U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Sistemas Distribuídos
EIC0036

Ano Letivo 2015/2016

Relatório Final

"*Distributed Backup System*"

Grupo T5G12

Diogo Belarmino Coelho Marques
up201305642@fe.up.pt

Pedro Miguel Pereira de Melo
up201305618@fe.up.pt

3 de Abril de 2016

Backup Subprotocol

A especificação deste projeto propunha que as seguintes ações fossem realizadas quando um *peer* recebesse uma mensagem ***PUTCHUNK***:

1. armazenar esse *chunk*;
2. esperar durante um período de tempo aleatório compreendido entre 0 e 400 ms;
3. enviar mensagem de confirmação ***STORED***;

No entanto, esta abordagem podia esgotar rapidamente o espaço destinado ao armazenamento de *chunks* em cada *peer*, se todos os *peers* aceitassem e armazenassem os *chunks* recebidos nas mensagens ***PUTCHUNK***. Sendo assim, ao nível do *peer* que recebe uma mensagem ***PUTCHUNK*** são realizadas as seguintes ações, pela ordem apresentada:

1. esperar durante um período de tempo aleatório compreendido entre 0 e 400 ms;
2. entretanto, ir registando número de confirmações recebidas dos *peers* (número de mensagens ***STORED***) para esse *chunk*;
3. se o número de confirmações recebidas (nível de replicação atual) for ainda inferior ao nível de replicação desejado, armazenar esse *chunk* e enviar mensagem de confirmação ***STORED***; caso contrário, a mensagem deve ser ignorada;

A nossa melhoria do sub-protocolo de *backup* não requer quaisquer mensagens adicionais nem alterações às informações nelas contidas, garantindo assim compatibilidade e interoperabilidade com os vários *peers* e com as diferentes implementações deste protocolo.

Restore Subprotocol

A especificação deste sub-protocolo não era a mais indicada para receber *chunks* de grandes dimensões. Para evitar que a rede *multicast* fosse sobrecarregada com mensagens ***CHUNK***, visto que apenas um dos *peers* dos vários ligados à rede está interessado em receber os *chunks* de determinado ficheiro, estes poderão ser enviados diretamente em *unicast* ao *peer* que os requisitou. Esta alteração implica a criação de um *socket* na máquina do *initiator peer* que invocou a operação de recuperação (*restore*) que receberá os *chunks* enviados em *unicast* pelos vários *peers*.

Por outro lado, um dos requisitos presentes na especificação deste projeto dizia que os *peers* deviam esperar um intervalo de tempo aleatório distribuído entre 0 e 400ms antes de enviar uma mensagem *CHUNK*, e ir registando entretanto as mensagens *CHUNK* recebidas dos restantes *peers*. Se pelo menos uma mensagem fosse recebida durante esse intervalo de tempo para esse chunk, esse *peer* cancelaria o envio.

A nossa abordagem procurou então reduzir o volume das mensagens ***CHUNK*** enviadas em *multicast, com a desvantagem de enviar duas mensagens para cada chunk* (uma em *multicast*, apenas contendo cabeçalho, tendo sido criada uma versão 2.0 da mensagem ***CHUNK*** sem corpo, e outra em *unicast*, contendo os *bytes* do ficheiro). Foi também criado uma nova versão da mensagem ***GETCHUNK***. Neste novo formato foi acrescentada uma linha adicional ao cabeçalho da mensagem com um único campo que representa a porta do *socket criado pelo peer* que invocou esta operação, para onde serão enviados os *chunks* do ficheiro requisitado.

Ao nível do *peer* que invoca a operação de *restore* do ficheiro são realizadas as seguintes ações pela ordem apresentada:

1. cria *socket* UDP *unicast* na máquina local, numa porta atribuída automaticamente pelo sistema operativo;
2. envia mensagens ***GETCHUNK*** (versão 2.0) sequencialmente para cada *chunk* do ficheiro armazenado remotamente contendo no cabeçalho extra a porta do *socket* UDP;
3. aguarda a recepção dos *chunks* (mensagens do tipo ***CHUNK***) através do *socket* UDP;

Com vista também a melhorar o funcionamento do sub-protocolo base, foi adicionado um *timeout* à operação de *restore* por parte do *peer* que a inicia, sem qualquer consequência para os *peers* nela intervenientes (a interoperabilidade continua garantida):

1. contador de tentativas inicializado a 1;
2. se não receber nenhuma mensagem ***CHUNK*** durante um período fixo de 2 segundos, é incrementado o contador de tentativas (i++);
3. se receber pelo menos *uma* mensagem ***CHUNK*** durante esse período de tempo, o valor do contador é reposto (i = 1);

4. ao fim de 5 tentativas, se nenhuma mensagem ***CHUNK*** for recebida no *socket*, a operação de recuperação (*restore*) do ficheiro termina com insucesso;

Ao nível do *peer* que recebe a mensagem ***GETCHUNK***, são realizadas as seguintes operações na nossa implementação, pela ordem apresentada:

SE (peer tiver armazenado esse chunk**)**
espera durante um período de tempo aleatório compreendido entre 0 e 400ms;
entretanto, vai registando as mensagens ***CHUNK*** recebidas para este *chunk;*

SE (não recebeu nenhuma mensagem ***CHUNK*****)**
cria socket *unicast* na máquina local;
envia mensagem ***CHUNK*** apenas com cabeçalho em *multicast* para todos os *peers*;
envia mensagem ***CHUNK*** completa em *unicast* para peer que invocou sub-protocolo de *restore;*

Delete Subprotocol

A nossa implementação desta melhoria consistiu na criação de uma nova mensagem de confirmação ***DELETED*****,** e de uma nova versão da mensagem ***DELETE*** existente, sem quaisquer campos adicionais. Esta alteração na versão servirá exclusivamente para indicar o tipo de resposta esperada nos *peers* que recebem a mensagem.

A nossa abordagem consistiu em registar na base de dados do *peer* que iniciou a operação de *backup* de um ficheiro os *peers* que guardaram *chunks* desse ficheiro. Quando esse *peer* invocasse a operação de *delete* sobre um ficheiro, esta poderia ser executada um número de vezes suficiente para receber a confirmação de todos os *peers* que tivessem guardado *chunks* desse ficheiro, libertando assim todo o espaço por eles ocupado.

Ao nível do *peer* que invoca a operação de *delete* sobre um ficheiro são realizadas as seguintes operações na nossa implementação, pela ordem indicada:

1. obter uma lista com os *peers* que têm pelo menos um *chunk* do ficheiro a apagar;
2. enviar nova versão da mensagem ***DELETE*** para todos os *peers*;
3. esperar durante um período tempo fixo (inicialmente 1 segundo);
4. entretanto, ir registando os *peers* que enviaram uma confirmação ***DELETED***, removendo esses peers da lista de peers obtida inicialmente;
5. se a lista de *peers* estiver vazia, retornar;
6. caso contrário, repetir todas os passos a partir do passo nº 2, no máximo 5 tentativas, passando o tempo de espera para o dobro em cada iteração.

Ao nível do *peer* que recebe a mensagem ***DELETE*** de um ficheiro, são realizadas as seguintes operações na nossa implementação, pela ordem indicada:

1. apaga todos os *chunks* desse ficheiro nele armazenados;
2. se a mensagem recebida tiver uma versão diferente da especificada neste protocolo (1.0), envia confirmação ***DELETED;***

Com vista também a melhorar o funcionamento do sub-protocolo base, foi ainda implementada a sugestão de enviar várias mensagens ***DELETE*** para assegurar que todos os *peers* ligados à rede recebem este pedido, com possível perda de alguns pacotes. Esta melhoria aplica-se apenas ao sub-protocolo definido na especificação deste projeto.

Reclaim Subprotocol

A especificação deste projeto propunha a execução das seguintes ações no *peer* quando fosse recebida uma mensagem ***REMOVED*** para determinado *chunk*:

```
SE (peer tiver armazenado esse chunk)
    atualizar contagem de peers que armazenam esse chunk;

    SE (novo grau de replicação < grau de replicação desejado)
        esperar durante um período de tempo aleatório compreendido entre 0 e 400 ms;
        entretanto, ir registando número de mensagens PUTCHUNK recebidas;

        SE (nenhuma mensagem PUTCHUNK tiver sido recebida)
       iniciar sub-protocolo de backup para esse chunk;
```

Esta implementação tem a seguinte desvantagem: se um *peer* falhar no sub-protocolo de *backup*, o nível de replicação desse *chunk* poderá continuar a ser inferior ao desejado. Na nossa implementação, ao nível do *peer* que invoca a operação de *reclaim* de vários *chunks* são realizadas as seguintes operações, pela ordem indicada:

SE (contagem local de *peers* que armazenam esse *chunk* for 1**)**
/ nenhum dos peers além do peer que invocou o sub-protocolo de reclaim tem esse chunk */*
iniciar sub-protocolo de *backup* para esse *chunk*
SENÃO
esperar por mensagens ***PUTCHUNK*** durante 1 segundo;

SE (pelo menos uma mensagem ***PUTCHUNK*** tiver sido recebida**)**
REPETIR 5 vezes, multiplicando o tempo de espera por dois a cada iteração
esperar durante um intervalo de tempo fixo (inicialmente 1 segundo)
entretanto, ir registando mensagens ***STORED*** recebidas
SE (número de mensagens ***STORED*** recebidas **>=** nível de replicação desejado**)**
break;
SENÃO
/* nenhum dos peers além do *peer* que invocou o sub-protocolo de *reclaim* tem esse *chunk* */
iniciar sub-protocolo de *backup* para esse *chunk*;

A nossa melhoria do sub-protocolo de *reclaim* não requer quaisquer mensagens adicionais nem alterações às informações nelas contidas, garantindo assim compatibilidade e interoperabilidade com os vários *peers* e com as diferentes implementações deste protocolo.