Projeto final: FD8 Torrent

Matheus Pimenta – 09/0125789
Felipe Rodopoulos – 10/0100601

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Disciplina: Teleinformática e Redes 2

2 de julho de 2014

Para a finalização da disciplina Teleinformática e Redes 2, foi pedido um projeto final cuja aplicação é o compartilhamento de arquivos em uma rede local. Este documento descreve a implementação.

1 Principais desafios

Esta seção descreve os principais desafios do sistema e como eles foram solucionados.

1.1 Estrutura de dados para representação do sistema de arquivos

O primeiro desafio enfrentado foi elaborar uma boa representação do sistema de arquivos, que facilitasse implementações simples e eficientes das quatro operações básicas: criação, recuperação, atualização e remoção de pastas e arquivos.

A solução pensada inicialmente utilizava:

- um *map* de *strings* para objetos do tipo *Folder*, onde a *string* chave deveria ser o caminho completo da pasta
- o tipo *Folder* possuía um *map* de *strings* para objetos do tipo *File*, onde a *string* chave deveria ser apenas o nome do arquivo
- o tipo *File* possuía campos para informar o tamanho do arquivo, os pares responsáveis e o usuário que enviou o arquivo

Esta solução foi considerada ruim, pois ao modificar o nome de uma pasta, era necessário passar pelo mapa atualizando chaves de muitas outras pastas. Além disso, o armazenamento físico de um arquivo em um par iria precisar que as pastas fossem criadas fisicamente.

Pensando em uma solução com a estrutura de uma árvore, chegamos na seguinte estrutura de dados:

- o tipo *Folder* possui um *map* de *strings* para objetos do tipo *Folder*, onde a *string* chave é apenas a concatenção de "/" com o nome da pasta
- o tipo *Folder* possui também um *map* de *strings* para objetos do tipo *File*, onde a *string* chave é a concatenação de "/" com o nome do arquivo

- o tipo *File* possui campos para informar o tamanho do arquivo, os pares responsáveis, o usuário que enviou o arquivo e um número identificador único
- um objeto do tipo *Folder* denominado *rootFolder* foi criado no sistema de arquivos

Com esta solução, a modificação do nome de uma pasta se resume a trocar a entrada da pasta no mapa de sua pasta mãe. Além disso, com um número identificador único para cada arquivo, o armazenamento físico pode ser feito em um grande pasta que contém apenas os arquivos físicos.

1.2 Criação de um algoritmo de duplicação de arquivos

Para duplicar arquivos entre os pares, foi desenvolvido o algoritmo Pimenta–Rodopoulos, que seleciona os arquivos que precisam ser duplicados e os distribui procurando os pares com menos arquivos. Procurou-se pares com menos arquivos para facilitar o trabalho do algoritmo de balanceamento.

```
Data: rootFolder
Result: um conjunto de comandos de duplicação
cmds ← []
peersFiles ← o mapa de IPs para conjuntos de arquivos obtido de rootFolder
filesPeers ← o mapa de IDs para pares de IPs obtido de rootFolder
toDup ← o conjunto dos arquivos de filesPeers com apenas apenas um IP válido
for file ∈ filesPeers do
    if peersFiles.contains(file.value.peer1) then
        file.value.peer2 ← 0
    else
        file.value.peer1 ← file.value.peer2
        file.value.peer2 ← 0
    end
end
while toDup.size() > 0 do
    peer ← o par com menos arquivos em peersFiles
    f ← null
    for file ∈ toDup do
        if peer.value.contains(file) then
            f ← filesPeers[file]
            pare
        end
    end
    if f = null then
        peersFiles.erase(peer.key)
    else
        toDup.erase(f.key)
        peer.value.insert(f.key)
        f.value.peer2 ← peer.key
        cmds.insert(DuplicationCommand(f.key, f.value.peer1, f.value.peer2))
    end
end
```

Algorithm 1: Algoritmo Pimenta–Rodopoulos

1.3 Criação de um algoritmo de balanceamento

Para balancear a quantidade de arquivos entre os pares, de modo que cada par possua no máximo $\lceil \frac{\#arquivos}{\#pares} \rceil$ arquivos físicos, foi desenvolvido o algoritmo Rodopoulos–Pimenta.

```
Data: rootFolder e averageFiles
Result: um mapa de IDs para comandos de balanceamento
cmds ← []
peersFiles ← o mapa de IPs para conjuntos de arquivos obtido de rootFolder
for file ∈ filesPeers do
    if peersFiles.contains(file.value.peer1) then
        file.value.peer2 ← 0
    else
        file.value.peer1 ← file.value.peer2
        file.value.peer2 ← 0
    end
end
while true do
    minimalPeer ← o par com menos arquivos em peersFiles
    maximalPeer ← o par com mais arquivos em peersFiles
    if maximalPeer.value.size() ≤ averageFiles then
        pare
    end
    for file ∈ maximalPeer.value do
        if minimalPeer.value.contains(file.key) then
            continue
        end
        maximalPeer.value.erase(file.key)
        minimalPeer.value.insert(file.key)
        f ← referência do objeto com ID file.key contido em rootFolder
        if f.peer1 = maximalPeer.key then
            f.peer1 ← minimalPeer.key
        else
            f.peer2 ← minimalPeer.key
        end
        if cmds.contains(file.key) then
            balCmd ← cmds[file.key]
            balCmd.peer1 ← f.peer1
            balCmd.peer2 ← f.peer2
        else
            cmds[file.key] ← BalancingCommand(file.key, maximalPeer.key, f.peer1, f.peer2)
        end
        pare
    end
end
```

Algorithm 2: Algoritmo Rodopoulos–Pimenta

2 Requisitos e implementações

Esta seção descreve a implementação de cada requisito do projeto final.

2.1 Cadastro de usuários

Requisito – O sistema deve permitir o cadastro de usuários com ID único.

Implementação – Para implementar um cadastro de usuários, decidimos que o mais adequando seria criar sessões de acesso ao sistema através de um nome de usuário único, ou seja, ao tentar acessar o sistema, o usuário deve entrar com um nome que não exista na tabela de usuários.

Para implementar a sessão de acesso com nome de usuário único, foi criado um número identificar único de sessão que só é válido até que o usuário saia do sistema. O endereço IP e o número identificador de sessão são utilizados para detectar falhas de pares.

2.2 Ações de usuários

Requisito – Usuários cadastrados podem:

- Acessar o estado do sistema
  - Número de arquivos no sistema
  - Capacidade total armazenada no sistema
  - Distribuição dos arquivos entre os pares
  - Número de pares ativos (alertar se houver apenas dois)
- Visualizar a árvore de diretórios do sistema de arquivos
- Criar diretórios
- Enviar arquivos
- Renomear diretórios/arquivos
- Remover diretórios/arquivos

Implementação – Para visualizar o número de arquivos, a capacidade total armazenada e o número de pares ativos no sistema, foi feita uma divisão lateral na interface do sistema que atualiza as informações de estado constantemente utilizando AJAX. Além das informações pedidas, são informados também o nome e o IP do usuário.

Para visualizar a árvore de diretórios do sistema de arquivos, foi criada um variável no navegador de páginas que armazena o caminho completo da pasta que está sendo visualizada atualmente. Esta variável é exibida na interface do sistema, juntamente com a quantidade de subpastas, arquivos e capacidade da pasta atual. Na área principal da interface, as subpastas e arquivos são listados. Para navegar entre as pastas, basta clicar no nome de uma pasta para entrar nela, ou clicar no botão "Nível acima" para ir até a pasta mãe da pasta atual. A listagem é atualizada automaticamente caso hajam mudanças no sistema de arquivos.

Para criar diretórios e enviar arquivos, foram criados dois botões ao lado do botão "Nível acima", que disparam requisições AJAX ao servidor *web*. O servidor *web* verifica o tipo de requisição, tentar realizar a operação chamando o módulo responsável pelo sistema de arquivos e retorna códigos de erro ou sucesso para o navegador.

Para renomear diretórios ou arquivos, cada item da listagem da pasta atual possui um pequeno botão com a imagem de um lápis que, ao ser clicado, abre uma caixa de entrada para o novo nome. Um botão de confirmação ao lado da caixa dispara a requisição AJAX para o servidor *web*.

Para remover diretórios ou arquivos, cada item da listagem da pasta atual possui também um pequeno botão com a imagem de um X vermelho que dispara a requisição AJAX.

Para visualizar a distribuição dos arquivos entre os pares, decidimos criar dois botões na linha de um arquivo da listagem. Cada botão contém a URL com o IP de um dos pares que está armazenando o arquivo naquele momento.

2.3 Interface do usuário

Requisito – A interface utilizada pelos usuários deve ser um *browser* (navegador) compatível com HTML 5.0.

Implementação – A interface do sistema foi implementada em HTML5 e CSS3. O sistema foi testado nos navegadores Google Chrome, Mozilla Firefox e Internet Explorer.

2.4 Utilização da aplicação

Requisito – Para utilizar o sistema o usuário deverá apontar seu *browser* para o endereço http://localhost:8080/

Implementação – A porta do servidor *web* foi parametrizada na etapa de compilação do sistema. A compilação final foi feita com a porta 8080, como pedido, de modo que o usuário pode apontar seu navegador para o endereço http://localhost:8080/ e utilizar o sistema.

Para iniciar o sistema, a aplicação abre um janela gráfica. Um botão central do tipo *toggle* foi colocado na janela para iniciar e finalizar o sistema. Ao iniciar o sistema, um botão aparece na parte superior da janela, que dispara a abertura do navegador padrão do sistema operacional no endereço http://localhost:8080/.

2.5 Descobrimento de pares

Requisito – Um par deve se capaz de descobrir os demais pares que estiverem executando na rede.

Implementação – Para solucionar este problema, pensou-se inicialmente em abrir um *socket* UDP para enviar segmentos *broadcast* e *multicast* de descobrimento. Os pares que recebessem estes segmentos deveriam responder confirmando a existência. A partir daí, cada par deveria enviar periodicamente segmentos direcionados para cada um dos outros pares e todos devem responder.

Pensando mais tarde em outra solução, vimos que a solução acima iria provocar um tráfego maior de segmentos UDP na rede e sua implementação seria um pouco mais complexa.

A solução de fato implementada foi enviar periodicamente segmentos sinalizadores de existência em *broadcast* e *multicast*. Os sinalizadores são suficientes tanto para descobrimento de novos pares quanto para detecção de falhas.

A decisão de enviar segmentos tanto em *broadcast* quando em *multicast* foi tomada por duas razões opostas. Por um lado, a tendência do *broadcast* é deixar de existir, em virtude do *multicast*. Uma das mudanças do IPV4 para o IPV6 é a remoção de *broadcast*. Por outro lado, *multicast* não é permitido por padrão em alguns ambientes. É preciso configurar para que seja permitido. Portanto, atualmente, os dois segmentos são complementares.

2.6 Tolerância a falha de um par

Requisito – O sistema deve ser tolerante a falha de uma máquina, ou seja, os pares ativos devem trocar informações de forma que todos se mantenham sincronizados em relação (i) ao sistema de arquivos, (ii) ao cadastro de usuários e (iii) ao estado do sistema. Para tal, cada arquivo deve ser armazenado em duplicidade na rede, ou seja, a quantidade total de dados armazenados no sistema deve ser igual ao dobro da soma dos tamanhos dos arquivos existentes.

Implementação – Para implementar este requisito, utilizou-se um temporizador que detecta a falha de um par caso um segmento UDP sinalizador não chegue em um determinado intervalo de tempo.

Ao detectar uma falha, cada par remanescente verifica se o seu endereço IP é o menor. Apenas o par de menor endereço IP irá calcular as duplicações e o balanceamento necessário.

Ao calcular os comandos de duplicação e balanceamento, o par de menor endereço IP envia estes comandos aos outros pares, que, por sua vez, atualizam suas próprias tabelas e enviam os arquivos necessários.

2.7 Balanceamento de arquivos

Requisito – O sistema deve promover o balanceamento de arquivos, distribuindo os arquivos entre os pares conectados de forma que cada par possua uma quantidade similar de arquivos, sendo no máximo $\lceil \frac{\#arquivos}{\#pares} \rceil$.

Implementação – Para implementar o balanceamento de arquivos, foi desenvolvido o algoritmo Rodopoulos–Pimenta (seção 1.3), que gera um conjunto de comandos de balanceamento. Os comandos

são processados por cada par, para que as tabelas possam ser atualizadas e os arquivos possam ser enviados e removidos.

3 Protocolo de comunicação

O protocolo de comunicação desenvolvido estabelece dez tipos de mensagem. Assim que um par aceita uma conexão TCP, ele deve receber um *byte*, que indica o tipo de mensagem. De acordo com cada mensagem, outros *bytes* devem ser recebidos e ações devem ser tomadas.

- SYNC – Ao receber esta mensagem, o par deve responder quatro *bytes* com um número identificador único de sessão de acesso, serializar toda a informação atual do sistema de arquivos e enviar pela conexão TCP. Todos os pares que já estão na rede conhecem o número identificador da próxima sessão de acesso. Portanto, qualquer par que recebe uma mensagem de sincronização pode fornecer ao novo par um número identificador válido. Assim que o par que requisitou sincronização recebe seu número identificador, ele começa a enviar sinalizadores com este número. Os pares que recebem estes sinalizadores podem então colocar o novo par em suas tabelas e atualizar o número identificador da próxima sessão de acesso.
- CREATE FOLDER – Ao receber esta mensagem, o par deve receber uma *string* terminada em nulo que contém um caminho de pasta e criar a pasta no seu sistema de arquivos.
- UPDATE FOLDER – Ao receber esta mensagem, o par deve receber uma *string* terminada em nulo que contém um caminho de pasta, outra *string* terminada em nulo que contém o novo nome da pasta e realizar a atualização no seu sistema de arquivos.
- DELETE FOLDER – Ao receber esta mensagem, o par deve receber uma *string* terminada em nulo que contém um caminho de pasta e remover a pasta do seu sistema de arquivos.
- CREATE FILE – Ao receber esta mensagem, o par deve receber uma *string* terminada em nulo que contém um caminho de arquivo, receber quatro *bytes* que indicam o tamanho de uma fila de *bytes*, receber a fila de *bytes*, que contém as informações do arquivo criado, e criar o arquivo no seu sistema de arquivos.
- UPDATE FILE – Ao receber esta mensagem, o par deve receber uma *string* terminada em nulo que contém um caminho de arquivo, outra *string* terminada em nulo que contém o novo nome do arquivo e realizar a atualização no seu sistema de arquivos.
- DELETE FILE – Ao receber esta mensagem, o par deve receber uma *string* terminada em nulo que contém um caminho de arquivo e remover o arquivo do seu sistema de arquivos.
- COMMANDS – Ao receber esta mensagem, o par deve receber quatro *bytes* que indicam o tamanho de uma fila de *bytes*, receber a fila de *bytes*, que contém a serialização de uma lista de comandos, e deserializar a fila de *bytes* chamando o módulo responsável pelo sistema de arquivos. Ao remontar os comandos, o par deve processá-los, enviando arquivos para duplicação/balanceamento, removendo arquivos que não pertecem mais a ele e atualizando seu sistema de arquivos.
- FILE – Ao receber esta mensagem, o par deve receber quatro *bytes* que contêm o número identificador único do arquivo, quatro *bytes* com o tamanho do arquivo e receber o arquivo.
- ACK – Esta mensagem é enviada pelo par que recebe alguma das oito mensagens acima, para sinalizar que as ações necessárias já foram executadas. Como a mensagem SYNC é um pedido de sincronização, não é necessário enviar um ACK no final.

Além das mensagens trocadas por TCP, o protocolo também estabelece que a carga útil dos sinalizadores UDP deve ser quatro *bytes*, que contêm o número identificador único de sessão de acesso, seguidos de uma *string* terminada em nulo que contenha o nome de usuário de um par.