrado em Engenharia Informática e Computação

# 

Elementos do Grupo:

Andreia Rodrigues – up201404691 – up201404691@fe.up.pt

Eduardo Leite – gei12068 – gei12068@fe.up.pt

Francisco Queirós – up201404326 – up201404326 @fe.up.pt

Fábio Caramelo – up201404783 – up201404783 @fe.up.pt

21/05/2017

**Introdução**

O nosso trabalho foi a continuação do primeiro projecto desenvolvido (“Distributed Backup Service”) com a versão de internet e implementa todos os protocolos e respectivos enhancements devidamente adaptados.

**Master Servers**

Como já não era possível utilizar o Multicast para as comunicações, surgiu a necessidade de introduzir o conceito de Master Server. Existe um conjunto de Master Servers cujo endereço é conhecido pelos Peers, que ao iniciarem se ligam a eles por TCP usando SSLSockets. Após estabelecida a ligação, o Peer envia para o servidor todas as ports onde está a ouvir, o seu ID e a porta que usa para enviar. Assim quando um Peer quer enviar uma mensagem a todos os outros Peers, pede uma “lista de contactos” ao Master Server a que está ligado.

**Segurança**

Um Peer precisa de se ligar e autenticar via SSL (os peers têm uma keystore, os Master Servers outra e partilham uma truststore) para obter a lista de contactos, desta forma só Peers autenticados têm acesso a este serviço. O grupo considerou após consultar o professor que verificar se as mensagens UDP vêm de um endereço de envio conhecido era uma camada extra desnecessária, contudo uma estrutura para implementar essa funcionalidade existe caso seja necessária no futuro.

Também é possível requerer confidencialidade nos ficheiros, isto foi conseguido com um novo argumento booleano no protocolo de backup, no caso de ser requerida confidencialidade, o ficheiro é primeiro lido, depois encriptado e só aí partido em chunks (para não interferir com o tamanho dos chunks). Os outros Peers guardarão estes chunks encriptados da mesma forma que os guardariam de outra forma, o initiator é o único a manter registo deste facto, efetuando a desencriptação no caso de executar um restore.

**Tolerância a falhas**

No caso do backup falhar a meio: ao iniciar este protocolo a primeira coisa que acontece é um registo do backup, assim sendo, implementamos um thread que está sempre a correr e a verificar de x em x tempo o replication degree de todos os ficheiros que estão backed up, corrigindo backups inacabados por qualquer motivo.

No caso do restore falhar a meio: se isto acontecer enquanto recebe os chunks, simplesmente não é efetuado o restore. Se durante a operação de escrita o Peer fôr abaixo, o ficheiro ficará com o nome de Temp, nome que é dado ao ficheiro até ao momento em que a escrita acaba. Ao iniciar o Peer, todos os ficheiros Temp são apagados, o restore tem de ser repetido (este facto é incontornável pois os Chunks em memória volátil são perdidos).

No caso do delete falhar a meio: todos os deletes são guardados num ficheiro de deletes logo que são iniciados, sendo de lá retirados apenas quando é efectuado um novo backup dos mesmos (identificados pelo fileID não pelo nome, para garantir que temos um registo da versão correcta), no caso de um delete falhar a meio ou um Peer estar desligado quando o delete acontece, existe um método que todos os Peer executam ao iniciar que pergunta a todos os outros Peers que ficheiros foram deletados e verifica se ainda está a guardar algum. O único downside desta solução é decidir durante quanto tempo guardar os registos.

No caso do reclaim falhar: como o reclaim é feito passo a passo, ele pode apenas não ter feito reclaim de tanto espaço quanto o utilizador queria, neste caso deve ser efectuado um novo reclaim do espaço que ainda devia ser reclaimed, sem qualquer prejuízo para o sistema.

No caso de um Peer perder toda a informação: periodicamente os Peers enviam os seus metadados para os Master Servers, se um Peer iniciar e vir que não tem metadados, vai perguntar aos Master Servers se têm algo guardado, este método não é perfeito mas evita que os Peers tenham de começar do zero perdendo todos os seus dados. Não cobre casos extremos como os Master Servers perderem estes dados no exacto momento em que o Peer também os perde, mas consideramos ser uma solução satisfatória no âmbito do projecto.

Para suportar o facto de um Master Server ir abaixo, existem vários Master Servers, quando um Peer perde a conexão com o seu Master Server irá procurar outros para se ligar, tal acontece sem qualquer prejuízo para o sistema.

**Escalabilidade**

O problema de escalabilidade do programa vinha do facto de um Master Server ter de manter múltiplas conexões, foi para resolver esse problema que se decidiu ter vários, este facto também ajudou com a tolerância a falhas. Claro que no caso de estarem todos em baixo o programa não funciona, mas é mais difícil que tal aconteça. Quando um Master Server vai abaixo, outros ficam com as conexões dele automaticamente e ao ligar-se. O Peer utiliza a função PeerID % N para determinar a que Master Server se vai ligar, para tentar distribuir as ligações de forma igual por todos os Master Servers.

**Notas**

Os endereços dos Master Server devem ser conhecidos logo à partida pelos Peers e entre eles, para efeitos de teste deixamos o sistema com 3 Master Servers, nas portas 5000, 5001 e 5002 configurados para estarem todos na mesma máquina (assim ligam-se uns aos outros usando localhost, sem necessidade de andar a passar os endereços uns dos outros). Eles funcionam da mesma forma em máquinas diferentes, contando que os endereços são dados, uma solução para evitar esta necessidade de parâmetros era um ficheiro de configuração partilhados pelos Peers e pelos Master Servers com o endereço de todos os Master Servers, contudo o grupo não considerou este problema relevante para o contexto deste trabalho.

**Conclusão**

O grupo conseguiu implementar todas as funcionalidades que queria, consideramos que chegamos a uma solução robusta que tem em conta todos os factores pedidos com um grau de satisfação elevado.

// READ ME

Iniciar o RMI com “java rmi.Rmi”

iniciar os Servers com “java server.main.Server <porta>”, porta entre 5000 e 5002 inclusivé

Iniciar os Peers com “java peer.main.Peer <peerID>”, peerID > 0

Iniciar o Client com “java client.Client <argumentos>”, os mesmo argumentos que no trabalho 1, no caso da Backup há um argumento extra no final, 0 para backup sem encriptação, 1 para backup com encriptação (ocupa mais espaço)

// GUIÃO

INICIAR RMI

INICIAR 3 MasterServer

INICIAR 3 Peers

(notar que as ligaçõe estão distribuídas)

Fazer um backup replication degree 2 e encrypted 1 (no peer 1, peninha)

(notar que existem mais chunks)

(notar que o metadado estão a ser guardado entretanto)

Desligar um master

server(notar que as ligações passam para outro)

Fazer restore do backup anterior

Fazer backup de replication degree 2 e encrypted 0 (no peer 2, pena)

(notar o tamanho diferente)

Desligar o Peer 3

(notar que desapareceu do master peer)

Fazer delete do backup anterior (o do peer 1)

Ligar o Peer 3

(notar que os chunks desaparecem)

Desligar o Peer2

Apagar os metadados do Peer2

Inicializar o Peer2

(notar que os metadados estão de volta)

Fazer RECLAIM 64 no Peer 3

(notar que ficou com o Chunk mais pequeno)