\section{Objetivo da Aplicação}

O objetivo desta aplicação é melhorar a implementação do projeto anterior adicionando as funcionalidades que nós achamos serem necessárias para a utilização do programa num ambiente empresarial de pequena dimensão. Achamos também por bem implementar as melhorias sugeridas na iteração anterior.

Para atingir estes objetivos implementamos funcionalidades que fortaleceram a segurança da aplicação, a tolerância a falhas e reduziram a carga na rede.

\section{Pressupostos}

Para implementar este sistema partimos de uma série de pressupostos:

- Os computadores encontram-se na mesma LAN;

- Algumas mensagens enviadas podem ser perdidas sendo que acabarão por chegar ao destino através de *time-outs* e retransmissões;

- Computadores podem falhar, perdendo ficheiros ou *metadata* importante. No entanto, podem recuperar dessas mesmas falhas;

- Todos os computadores da rede se regem pelos protocolos estabelecidos e assume-se que os dados guardados não serão alterados;

\section{Especificação}

O problema a ser resolvido por este sistema é a criação de um sistema distribuído *peer-to-peer* de *backup* e restaurode ficheiros para uma rede local (LAN). O sistema utilizaria espaço disponível no disco para armazenar ficheiros de outros computadores da LAN num ambiente cooperativo. Cada computador pode determinar quanto espaço aloca para este propósito e pode a qualquer momento limitá-lo ou expandi-lo.

Para isto foi criado um protocolo de comunicação em que a maioria das mensagens entre *peers* é enviada por UDP Multicast sendo que algumas em específico são enviadas por TCP para evitar uma carga exagerada da rede.

Cada ficheiro tem um computador de origem, que possui a cópia original do mesmo. Para cada ficheiro é gerado um identificador obtido aplicando SHA256 (uma função de encriptação) a uma string que contém o caminho para o ficheiro e a última data de modificação do mesmo, de modo a que esta seja única.

Cada ficheiro é dividido em *chunks* de 64000 bytes, para que o limite máximo de um pacote UDP (65536 bytes / 64KB) não seja excedido. O sistema efetua o *backup* de cada *chunk* individualmente. Cada *chunk* é identificado pelo par identificador do ficheiro – número do *chunk*. O último *chunk* é sempre mais pequeno que os outros para que ao receber um ficheiro na altura do restauro o sistema saiba qual é o último *chunk* desse mesmo ficheiro. No caso do último ter o tamanho máximo de um *chunk*, cria-se mais um *chunk* de tamanho 0. O restauro de cada *chunk* é independente do restauro de outros.

Cada *chunk* é encriptado com recurso ao algoritmo DES (*Data Encryption Standard*), sendo usada como chave a *password* do utilizador (já encriptada por sua vez recorrendo ao mesmo algoritmo, usando como chave um conjunto aleatório de caracteres).

Cada *chunk* tem um grau de replicação determinado pelo utilizador, que indica o número de máquinas que terão de armazenar esse mesmo *chunk*. Se o grau de replicação já tiver sido atingido, mais nenhuma máquina armazenará esse *chunk*, mesmo que lhe seja pedido. Isto permite a racionalização do espaço em disco de modo a que haja sempre o máximo de espaço possível disponível na rede. Se o número de replicações não puder ser atingido no estado atual da rede, quando se juntar mais uma máquina à rede ser-lhe-á pedido que guarde o *chunk* isto até que o grau de replicação seja satisfeito.

Como foi referido acima, o sistema permite a gestão do espaço disponível para o serviço. Se um utilizador decidir recuperar espaço em disco, alguns *chunks* poderão ser apagados. Nesta situação o serviço envia uma mensagem para cada *chunk* que foi apagado. No momento de receção desta mensagem, um computador que tenha no disco esse *chunk* certifica-se que esse *chunk* voltará a atingir o grau de replicação anterior.

Quando um ficheiro é apagado pelo utilizador este envia uma mensagem para a rede e todos os *peers* apagam os *chunks* correspondentes a esse ficheiro.

Ao utilizador é fornecida uma interface na linha de comandos para interagir com o sistema. Esta permite-lhe:

- Fazer login com uma combinação utilizador-password para entrar na aplicação;

- Escolher uma de várias configurações de rede já criadas, ou criar uma nova;

- Executar qualquer uma das operações referidas acima (backup de um ficheiro, restauro de um ficheiro, apagar um ficheiro, gestão do espaço disponível para a aplicação);

- Sair da aplicação.

\section{Protocolo}

O protocolo é dividido em 5 subprotocolos:

- Backup de ficheiros

- Backup da base de dados de ficheiros

- Restauro de ficheiros

- Eliminação de ficheiros

- Gestão do espaço disponível para a aplicação

São utilizados 3 canais de comunicação:

- Canal de Controlo (MC), onde são enviadas todas as mensagens que não sejam de backup ou restauro;

- Canal de Backup (MDB);

- Canal de Restauro (MDR);

\subsection{Formato das Mensagens}

As mensagens são compostas por duas partes: o header e o body. O body só é definido em algumas mensagens.

O header é composto por uma série de campos separados por ' ' e é delimitado terminada pela sequência '0xD' '0xA' que simboliza <CRLF>.

O body representa o chunk em si e tem tamanho variável. Se for menor que 64000 bytes é o último chunk do ficheiro.

Estrutura:

<MessageType> <Version> <FileId> <ChunkNo> <ReplicationDeg> <CRLF>

Alguns destes campos podem não ser utilizados por algumas mensagens mas a ordem é sempre esta.

Segue-se a explicação dos campos:

- <MessageType> - O Tipo da mensagem a ser enviada (PUTCHUNK, DELETE, GETCHUNK, …);

- <Version> - A versão do protocolo;

- <FileId> - Obtido usando a função de *hash* criptográfica SHA256. Como o nome indica o seu tamanho é 256 bits (32 bytes) e é codificado com uma sequência de 64 caracteres. Cada byte da *hash* é codificado com dois caracteres ASCII correspondentes à representação hexadecimal desse byte (0xB2 -> 'B''2') e é representada do byte mais significativo ao menos significativo;

- <ChunkNo> - Inteiro e começa em 0. Codificado na sequência de caracteres ASCII correspondente à representação decimal deste número;

- <ReplicationDeg> - Grau de replicação desejado para o ficheiro.

\subsection{Subprotocolo de Backup}

Para fazer backup o peer-iniciador envia uma mensagem para o canal MDB cujo body é o conteúdo desse chunk. A mensagem também inclui o id do chunk (FileId + ChunkNo) e o grau de replicação.

Estrutura:

PUTCHUNK <Version> <FileId> <ChunkNo> <ReplicationDeg> <CRLF> <CRLF> <Body>

Quando um peer recebe um PUTCHUNK, após um intervalo aleatório entre 0 e 400 milissegundos, verifica se já foi atingido o grau de replicação desejado e caso já tenha sido atingido não faz nada. Caso contrário o chunk é guardado em disco e é enviada para o canal MC uma mensagem com o seguinte formato:

STORED <Version> <FileId> <ChunkNo> <CRLF> <CRLF>

O iniciador recolhe mensagens de confirmação durante meio segundo e se o número de mensagens recebidas for menor que o grau de replicação envia a mesma mensagem PUTCHUNK e duplica o tempo de receção de mensagens de confirmação. Este procedimento é repetido até um máximo de 5 vezes.

Um peer conta o número de confirmações para cada *chunk* que tem guardado e guarda essa informação na base de dados.

\subsection{Subprotocolo de Backup da Base de Dados}

Para oferecer persistência dos dados (dados do utilizador, chunks guardados, ficheiros replicados, metadados, etc.) o sistema guarda todos estes num ficheiro (database.cu).

Para garantir que este ficheiro resiste a falhas o sistema replica a base de dados pela rede quando é executado um backup, eliminação de ficheiros ou de chunks (devido à remoção de espaço disponível à aplicação), recorrendo à seguinte mensagem:

PUTDB <hashDB> <ChunkNo> <CRLF> <CRLF> <Body>

A <hashDB> é um identificador único obtido através da junção do nome de utilizador, da sua password e do seu e-mail, encriptado recorrendo à função de *hash* criptográfica SHA256.

Para recuperar a base de dados é usada a seguinte mensagem:

GETDB <hashDB> <ChunkNo> <CRLF> <CRLF>

Para não sobrecarregar a rede, cada peer espera um intervalo aleatório entre 0 e 400 milissegundos até enviar a mensagem DB. Se receber um DB com a mesma hashDB durante este intervalo, não envia a mensagem DB.

No ato de receção desta mensagem um peer que possua uma cópia da base de dados pretendida responde no canal MDR com a seguinte mensagem:

DB <hashDB> <chunkNo> <CRLF> <CRLF> <Body>

No arranque do programa, é oferecido ao utilizador a possibilidade de carregar a base de dados existente no disco ou carregar através da rede. Para carregar a base de dados através da rede, é enviado um pedido de GETDB.

\subsection{Subprotocolo de Restauro}

Neste protocolo são utilizados os canais MC e MDR.

Quando um utilizador pretende restaurar um ficheiro o serviço envia para o canal de controlo uma mensagem GETCHUNK para cada *chunk* do mesmo com o seguinte formato:

GETCHUNK <Version> <FileId> <ChunkNo> <CRLF> <CRLF>

No ato de receção desta mensagem um peer que possua uma cópia deste do chunk pretendido cria um socket TCP e responde pelo canal MDR com a seguinte mensagem:

ME <FileId> <ChunkNo> <IP> <Porta>

O par <IP> <Porta> são correspondentes ao socket criado. O peer ficará agora à espera da conexão do peer-iniciador, durante um período máximo de 5 segundos.

Para cada chunk o peer-iniciador só abre o socket TCP com os dados da primeira mensagem ME que receber. Ao verificar uma nova ligação, o peer detentor envia então os dados do chunk utilizando uma mensagem com o seguinte formato:

CHUNK <Version> <FileId> <ChunkNo> <CRLF> <CRLF> <Body>

Em comparação com a implementação original, utilizando TCP obtemos maior grau de fiabilidade – visto que o protocolo TCP é mais fiável que o UDP - e diminuímos o tráfego na rede, visto que apenas o peer que pretende receber o chunk o recebe.

\subsection{Subprotocolo de Eliminação de Ficheiros}

Quando o utilizador desejar eliminar um ficheiro, o serviço envia a seguinte mensagem pelo canal MC:

DELETE <FileId> <CRLF>

Um peer ao receber esta mensagem elimina todos os chunks com esse FileId, e responderá pelo mesmo canal com a seguinte mensagem:

DELETED <FileId> <CRLF>

Caso a replicação de algum chunk do ficheiro não chegue a 0 o peer-iniciador cria uma thread que irá enviar periodicamente – de 30 em 30 minutos - mensagens DELETE de modo a apagar os chunks desse ficheiro na rede.

Os dados da mesma são guardados mesmo que o programa termine ou falhe, ou seja, são persistentes.

\section{

\subsection{Subprotocolo de Gestão do espaço disponível para a aplicação}.

Para efetuar a gestão do espaço disponível em disco, disponibilizamos duas opções na nossa interface com utilizador:

- Reduzir o espaço alocado;

- Aumentar o espaço alocado.

Por defeito são alocados 100MB de espaço, podendo ser posteriormente alterados para um valor à vontade do utilizador, sendo reservado um espaço mínimo de 64KB.

Quando o utilizador remove espaço e o restante é menor que o espaço ocupado pelos chunks armazenados, o sistema a seguinte mensagem por cada chunk removido:

REMOVED <Version> <FileId> <ChunkNo> <CRLF> <CRLF>

Quando recebe esta mensagem, um peer que tenha uma cópia local do chunk atualiza a contagem local desse chunk. Se a contagem desse chunk descer abaixo do grau de replicação desejado o peer inicia o subprotocolo de backup após um intervalo aleatório entre 0ms e 400ms. Se no intervalo receber um PUTCHUNK não inicia o subprotocolo.

Este peer recolhe todas as mensagens de confirmação para esse chunk e, se a replicação ficar abaixo do desejado, cria uma thread que espera que novos utilizadores entrem na rede ou que um peer apague um ficheiro - que não o(s) supervisionado(s) - e inicia novamente o subprotocolo de backup.

\section{Main Features}

\begin{itemize}

\item Security

\begin{itemize}

\item Peer authentication using encrypted passwords

\item Guaranteed chunk confidentiality, encrypting the data sent from a computer to the network

\end{itemize}

\item Fault tolerance

\begin{itemize}

\item Use of write-ahead logging, so the aplication is able to recover from a crash and execute integrity checks to detect corruption in files

\item Avoidance of corrupted data through cached backups of the active file(s)

\end{itemize}

\item Communication

\begin{itemize}

\item Implementation of the TCP protocol in some cases where not all the peers need to receive the packet(s)

\item Possibility to name the communication channels making use of a DNS

\end{itemize}

\item In addition to this we will implement all the previous requested enhancements

\begin{itemize}

\item Backup: After waiting a random time, a peer will save the chunk in the hardrive only if the current replication degree is lower than the desired degree

\item Restore: Implementation of the TCP protocol to ensure that only the necessary client receives the data requested without fail, putting less strain on the network

\item Delete: Implement a response message to the delete request. This message will allow the host to send the delete request until all the responses indicate the replication degree of this chunk has reached zero.

\item Space Reclaim: Similar to the delete protocol, a list of chunks with low degree will be kept and a request to backup a chunk will be sent to newly connected clients until the replication degree is satisfied.

\end{itemize}

\item Core program functions

\begin{itemize}

\item Backup a file in the network with the desired replication degree

\item Restore a file previously backed up

\item Remove a file previously stored in the network

\item Disk allocation management

\end{itemize}

\end{itemize}

\section{Target Platforms}

\begin{itemize}

\item Java standalone application for Windows/Linux/Mac

\end{itemize}

\section{Additional Services and Improvements}

If time allows us, we will develop a Graphical User Interface for the main application and an Android port, while also allowing interoperability between all the target platforms.

\section{Target}

With this project we hope to achieve a final grade of 17 if the Android Version is not implemented and 19 if it is implemented.

\end{document}

\subsection{Workspaces}

Ao iniciar o programa, é possível fazer login com um workspace previamente criado ou criar um novo. Aquando da criação do workspace, é necessário introduzir o username, password e E-mail pretendido.

Todos os ficheiros guardados – base de dados, chunks guardados, configurações de rede, etc. – são únicos para cada workspace, sendo criada uma pasta individualizada para cada workspace.

Isto permite usar a aplicação em vários sítios diferentes, evitando deste modo a junção desnecessária de ficheiros, mantendo a independência dos vários contextos de trabalho, por exemplo trabalho em casa ou na empresa. O E-mail é usado para diferenciar a hash da base de dados, caso dois utilizadores da mesma rede possuam workspaces com o mesmo username e password.

A password é encriptada com recurso ao algoritmo DES (*Data Encryption Standard*).

\subsection{Encriptação dos Chunks}

Para garantir que os ficheiros não são lidos ou percebidos por peers que tenham efetuado backup dos mesmos, os seus chunks são encriptados no momento da divisão do ficheiro, com recurso ao algoritmo DES (*Data Encryption Standard*), usando como chave a \textit{password} do utilizador (já encriptada por sua vez recorrendo ao mesmo algoritmo, usando como chave um conjunto aleatório de caracteres).

No momento do restauro do ficheiro, ao juntar os vários num só ficheiro como originalmente, os vários chunks são decriptados, repondo o conteúdo original.

\subsection{Implementação de TCP}

Foi implementado o protocolo TCP no subprotocolo de restauro, de modo a aumentar significativamente a fiabilidade da comunicação, bem como reduzir do mesmo modo a carga na rede.

\subsection{Configurações de Rede}

Após efetuar o login é possível carregar configurações de rede já existentes – identificadas por um nome – ou criar uma nova, possibilitando deste modo guardar várias configurações. Isto é bastante útil, pois imaginando uma situação de trabalho, poder trabalhar em diferentes departamentos sem ter de reintroduzir as suas configurações de rede.

\subsection{Log}

Foi implementado também um log, que serve para duas funções:

- mostrar o estado do programa na linha de comandos

- produzir um ficheiro de texto com todas ações efetuadas

Esta última acaba por funcionar como um “histórico”, visto que todas as ações estarão associadas a um timestamp e estarão guardadas num ficheiro para consulta posterior.

\section{Write ahead logging}

A nossa implementação de write ahead passa por partilhar a base de dados do utilizador na rede. Isto permite que a aplicação sobreviva crashes e a falhas completas, pois no caso de não existir a base de dados, o utilizador pode pedi-la à rede.

Aquando da recuperação da base de dados, é enviada uma mensagem de DELETED por cada ficheiro que anteriormente (antes do crash) estivesse “backed-up” no computador do utilizador, de modo a informar que o utilizador perdeu todos os seus chunks aos quais tinha feito backup.

\section{Conclusão}

Tendo em conta o que nos propusemos a implementar e o que foi realmente implementado, podemos concluir que atingimos praticamente todos os objetivos.

Foi especialmente difícil trabalhar sobre código que já vinha do projeto anterior, visto que certas coisas que era preciso assumir neste projeto não são assumidas, e como tal, foi preciso uma extensa reorganização e mudança de toda a estrutura feita anteriormente, bem como construir sob o que já estava feito.

Obviamente, cimentámos o nosso conhecimento sobre o protocolo TCP e UDP, que já vinha da Unidade Curricular de Redes de Comunicação do semestre passado. Realçamos também o conhecimento sobre como encriptar informação em java e sobre como desenhar protocolos de comunicação. No geral, podemos afirmar que estamos bastante mais preparados para desenvolver aplicações dentro deste domínio.

\section{Melhorias a Implementar}

De forma a melhorar o nosso projeto, poderíamos implementar uma GUI (Grafical User Interface), de modo a melhorar a interação com o utilizador.

Poderíamos também ter portado a nossa aplicação para dispositivos móveis (Android, iOS e Windows Phone), que iria oferecer a possibilidade de fazer todas as operações disponíveis na versão Desktop, mas sobre mensagens e imagens.

Implementar Sliding Window para o protocolo TCP.

\section{Manual do Utilizador}

A lista de passos a efetuar para testar o nosso programa é a seguinte:

- Descompactar o ficheiro .zip submetido;

- Correr o ficheiro Main.java que se encontra dentro da pasta main que está dentro da pasta src (src/main/Main.java);

- Fazer o login/criar uma conta;

- Criar configuração de rede/Carregar uma já criada;

- Executar qualquer uma das operações referidas acima (backup de um ficheiro, restauro de um ficheiro, apagar um ficheiro, gestão do espaço disponível para a aplicação);

- Sair da aplicação se assim desejar.

\subsection{Exemplos de Execução}