Relatório de TFC
do curso de Licenciatura em Engenharia Informática e de Computadores
(LEIC)

Ano Lectivo 2001/2002

Departamento de Engenharia Informática

N.º da Proposta: 149

Título: Sistema de Ficheiros Distribuído para Dispositivos Móveis com Windows CE

Professor Orientador:

Paulo Ferreira _____

Co-Orientador:

João Garcia _____

Aluno:

Nº 45484, João Pedro Barreto _____

Agradecimentos

Ao Prof. Paulo Ferreira pelo enorme apoio prestado à realização deste trabalho final de curso e pelos preciosos comentários durante a elaboração deste documento.

Ao João Garcia, pela valorosa revisão final deste documento.

Aos meus colegas de sala, pelo excelente acolhimento desde o primeiro dia de trabalho na aconchegada sala 615.

A todos os restantes colegas do Grupo de Sistemas Distribuídos pelo apoio facultado ao longo deste trabalho.

À minha família, por tudo.

Aos meus amigos (vocês sabem quem são) por tudo o que nunca foi dito mas que sempre foi percebido.

Resumo e Palavras Chave

A evolução do poder computacional e da capacidade de memória dos computadores actuais tem vindo a ser acompanhada por uma redução significativa das suas dimensões físicas. A portabilidade crescente que resulta desta evolução tem dado origem a dispositivos com características cada vez mais adequadas à computação móvel. Com a mobilidade veio também a necessidade de tecnologias de rede sem fios que pudessem libertar os novos equipamentos das ligações fixas à rede. Esse tipo de tecnologias já está disponível, sob a forma de protocolos de redes locais sem fios como o IEEE 802.11 ou o *Bluetooth*, o que torna o paradigma da computação móvel numa realidade prometedora.

A realidade actual permite antever um cenário futuro em que o uso destes computadores portáteis com capacidade de comunicação sem fios seja generalizado a uma larga gama de utilizadores, que tira partido destes equipamentos em tarefas que hoje em dia são feitas em *desktop PCs*. Como tal a possibilidade de partilha de dados e o acesso aos mesmos a partir de qualquer localização surge como um objectivo altamente atractivo. Em particular, o presente trabalho visa alcançar esse objectivo através de um sistema de ficheiros distribuído, que permita aos utilizadores e aplicações partilhar e aceder a ficheiros, usando para esse efeito a interface a que estão habituados para aceder ao sistema de ficheiros local.

O universo computacional definido por estes novos computadores móveis é, contudo, bastante diferente daquele que é tipicamente encontrado em redes fixas. As características específicas dos ambientes de computação móvel trazem novos problemas para os quais os sistemas convencionais, orientados para redes fixas, não são soluções eficazes. É necessário, pois, a concepção de sistemas de ficheiros distribuídos que tenham um conjunto de propriedades que os tornem adaptados aos ambientes móveis.

O presente documento analisa e discute os conceitos e metodologias usualmente associados à concepção de sistemas de ficheiros distribuídos genéricos, por forma a confrontá-los com os requisitos inerentes aos novos ambientes de computação móvel. A abordagem seguida pretende considerar dois ambientes móveis distintos: o de nós móveis que usam uma infra-estrutura pré-existente fixa e o de redes formadas exclusivamente por nós móveis que constituem de forma espontânea uma rede *ad-hoc*. Como resultado, define um conjunto de opções arquitecturais e de desenho que devem ser seguidas para atingir as propriedades desejáveis a um sistema adaptado a esses ambientes. Adicionalmente, é proposto um esquema de nomes lógicos do sistema que possui um conjunto de vantagens que o tornam apropriado para um sistema de ficheiros distribuído para redes *ad-hoc*. Finalmente, é descrito e avaliado um sistema realizado com base nas soluções propostas, desenvolvido sobre o sistema operativo Microsoft Windows CE 3.0.

Palavras chave: sistemas de ficheiros distribuídos, computação móvel, redes móveis, redes *ad-hoc*, *caching*, sistemas embebidos, Windows CE.

Índice

Lista de Figuras

Lista de tabelas

O presente relatório pretende definir o conteúdo, objectivos e metodologia utilizados no Trabalho Final de Curso (TFC), de acordo com o que é especificado no Regulamento de Trabalho Final do Curso de Licenciatura em Engenharia Informática e de Computadores.

Tendo em conta os pressupostos do Regime Integrado em que este TFC se enquadrou, as secções seguintes são apresentadas na dupla óptica de TFC e de trabalho preliminar para tese de mestrado.

1 Introdução

Ao mesmo tempo que a lei de Moore [1] continua a verificar-se, com o poder computacional e a capacidade de memória a duplicar em cada 18 meses, os computadores têm evoluído para dimensões físicas cada vez menores. A portabilidade crescente que resulta desta evolução tem dado origem a dispositivos com características cada vez mais adequadas à computação móvel. Consequentemente, com a mobilidade veio também a necessidade de tecnologias de rede sem fios que pudessem libertar os novos equipamentos das ligações fixas à rede. Esse tipo de tecnologias já está disponível, sob a forma de protocolos de redes locais sem fios como o IEEE 802.11 [2] ou o Bluetooth [3], o que torna o paradigma da computação móvel numa realidade prometedora.

A realidade actual permite antever um cenário futuro em que o uso destes computadores portáteis com capacidade de comunicação sem fios seja generalizado a uma larga gama de utilizadores, que tira partido destes equipamentos em tarefas que hoje em dia são feitas em *desktop PCs*. Surge assim a noção de *info-appliances*, aplicações embebidas em computadores portáteis que oferecem muitas das funcionalidades que antes eram encontradas exclusivamente em estações de trabalho fixas. Neste contexto, a possibilidade de acesso a um repositórios de dados partilhado a partir de qualquer localização é obviamente um objectivo de extremo interesse. Em particular, a motivação por detrás deste trabalho traduz-se na possibilidade de permitir um acesso eficaz a essa informação através de um sistema de ficheiros distribuído. Um sistema de ficheiros distribuído surge como um objectivo altamente desejável pela facilidade que concede aos utilizadores e aplicações de partilhar e aceder a dados, utilizando para isso uma interface semelhante à que estão habituados para aceder ao sistema de ficheiros local.

Como exemplo motivador do trabalho que se descreve ao longo deste documento, considere-se a situação de várias pessoas que se encontram numa reunião, tendo cada uma um equipamento portátil com capacidade de comunicação sem fios. O conjunto de equipamentos pertencentes aos participantes na reunião poderia formar, de forma espontânea, uma rede que se manteria durante a duração da reunião. Tal rede permitiria que, por exemplo, um elemento pudesse disponibilizar às restantes pessoas um documento que considerasse relevante para o contexto da reunião, tal como é ilustrado na Figura 1. Para isso bastaria que os restantes elementos da reunião abrissem o ficheiro que estava partilhado no equipamento do respectivo dono, usando para isso uma aplicação tradicional de processamento de texto. Após o fecho da reunião, cada trabalhador pode voltar ao seu posto de trabalho onde, através de uma ligação a um ponto de acesso à rede fixa, sincronizaria os apontamentos tirados durante a reunião com os seus documentos no sistema de ficheiros local.

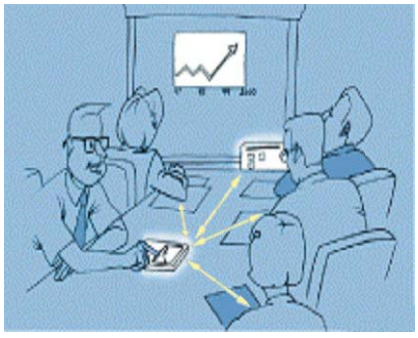

Figura 1: Exemplo de rede *ad-hoc*

1.1 Computação móvel

O universo computacional definido por estes novos computadores móveis é bastante diferente do actual, dominado por *desktop PCs*. O poder computacional de cada equipamento individual é limitado se comparado com um computador tradicional. Os processadores usados são mais lentos, a bateria é limitada, a sua memória disponível varia entre algumas centenas de *kilobytes* a poucos *megabytes* e as suas ligações sem fios à rede têm geralmente uma largura de banda inferior a 1Mbps [4].

Outro aspecto de grande relevância é a heterogeneidade patente no ambiente móvel. Ao contrário dos *desktop PCs*, em que um sistema operativo é proeminente (Microsoft Windows sobre processadores Intel ou compatíveis), os dispositivos móveis são caracterizados por uma variedade muito maior, uma vez que nenhuma marca domina o mercado de forma clara. Neste ambiente, é comum o uso de diversos processadores e sistemas operativos construídos especificamente para cada gama concreta de dispositivos.

Estes computadores de pequena dimensão com capacidades de comunicação sem fios têm um grande potencial para uso em aplicações distribuídas. No entanto, este potencial é actualmente pouco explorado devido à natureza das infra-estruturas de suporte e aplicações existentes.

A infra-estrutura existente é normalmente baseada em recursos típicos dos ambientes fixos: máquinas servidoras (desde servidores WWW, servidores de correio electrónico até servidores de sistemas de ficheiros distribuídos) e uma rede fixa, constituída pelos seus cabos, *hubs*, *switches*, *routers*, etc. Esta infra-estrutura suporta eficazmente arquitecturas baseadas no modelo cliente-servidor.

Os computadores móveis têm sido integrados nesta infra-estrutura como clientes adicionais que utilizam as suas ligações sem fios para terem acesso a uma rede fixa onde são disponibilizados os serviços das máquinas servidoras aí localizadas. Telefones portáteis e *Personal Digital Assistants* (PDA), por exemplo, são geralmente utilizados para enviar e receber mensagens de *sms* ou correio electrónico através de servidores situados na rede fixa.

Contudo, este tipo de solução têm desvantagens claras para o universo específico dos computadores móveis que não permitem aproveitar todo o seu potencial. Considere-se, por exemplo, que um servidor se torna indisponível quando dois PDAs pretendem trocar mensagens entre si. Ou, de modo semelhante, que um dos PDAs se move para além do alcance à rede fixa. Neste caso, a troca de mensagens deixa de ser possível, muito embora os PDAs possam estar mutuamente acessíveis para comunicar directamente entre si.

1.2 Redes *Ad-Hoc*

Uma abordagem alternativa às soluções baseadas na utilização de arquitecturas cliente-servidor com ligação a uma rede fixa consistiria em arquitecturas baseadas em interacções *peer-to-peer* [5] entre cada computador móvel. Deste modo, a existência de uma rede fixa onde estariam localizados os servidores deixaria de ser necessária e os problemas citados atrás deixariam de ocorrer. Usando as suas capacidades de comunicação sem fios, os computadores móveis poderiam detectar a existência de outros pares e ligar-se entre si para formar espontaneamente uma rede. A partir desse momento, seria possível comunicarem e colaborarem entre si sem a necessidade de uma ligação a uma infra-estrutura pré-existente na rede fixa. A estas redes de formação espontânea chamam-se frequentemente Redes *Ad-Hoc* [4] (embora por vezes também se use a expressão *Wireless Personal Area Networks* [4] para o mesmo efeito). Tipicamente, cada utilizador de um dispositivo de computação móvel pertence

a uma rede *ad-hoc* que consiste no conjunto de outros equipamentos que, em cada momento, estão dentro do seu alcance para estabelecer uma ligação sem fios.

As redes *ad-hoc* são ideais em situações em que a instalação de uma infra-estrutura é demasiado cara ou vulnerável, em que a rede é de carácter transitório ou em que a infra-estrutura foi destruída [6]. Entre estas situações encontram-se várias aplicações que vão desde as de cariz militar às de cariz comercial.

1.2.1 Principais Características das Redes *Ad-Hoc*

Entre as características mais relevantes das redes *ad-hoc*, as seguintes diferenciam-nas das redes fixas tradicionais:

- Topologia dinâmica

  Os nós podem mover-se livremente, o que pode provocar que a topologia da rede mude rapidamente em alturas imprevisíveis e de modo aleatório.

- Poder computacional e capacidade de memória dos nós relativamente reduzidos

  Apesar da evolução que se tem verificado nos dispositivos portáteis, o poder computacional e a capacidade de memória do seu universo definido por estes computadores serão sempre inferiores se comparados com os dos computadores fixos.

- Largura de banda limitada e ligações de capacidade variável na mesma rede

  As ligações sem fios continuarão a oferecer uma capacidade significativamente menor que a oferecida pela alternativas com fios. Como consequência, o uso da rede pelas aplicações deverá aproximar ou exceder a capacidade da rede frequentemente.

- Utilização da rede é restringida pela energia disponível nos equipamentos

  Muitos dos nós constituintes de uma rede *ad-hoc* deverão depender de meios de energia limitadas, tais como baterias. Para estes nós, a optimização do uso de energia é um aspecto vital do seu funcionamento.

- Segurança física limitada

  As redes móveis sem fios introduzem um novo conjunto de vulnerabilidades àquelas que já existem nas redes fixas tradicionais. Tipicamente, as redes móveis sem fios são mais vulneráveis a ameaças físicas, pois qualquer dispositivo que se encontre dentro do alcance da rede sem fios pode participar nela, passiva ou activamente. Ataques por escuta de rede, personalização ou negação de serviço são assim mais facilitados. Por outro lado, situações de perda ou roubo de computadores de pequenas dimensões e com elevada mobilidade são bastante prováveis, pelo que os dados neles armazenados passam também a estar ameaçados.

Estas características influenciarão preponderantemente qualquer sistema que se execute sobre uma rede deste tipo. Em particular, o dinamismo da topologia, as limitações de poder computacional e capacidade de memória, assim como a largura de banda reduzida limitarão significativamente o funcionamento de sistemas de ficheiros distribuídos em ambientes *ad-hoc*.

1.3 Objectivos do trabalho

Recapitulando, as características próprias dos ambientes de computação móvel trazem consigo novos desafios, para os quais os sistemas de ficheiros distribuídos existentes orientados para redes fixas não são apropriados.

A solução passa, pois, pela concepção de um sistema que permita efectuar a partilha de ficheiros de um modo que seja adaptado às especificidades próprias dos ambientes de computação móvel. Nomeadamente, deverão ser contempladas opções de arquitectura, esquema de nomes, gestão de *cache* e segurança que contribuam para a eficácia do sistema em ambientes móveis. É esse o objectivo do trabalho que se apresenta neste documento.

1.4 Estrutura do documento

Este documento encontra-se dividido em cinco capítulos. Este primeiro capítulo pretende introduzir e motivar o leitor para o trabalho que se apresentará nos capítulos seguintes. O segundo capítulo descreve os conceitos e a metodologia geralmente associada a sistemas de ficheiros distribuídos, apresentando e discutindo as principais características das soluções existentes. O terceiro capítulo aborda e discute as principais opções de arquitectura e desenho do trabalho. O quarto capítulo descreve a realização do sistema que foi efectuada até à data deste documento. Finalmente, o quinto capítulo expõe as conclusões e apresenta os principais aspectos que deverão ser alvo de trabalho futuro no âmbito deste trabalho final de curso integrado.

2 Trabalho relacionado

Um sistema de ficheiros distribuído (*distributed file system*, ou DFS) permite a um processo aceder a ficheiros situados noutras máquinas, idealmente de uma forma idêntica à que utiliza para o acesso aos ficheiros locais. Numa rede de máquinas que utilizem o mesmo DFS, cada aplicação continua a executar-se apenas na máquina local, mas pode aceder aos ficheiros situados remotamente [7].

Grande parte da informação que as aplicações pretendem partilhar está armazenada de forma persistente num sistema de ficheiros. Com DFSs, essa informação, sob a forma de ficheiros, passa a estar acessível a partir de todas as máquinas clientes do sistema. Um relatório ou uma folha de cálculo podem agora circular entre utilizadores de máquinas diferentes com a mesma facilidade que anteriormente eram acedidos por utilizadores diferentes da mesma máquina. A partilha de informação fica assim muito mais facilitada. A popularidade de DFSs como o NFS, AFS ou CIFS são uma prova de que a sua funcionalidade é apropriada aos requisitos dos utilizadores.

O resto desta secção pretende analisar e comparar as principais decisões de desenho que se encontram durante a concepção de um DFS. A secção 2.1 começa por definir quais as propriedades que são usualmente desejáveis num DFS. A secção 2.2 aborda os aspectos relacionados com o espaço de nomes do sistema. A secção 2.3 expõe as semânticas de partilha que são geralmente oferecidas por um sistema de ficheiros. A secção 2.4 confronta as abordagens cliente-servidor e *peer-to-peer* para a arquitectura de um DFS. A secção 2.5 aborda as principais opções relacionadas com o uso de mecanismos de *caching*. A tolerância a faltas de sistemas deste tipo é discutida na secção 2.6, seguindo-se de uma análise às soluções de sistemas replicados na secção 2.7. As secções 2.8 e 2.9 abordam os aspectos da escalabilidade e segurança dos DFSs. Os aspectos discutidos nestas secções são, finalmente, concretizados na secção 2.10, que descreve alguns sistemas de ficheiros distribuídos cujas características poderão ser relevantes no âmbito deste relatório. Finalmente, na secção 2.11 é feita uma análise comparativa das principais decisões discutida nas secções anteriores, enquadrando-as no contexto das redes móveis e redes *ad-hoc*.

2.1 Propriedades desejáveis

Um aspecto fundamental num sistema de ficheiros distribuído – e uma grande vantagem face a outras eventuais soluções de partilha de informação – é a transparência do acesso a ficheiros remotos. Idealmente, as aplicações deverão poder utilizar um sistema de ficheiros distribuído da mesma forma como acedem ao sistema de ficheiros local. Esta transparência é muitas vezes designada por transparência de rede [8]. Um DFS diz-se possuir esta propriedade se os clientes podem aceder aos seus serviços de ficheiros através das mesmas operações que usam para aceder aos ficheiros locais.

Outro aspecto da transparência de um DFS relaciona-se com a possibilidade dos utilizadores acederem ao DFS a partir de máquinas distintas. Um sistema transparente deverá fornecer ao utilizador o mesmo ambiente, independentemente da máquina que este use para aceder aos serviços. Por exemplo, a árvore de directorias apresentada pelo DFS deverá ser sempre a mesma em qualquer máquina.

O desempenho do DFS é outro factor fundamental da sua transparência. Em geral, a principal medida do desempenho de um sistema de ficheiros é dada pelo tempo necessário a servir os pedidos dos clientes. Num sistema de ficheiros local, este valor é tipicamente dominado pelos tempos de acesso ao disco, face aos reduzidos tempos de processamento e de acesso à memória principal. No caso distribuído, há que adicionar a estes tempos o acréscimo resultante da utilização da rede para serviço dos pedidos. Este acréscimo é constituído pelo tempo necessário para enviar o pedido ao servidor pela rede, assim como o tempo da recepção da respectiva resposta. Em qualquer destas direcções deve ser também considerado o tempo de processamento do protocolo de comunicação. Num DFS óptimo, o seu desempenho deverá ser idêntico ao de um sistema de ficheiros local.

Finalmente, e tal como qualquer sistema distribuído, um DFS deverá possuir, tanto quanto possível, as propriedades de tolerância a faltas e escalabilidade.

Um sistema tolerante a faltas deverá continuar a funcionar, eventualmente de forma degradada, na presença de faltas. No caso de um DFS, deverão ser esperadas, entre outras, faltas na rede de comunicação, nas máquinas intervenientes no sistema ou nos recursos de armazenamento (como discos magnéticos). A degradação resultante deverá ser proporcional às faltas que ocorram [8]. Ou seja, um sistema que veja a sua funcionalidade totalmente interrompida devido a uma pequeno número de faltas não é tolerante a faltas.

Por outro lado, um DFS escalável deverá ser capaz de se adaptar a cargas de serviço crescentes. Em particular, os servidores de um DFS não deverão ficar facilmente saturados perante um número elevado de pedidos.

2.2 Espaço de nomes

Uma característica fundamental de um sistema de ficheiros é o modo como é feito o mapeamento entre os objectos lógicos e físicos do sistema. Os utilizadores lidam com objectos lógicos, geralmente representados por nomes de ficheiros, enquanto que o sistema processa objectos físicos, como blocos de dados numa unidade de armazenamento. Num DFS, esta abstracção é estendida para suportar a propriedade da transparência de rede, citada na secção anterior. Num sistema de ficheiros local, o domínio dos nomes físicos consiste no espaço de endereçamento dentro de um disco. Os objectivos de um DFS obrigam a que esse domínio passe a incluir também uma nova dimensão que indique qual a máquina dentro da rede em cujo disco o ficheiro está armazenado. Um passo adicional nesta abstracção leva à noção de replicação de ficheiros. Neste caso, o mapeamento resulta num conjunto de localizações que representam os objectos físicos onde um ficheiro está replicado.

2.2.1 Transparência e independência de localização

No que toca à transparência do espaço de nomes lógicos de um DFS, duas noções podem ser distinguidas:

Transparência de localização (*location transparency* [8]): verificada quando o nome de um ficheiro não revela ao utilizador qual é o seu local físico de armazenamento;

Independência de localização (*location independence* [8]): verificada quando o nome do ficheiro não precisa de ser alterado quando o respectivo local físico de armazenamento varia.

A independência de localização é uma propriedade mais forte, uma vez que qualquer nome que seja independente da localização é obrigatoriamente transparente da mesma.

A maioria dos DFS actualmente em uso apenas suporta a transparência de localização. Esta propriedade assegura um modo eficaz e conveniente de partilhar ficheiros. Os utilizadores

podem assim aceder a ficheiros remotos como se estes fossem locais, uma vez que não se apercebem da sua localização noutro computador da rede.

Por outro lado, a independência de localização está usualmente associada aos conceitos de mobilidade ou migração de ficheiros. Um DFS que tenha um espaço de nomes lógicos com esta propriedade é capaz de movimentar os ficheiros entre localizações físicas de modo completamente transparente aos utilizadores. Alguns DFSs tiram partido desta funcionalidade para efeitos de desempenho, disponibilidade ou facilidade de administração. Um exemplo concreto é dado pelo OceanStore (secção 2.10.10).

2.2.2 Esquemas de nomes

Em [8] são apresentadas três abordagens principais para a concepção de um esquema de nomes num DFS.

Na abordagem mais simples, os nomes de ficheiros são obtidos por uma combinação do nome da máquina onde o ficheiro está armazenado e o seu nome local. Este esquema assegura assim que os nomes lógicos dos objectos do DFS sejam únicos. Contudo, é claro que os nomes não possuem transparência ou independência de localização. Apesar de tudo, a transparência de rede é conseguida, pois os utilizadores podem utilizar as operações do sistema de ficheiros local para acederem aos objectos do DFS. Um exemplo deste esquema de nomes é dado pelo CIFS [10], cujos nomes – definidos de acordo com a *Universal Naming Convention* (UNC) [11] – têm a seguinte forma:

// nome do computador / directoria partilhada / nome do ficheiro relativo à directoria.

A segunda abordagem oferece meios para os clientes do DFS incluírem directorias remotas dentro do seu espaço de nomes local. Ao procedimento de inclusão de uma directoria, a nomenclatura anglo-saxónica designa usualmente por *mount*. Esta solução permite uma grande versatilidade na estrutura de nomes do DFS. Como tal, a estrutura de nomes visível pode não ser idêntica em todas as máquinas do sistema. Como consequência, este esquema de nomes não garante transparência na mobilidade dos utilizadores. Ou seja, um utilizador poderá encontrar os seus ficheiros em pontos diferentes da árvore de directorias, consoante a máquina que esteja a utilizar para aceder aos serviços do DFS.

De um ponto de vista global, a estrutura de nomes do DFS é constituída por várias árvores, uma por cada máquina, com algumas sub-árvores coincidentes. Estas sub-árvores coincidentes correspondem a directorias que são *mounted* em sistemas de ficheiros locais de outras máquinas. Este esquema é encontrado em vários DFSs actuais, tendo ganho especial popularidade devido à sua aplicação no Sun NFS.

Uma terceira abordagem consiste numa estrutura de nomes global que cubra todos os ficheiros do sistema. Idealmente, esta abordagem alcança uma integração total entre os sistemas de ficheiros locais que componham o DFS, pois é oferecido o mesmo espaço de nomes a todos os clientes.

Na prática, contudo, existem alguns detalhes próprios dos sistemas de ficheiros locais que tornam difícil de atingir o objectivo de um espaço de nomes global. Em concreto, a maioria dos sistemas operativos assume a existência de ficheiros especiais, específicos do hardware de cada máquina, que residem em directorias fixas. Por exemplo, o sistema operativo Unix trata os dispositivos de I/O específicos de cada máquina como ficheiros localizados dentro da directoria */dev*. Esta terceira abordagem é adoptada, embora com algumas variações, por sistemas como o Locus (secção 2.10.1) e AFS (secção 2.10.4), entre outros.

2.3 Semânticas de partilha

Um aspecto importante da caracterização de um sistema de ficheiros, seja ele local ou distribuído, é a semântica de partilha que oferece. A semântica de partilha deverá definir, em particular, quando é que as modificações de dados por um cliente são observadas - se é que o são – pelos restantes clientes. Historicamente, várias semânticas podem ser distinguidas nos sistemas de ficheiros concebidos. As principais semânticas são caracterizadas resumidamente nas sub-secções seguintes: semântica Unix, de sessão, de ficheiros partilhados imutáveis e transaccional. Por fim, a última sub-secção faz uma análise comparativa das semânticas mais frequentes nos DFS actuais.

2.3.1 Semântica UNIX

Esta semântica assegura as seguintes propriedades:

Qualquer operação de leitura reflecte as operações de escrita anteriormente aplicadas ao ficheiro em questão. Em particular, escritas a um ficheiro aberto por um cliente são visíveis a todos os outros clientes que tenham esse ficheiro aberto.

É permitido que vários clientes partilhem o ponteiro de localização dentro de um ficheiro. Consequentemente, o avanço desse ponteiro como resultado de uma operação de leitura ou escrita por um cliente afecta os restantes clientes que partilhem o ponteiro.

Esta semântica obriga geralmente a uma realização que associe um ficheiro a uma imagem física única que serve todos os acessos ao ficheiro. A obrigação de existir um único ponto de acesso aos serviços sobre um ficheiro resulta em contenção e consequente atraso no caso de acessos concorrentes por vários clientes.

Por outro lado, a propriedade da partilha do ponteiro de localização dentro de um ficheiro é uma especificidade do sistema operativo Unix que normalmente só existe em DFSs por razões de compatibilidade. Dado que a sua realização obriga geralmente a esforços consideráveis no desenho dos DFSs e tipicamente traz prejuízos de desempenho, muitos DFSs com esta semântica apenas garantem parcialmente esta propriedade. Locus (secção 2.10.1) e Sprite (secção 2.10.3) estão entre os DFSs que suportam esta semântica.

2.3.2 Semântica de sessão

Considere-se uma sessão como uma série de acessos de leitura ou escrita, delimitados pelas operações de abertura e fecho de um ficheiro.

Escritas num ficheiro aberto são imediatamente visíveis para os clientes que abriram o ficheiro mas são invisíveis para os restantes clientes que tenham o mesmo ficheiro aberto nesse instante.

Quando um ficheiro é fechado, as alterações efectuadas sobre ele serão visíveis para sessões que sejam iniciadas posteriormente sobre o mesmo ficheiro. Clientes que já tenham o mesmo ficheiro aberto não observam qualquer modificação.

Em contraste com a semântica Unix, esta semântica permite que cada ficheiro aberto esteja temporariamente associado com várias imagens físicas simultaneamente. Consequentemente, múltiplos clientes podem ler ou escrever sobre as imagens sem que sofram atrasos resultantes de contenção.

2.3.3 Semântica de ficheiros partilhados imutáveis

Considere-se a distinção de ficheiros partilhados e ficheiros não partilhados num sistema de ficheiros. Assuma-se também a possibilidade de alteração desse estado por parte do utilizador.

Um ficheiro que seja declarado como partilhado não pode ser modificado.

Esta semântica, limita consideravelmente as potencialidades de partilha de ficheiros, pois a possibilidade de escritas concorrentes sobre os ficheiros é eliminada. Em contrapartida, simplifica grandemente a realização de um sistema de ficheiros, dado que qualquer partilha que ocorra será sempre em modo de leitura apenas (*read-only*).

2.3.4 Semântica transaccional

Os efeitos de múltiplas sessões sobre um ficheiro (delimitadas pela abertura e fecho do mesmo) são equivalentes aos que obteriam se a execução dessas sessões fosse serializada por uma determinada ordem.

Basicamente, esta semântica identifica uma sessão como uma transacção. A sua realização pode ser conseguida pelo uso de trincos lógicos que são automaticamente aplicados a um ficheiro aquando da sua abertura e libertados no seu fecho.

2.3.5 Resumo

As semânticas Unix e de sessão dominam os DFSs disponíveis actualmente. A escolha entre cada uma destas semânticas influencia significativamente as decisões de desenho de um DFS que serão discutidas nas próximas secções. Como consequência, obtém-se as seguintes vantagens:

- uma realização mais simplificada (e com eventuais melhorias a nível de desempenho, tolerância a faltas ou escalabilidade), no caso da semântica de sessão; ou
- uma consistência mais fortalecida (ver secção 2.7.1), no caso da semântica Unix.

2.4 Arquitecturas

De um modo genérico, as arquitecturas dos DFS distinguem-se entre cliente-servidor e *peer-to-peer*. Num sistema com uma arquitectura cliente-servidor, todas as alterações têm de ser efectuadas no servidor antes de serem reflectidas nos restantes clientes do sistema. As arquitecturas cliente-servidor simplificam os sistemas, mas podem implicar uma propagação lenta das alterações efectuadas sobre os ficheiros e o seu funcionamento é vulnerável a falhas no servidor.

Em contraste, os sistemas *peer-to-peer* permitem que qualquer *peer* propague as suas alterações para o *peer* que contenha os ficheiros a modificar, em vez de estar dependente da disponibilidade de um servidor central. Basta para isso que ambos os *peers* estejam mutuamente acessíveis. A realização destas arquitecturas é, contudo, normalmente mais complexa que no caso cliente-servidor.

As arquitecturas cliente-servidor, apropriadas para a grande maioria das aplicações sobre redes fixas, também se podem considerar adequadas para casos de computadores móveis que se desliguem de uma rede central fixa e assim continuem até regressarem. Um exemplo deste caso é o de um trabalhador que desligue o seu computador portátil da rede central da empresa para ir para casa ou ir em viagem e só o volte a ligar quando retornar. Porém, em situações cujo padrão de conectividade não seja tão bem definido nem previsível, uma abordagem *peer-to-peer* poderá ser interessante. Neste caso, se algumas máquinas se tornassem imprevisivelmente inacessíveis, os restantes *peers* poderiam continuar a interagir, pois não necessitavam da presença de algum servidor central. Um exemplo claro de situações cujo padrão de conectividade é geralmente imprevisível é dado pelo ambiente das redes *ad-hoc*.

Na discussão que se segue nas próximas secções é usada uma nomenclatura associada às soluções cliente-servidor. Deve-se referir, contudo, que os temas analisados são também aplicáveis a sistemas *peer-to-peer*.

2.5 *Caching*

Face aos custos de desempenho que resultam da utilização da rede para satisfazer os pedidos de um DFS, praticamente todos os sistemas actuais usam mecanismos de *caching*. Quando os acessos do DFS exibem alguma localidade referencial, o desempenho do sistema pode melhorar consideravelmente com o uso destes mecanismos. Como será mostrado nas secções seguintes, factores como tolerância a faltas ou escalabilidade também podem obter ganhos com o uso da *cache*.

Primeiro que tudo, consideremos o funcionamento de um DFS sem *cache*. Sempre que ocorre um acesso (de leitura ou escrita) por parte de um cliente, esse acesso resultará em uma ou mais mensagens enviadas pela rede ao servidor que contenha o ficheiro em causa. Essas mensagens são processadas pelo servidor e a sua resposta é enviada de volta. Ou seja, cada acesso a um ficheiro obriga a que o servidor respectivo o sirva e introduz tráfego na rede.

Em contraste, um DFS que utilize o mecanismo de *caching* terá um comportamento diferente. Quando ocorre um acesso, se os dados necessários para o satisfazer não estiverem presentes localmente, serão copiados pela rede, do servidor para o cliente. Usualmente, a quantidade de dados copiada é muito maior que os dados que são de facto necessários para satisfazer o acesso. Finalmente, os acessos são efectuados sobre a cópia armazenada localmente, em *cache*. Os ficheiros continuam a ser associados a uma cópia primária situada no servidor e de onde foram copiados para *cache*. Contudo, porções dessa cópia primária estão espalhada pelas *cache*s dos clientes do DFS. Quando uma *cache* é modificada, as alterações terão de ser propagadas para a cópia primária e, dependendo da semântica de partilha adoptada (ver secção 2.3), também para outras *cache*s.

No desenho do mecanismo de *caching* de um DFS, as seguintes decisões são essenciais:

- a granularidade dos dados em *cache*;
- o local físico onde a *cache* ficará armazenada no cliente;
- como as modificações de dados em *cache* são propagadas;
- como determinar se os dados em *cache* estão consistentes.

2.5.1 Granularidade

No que se refere à granularidade dos dados em *cache*, as abordagens existentes variam desde o armazenamento em *cache* de ficheiros completos a blocos de ficheiros. Nos DFSs que guardam blocos, a sua dimensão corresponde normalmente à unidade de transferência entre a memória secundária e os *buffers* da memória primária. O aumento da dimensão da unidade de dados em *cache* implica geralmente um aumento da probabilidade do próximo acesso utilizar os dados existentes em *cache*. Inversamente, esse aumento da granularidade dos dados em *cache* exige tempos superiores de transferência dos dados para a *cache*, assim como aumenta a probabilidade de ocorrerem problemas de consistência (por falsa partilha [7]).

Deve também ser tomado em consideração o facto de alguns clientes não disporem de recursos de memória suficientes para *cache*s de grandes dimensões. Por exemplo, os ambientes móveis são usualmente caracterizados por dispositivos portáteis de memória

reduzida. Nestes ambientes, o *caching* de ficheiros completos de grande dimensão poderá ser mesmo inviável.

À partida, o *caching* de ficheiros inteiros afigura-se como a solução ideal para semânticas de sessão. Após o carregamento de um ficheiro em *cache*, os acessos de leitura e escrita podem ser servidos localmente. De acordo com a semântica de sessão, a manutenção da consistência dos ficheiros apenas poderá obrigar à propagação das escritas no momento do seu fecho. O carregamento de ficheiros de grandes dimensões pode, no entanto, ser uma limitação importante, especialmente quando apenas poucos blocos são modificados. Esta limitação ganha maior peso quando se consideram redes com larguras de banda reduzida, característica típica de redes móveis. Uma variante por vezes utilizada consiste em carregar apenas partes dos ficheiros. Concretamente, versões recentes do AFS guardam porções de 64kb em *cache* [8]. A semântica de sessão obriga a que os acessos sejam feitos sobre a imagem do ficheiro completo tal como esta existia no momento da abertura do ficheiro. Deste modo, a utilização de imagens parciais do ficheiro traz consigo uma complexidade adicional que é necessária para assegurar que todas as partes acedidas numa sessão correspondem à mesma versão inicial do ficheiro aberto.

Por outro lado, uma semântica Unix adequa-se a blocos de ficheiros. A dimensão destes blocos deverá ser criteriosamente escolhida, tendo em conta factores como a dimensão dos datagramas utilizados pelo protocolo de comunicação em rede ou da unidade de transferência entre o disco e os buffers da memória primária. Por outro lado, o padrão de acessos expectável deverá também ser analisado, de modo a escolher dimensões de blocos que minimizem a frequência de situações de falsa partilha.

2.5.2 Localização

A localização dos dados da *cache* costuma adoptar uma de duas soluções: em memória primária ou em disco. A primeira solução tem como principal vantagem o reduzido tempo de acesso. Por outro lado, as *cache*s armazenadas em memória secundária podem tirar partido de alguns aspectos significativos. Entre eles, a autonomia dos discos, em contraste com a volatilidade da maioria das memórias primárias; e a sua dimensão, que tipicamente é superior à da memória primária em duas ordens de grandeza.

2.5.3 Política de modificação

Como resultado de acessos de escrita, a *cache* poderá ficar com blocos modificados pelo cliente (ditos sujos). Nesta situação, os blocos sujos têm de ser propagados para a cópia primária, localizada no servidor. A estratégia adoptada para a propagação dos blocos sujos é um importante factor do desempenho e da fiabilidade de qualquer DFS.

Uma primeira solução consiste em propagar imediatamente e de forma síncrona as modificações causadas por acessos de escrita. Esta solução, designada por *write-through*, prima pela simplicidade e tem a vantagem de ser uma política de modificação fiável. Se ocorrer uma falha no cliente, as modificações por si efectuadas não se perdem. No entanto, esta abordagem esconde as eventuais vantagens do mecanismo de *caching* no caso de escritas, pois todas as escritas implicam que o cliente as propague ao servidor.

Uma solução alternativa passa por propagar assincronamente os blocos sujos com um determinado atraso face ao momento em que estes foram modificados em *cache*. Deste modo, os acessos de escrita são completados mais rapidamente, uma vez que a sua propagação para o servidor só é feita posteriormente de forma assíncrona. Outra vantagem prende-se com situações em que um bloco é apagado ou é re-modificado antes de ser propagado para o servidor. No primeiro caso, o cliente pode evitar a propagação desnecessária do bloco

apagado; no segundo, o bloco sujo resultante das múltiplas modificações apenas tem de ser enviado para o servidor uma vez. Contudo, em caso de falha no cliente, as modificações que ainda não tivessem sido propagadas podem perder-se, o que é uma desvantagem em termos da fiabilidade desta política.

Uma estratégia para a propagação à posteriori dos blocos sujos consiste em fazê-lo quando estes são expulsos da *cache* ou após um intervalo de tempo pré-fixado sem que tenham sofrido modificações. Esta é a política seguida pelo Sprite (ver secção 2.10.3). Uma alternativa consiste em propagar os blocos sujos quando o ficheiro respectivo é fechado, sendo designada por *write-on-close*.

Relacionando com as semânticas de partilha, existe uma clara adequação da abordagem *write-on-close* à semântica de sessão. De igual modo, a solução de *write-through* adapta-se facilmente aos requisitos de consistência da semântica Unix.

2.5.4 Validação

Antes de utilizar um bloco de dados em *cache*, o cliente precisa de saber se esses dados estão consistentes. Ou seja, é necessário assegurar que os dados em *cache* só são acedidos se reflectirem os dados da cópia primária no servidor. A validação da *cache* pretende resolver este problema. Genericamente, as soluções podem dividir-se em duas categorias: validação iniciada pelo cliente ou pelo servidor.

Na validação iniciada pelo cliente, é este que verifica junto do servidor se os dados locais em *cache* estão consistentes com a cópia primária no servidor. O momento em que esta verificação é efectuada está directamente relacionado com a semântica de partilha que se pretende suportar. Por exemplo, a verificação pode ser feita antes de cada acesso ou antes do primeiro acesso a um bloco de um ficheiro aberto. Esta solução tem a grande desvantagem de, no caso de blocos em *cache* que estejam consistentes, serem sempre feitos acessos desnecessários ao servidor.

Por outro lado, a validação iniciada pelo servidor obriga o servidor a manter em estado informação sobre quais os blocos (ou ficheiros completos) que cada cliente contém em *cache*. Quando o servidor detectar uma potencial inconsistência de um bloco ou ficheiro na *cache* de algum cliente, é responsável por notificá-lo da invalidação desse bloco ou ficheiro. Esta abordagem elimina a necessidade da troca de mensagens de validação de blocos no caso de ficheiros acedidos apenas por um cliente, ou seja, de forma não concorrente. Dado que a quantidade de acessos concorrentes a ficheiros é geralmente muito reduzida [7], esta solução permite eliminar uma porção considerável do tráfego de rede que a validação iniciada pelo cliente produzia. Esta solução tem, no entanto, a desvantagem de obrigar o servidor a manter estado, o que terá consequências na tolerância a faltas, conforme será mostrado na secção 2.6.1.

Concluindo, a decisão entre validação iniciada pelo cliente ou pelo servidor implica:

- acessos mais demorados e maior carga do servidor, na primeira opção; e
- obrigatoriedade de manutenção de estado no servidor, na segunda opção.

2.6 Tolerância a faltas

Nas próximas secções são analisados alguns factores preponderantes na tolerância a faltas exibida por um DFS.

2.6.1 Serviços *stateful* ou *stateless*

Um servidor pode ser considerado como *stateful* ou *stateless*. O primeiro caso ocorre quando o servidor precisa de guardar informação sobre os seus clientes ao longo de vários pedidos. De modo inverso, um servidor *stateless* não guarda qualquer informação sobre um cliente após ter servido o seu pedido. Esta distinção quanto ao estado que os servidores mantêm entre pedidos de clientes têm consequências evidentes na sua tolerância a faltas.

Um servidor *stateful* perde todo o seu estado volátil em caso de uma falha. Para que o sistema recupere correctamente, é necessário proceder à restauração do estado perdido, geralmente baseada num protocolo de diálogo com os clientes. Uma alternativa mais simples, mas também menos interessante, consiste em abortar as operações que estavam em curso no momento da falha. Do lado do cliente, uma falha também pode prejudicar o servidor, já que os recursos de memória alocados para guardar estado relativo a esse cliente deixam de ser utilizados. O servidor deverá ter capacidade de tomar conhecimento dessas situações, de modo a poder recuperar o espaço em memória ocupado por estado obsoleto relativo a clientes faltosos.

Claramente, um servidor *stateless* evita todos os problemas referidos no parágrafo anterior. Como não mantém estado entre pedidos, uma falha apenas o obriga a reiniciar a sua execução, após a qual ficará imediatamente pronto para servir qualquer pedido. Do lado dos clientes, o servidor apenas parecerá mais lento durante o período entre a detecção da falta e a sua reinicialização. Durante este período, os clientes apenas deverão repetir os seus pedidos até que o servidor fique disponível e retorne as respectivas respostas. Para que os pedidos possam ser repetidos, as operações do DFS deverão ser idempotentes, ou seja, o seu efeito e resultado deverá ser o mesmo se executadas consecutivamente.

Contudo, a simplicidade da abordagem *stateless* impõe custos de desempenho no sistema. O facto de não existir informação mantida entre pedidos no servidor inibe geralmente soluções que acelerem o processamento dos pedidos. Por exemplo, uma solução *stateful* pode tirar partido da abertura de um ficheiro para guardar em memória primária alguma informação sobre o ficheiro que permita acelerar acessos posteriores. A essa informação pode ser associado um identificador que é indicado pelo cliente nos seus pedidos de acessos sobre o ficheiro. No caso *stateless*, esta optimização não é possível. Por outro lado, a ausência de estado no servidor *stateless* obriga a que todos os pedidos sejam auto-contidos, ou seja, contenham toda a informação necessária para serem servidos. Em contraste, um servidor *stateful* pode manter alguns dados acessíveis na sua memória, tais como a localização física do ficheiro ou o ponteiro actual de leitura ou escrita, evitando que estes sejam recebidos ou calculados para todos os pedidos.

Em resumo, a simplicidade de fornecer um serviço tolerante a faltas com uma abordagem *stateless* tem o preço de obrigar a mensagens de pedidos mais longas e um processamento mais lento dos mesmos.

Mais uma vez, a adopção de uma destas abordagens está usualmente ligada a outras decisões do desenho do DFS. Nomeadamente, a validação de *cache* iniciada do lado do servidor e o uso de protocolos orientados à ligação são apropriados para servidores *stateful*, dada a sua necessidade de manter estado. De modo semelhante, a validação iniciada do lado do cliente e o uso de protocolos não orientados à ligação são apropriados a soluções *stateless*.

2.6.2 Disponibilidade e durabilidade

No contexto da tolerância a faltas, podem distinguir-se duas propriedades fundamentais dos ficheiros de um DFS: a disponibilidade e a durabilidade. Ambos os conceitos são muitas vezes confundidos, provavelmente devido a terem soluções parecidas.

A disponibilidade de um ficheiro refere-se à probabilidade de, num determinado momento, ele poder ser acedido por um cliente do sistema. Um DFS com uma elevada disponibilidade deverá permitir que, mesmo perante falhas de algumas máquinas (incluindo servidores), os clientes possam prosseguir a sua utilização regular dos ficheiros do sistema.

Ortogonalmente, a durabilidade de um ficheiro está associada à probabilidade de um ficheiro sobreviver a faltas do sistema. Ou seja, um DFS com elevada durabilidade deverá garantir que, em caso de falta de alguns componentes do sistema – ou mesmo a sua destruição – os ficheiros possam ser recuperados sem quaisquer perdas. Esta propriedade é por vezes também referida por robustez de um ficheiro. Note-se que o facto de um ficheiro sobreviver a uma situação faltosa não implica que esteja disponível nesse momento. Daqui se depreende a distinção entre durabilidade e disponibilidade de um ficheiro.

As soluções para o aumento da disponibilidade e durabilidade de um DFS passam frequentemente pela replicação dos objectos do sistema, como será descrito na secção seguinte.

2.7 Sistemas replicados

Tipicamente, a disponibilidade de um DFS é melhorada com a replicação dos dados partilhados. Isto porque se permite que as aplicações possam continuar a funcionar sobre os dados de uma réplica, mesmo quando as restantes réplicas se tornem temporariamente inacessíveis.

Por outro lado, se o sistema replicado assegurar que as réplicas são mantidas consistentemente, a perda de uma réplica não implica que as alterações que foram efectuadas sobre a mesma sejam perdidas. A redundância introduzida pela replicação permite, nestes casos, que um ficheiro replicado sobreviva. Deste modo, consegue-se garantir uma maior durabilidade dos dados face a falhas ou ataques ao sistema.

Finalmente, a replicação não contribui apenas para a melhoria da tolerância a faltas do DFS. Se, para servir os pedidos dos clientes, o sistema suportar a selecção da réplica mais próxima, o desempenho é também melhorado.

O principal desafio de um sistema replicado é a manutenção da consistência das suas réplicas. Considere-se um sistema distribuído com várias réplicas do mesmo objecto partilhado. Ao longo do tempo, poderão ocorrer várias tentativas de alteração concorrentes das várias réplicas. O problema da consistência entre as réplicas obriga à aplicação de uma política de consistência. O objectivo de uma política de consistência é garantir que quaisquer acessos concorrentes ao mesmo objecto partilhado sejam, eventualmente no futuro, observadas pela mesma ordem por todas as réplicas do sistema. Assim se assegura a consistência entre réplicas.

2.7.1 Políticas de consistência pessimistas vs. optimistas

De um modo geral, as políticas de consistência seguem uma de duas abordagens: pessimista ou optimista [12]. As consequências da utilização de cada uma destas categorias de política sobre o desempenho, a consistência entre réplicas e a disponibilidade são bem conhecidas. Políticas pessimistas permitem uma consistência forte entre réplicas, a custo de desempenho e

disponibilidade. Políticas optimistas fornecem uma consistência relaxada mas requerem menor comunicação, obtendo melhor desempenho e disponibilidade.

As realizações usuais de políticas pessimistas proíbem quaisquer alterações concorrentes às réplicas. Ou seja, qualquer tentativa de acesso de escrita sobre uma réplica implica a alteração do objecto original, tipicamente remoto. O objecto original sobre o qual todas as alterações têm de ser aplicadas neste tipo de modelos é por vezes referido como cópia primária. Esta aproximação tem duas grandes desvantagens. Primeiramente, o acesso à cópia primária pode tornar-se facilmente um ponto de congestão no caso do objecto ser alvo de alterações frequentes. Por outro lado, nenhumas alterações podem ser concretizadas sem que se contacte a cópia primária. Do ponto de vista de um ambiente distribuído, este aspecto pode implicar a interrupção do processamento normal das aplicações durante situações de fraca ou nenhuma conectividade à cópia primária. Embora este tipo de estratégia seja apropriado ao universo dos sistemas sobre redes fixas, é totalmente desaconselhado aos ambientes móveis e, em particular ao caso das redes *ad-hoc*, em que situações de conectividade fraca ou nula entre as réplicas ocorrem frequentemente.

No outro extremo, os sistemas com políticas optimistas tipicamente permitem que qualquer máquina que contenha uma réplica a altere localmente, sem necessidade de consultar previamente outras réplicas. As políticas optimistas minimizam os requisitos de largura de banda e de conectividade para os acessos de escrita aos objectos partilhados. Obviamente, sistemas de replicação optimista podem permitir que acessos de escrita conflituosos sejam aplicados a réplicas distintas. No entanto, a experiência obtida no uso de sistemas de ficheiros distribuídos mostra que este tipo de conflitos é raro e geralmente de fácil resolução.

Uma alternativa a estes tipos extremos de políticas de consistência é descrita em [13]. Segundo os autores, existe um espectro entre consistência forte (assegurada por políticas pessimistas) e relaxada (fornecida por políticas optimistas) que é semanticamente adequado a uma vasta gama de aplicações. Alguns resultados do artigo mostram que várias aplicações poderão ter vantagens por utilizarem um modelo de consistência intermédio.

2.8 Escalabilidade

A escalabilidade de um DFS determina quantos clientes e servidores o sistema consegue suportar de forma eficiente [7].

Um factor fortemente influente da escalabilidade é a maneira como a carga no servidor aumenta com o número de clientes. Em geral, arquitecturas baseadas em recursos e unidades de controlo centralizados devem ser evitados em sistemas cuja escalabilidade seja uma propriedade essencial. A centralização pode ser considerada como a origem de assimetrias funcionais [8] no sistema. A alternativa ideal passa, portanto, por conceber uma arquitectura que seja funcionalmente simétrica. Ou seja, todas as máquinas que constituintes do sistema deverão ter papéis semelhantes na operação do mesmo, e consequentemente, o mesmo nível de autonomia. Esta alternativa corresponde ao paradigma das arquitecturas *peer-to-peer*, referido na secção 2.4.

No entanto, a simetria funcional do sistema é geralmente um objectivo inatingível. Por exemplo, muitos sistemas baseiam-se na existência de um conjunto restrito de máquinas servidoras sobre as quais uma entidade central de administração possa ter controlo. Esta exigência torna inviável uma estratégia *peer-to-peer* global, pois obriga à existência de servidores com responsabilidades diferenciadas no DFS. A autonomia e simetria funcional devem, apesar de tudo, ser objectivos importantes a considerar no desenho de um DFS. A

utilização de mecanismos intensivos de *caching* ou de semânticas de partilha relaxadas são exemplos de medidas que contribuem para esse objectivo.

2.9 Segurança

A segurança de um DFS pode ser decomposta nos seguintes principais problemas: autenticação de utilizadores, controlo de acesso, privacidade e integridade dos dados.

As soluções iniciais de DFSs baseavam-se geralmente na assunção de uma rede de comunicação segura e de utilizadores bem comportados. Um exemplo é dado pelo caso do NFS, cuja configuração mais usual se baseia numa autenticação feita pelo envio, em claro, dos identificadores de utilizador e grupo (*uid* e *gid*, respectivamente) do utilizador [7]. No entanto, este mecanismo é vulnerável a ataques, por exemplo, com mensagens forjadas que contenham *uids* e *gids* de outros utilizadores do DFS.

A evolução dos DFSs tem vindo, no entanto, a reforçar a importância dos aspectos de segurança associados a estes sistemas. Muitos dos sistemas começam a incluir mecanismos de segurança que permitem a sua operação em redes inseguras e perante utilizadores mal intencionados. Além disto, alguns sistemas consideram mesmo modelos de confiança em que também a administração dos servidores e dos seus recursos de armazenamento deixa de ser confiável. Um exemplo de um DFS que assume este último modelo de confiança é o OceanStore (secção 2.10.10).

Os problemas da autenticação e do controlo de acesso já existem nos sistemas de ficheiros locais de sistemas multi-utilizador. No entanto, no caso distribuído, estes problemas ganham uma complexidade acrescida, sendo necessária a aplicação de outros mecanismos de segurança.

No caso da autenticação, os sistemas actuais incorporam normalmente mecanismos baseados em chave privada. Cada mensagem contém um autenticador cifrado com uma chave de sessão, anteriormente negociada com um servidor de autenticação, que assegura a autenticidade dos interlocutores. O AFS (secção 2.10.4) foi um dos primeiros sistemas comerciais a usar autenticação *Kerberos* [14] para este efeito.

O controlo de acesso é idêntico ao dos sistemas de ficheiros locais. No entanto, devido à sua maior escala, são por vezes usados mecanismos mais sofisticados, tais como listas de controlo de acesso (ACLs) [7].

A privacidade dos dados pode ser analisada de dois modos: na comunicação pela rede insegura ou no armazenamento num servidor de administração não confiável. Qualquer uma destas vertentes pode ser solucionada pelo uso de cifra dos conteúdos dos ficheiros. No primeiro caso, os dados podem ser cifrados antes de serem enviados para a rede com a chave de sessão negociada entre cliente e servidor durante a autenticação, sendo decifrados quando chegam ao destino. Uma solução alternativa consiste em cifrar o conteúdo dos ficheiros do lado do cliente usando uma chave fornecida pelo utilizador. Deste modo, os dados seriam enviados e armazenados do lado do servidor já cifrados. Esta solução visa resolver tanto os problemas de uma rede insegura como um servidor não confiável. No entanto, não é transparente para o utilizador, que tem de memorizar e especificar uma chave para aceder aos ficheiros. Por outro lado, o facto dos dados permanecerem cifrados do lado do servidor poderá ser uma limitação na eficácia de alguns mecanismos do DFS. Um exemplo particular prende-se com os mecanismos de detecção e resolução de conflitos no caso do Bayou (secção 2.10.9), que podem perder a sua eficácia perante conteúdos cifrados das réplicas. O sistema

OceanStore (secção 2.10.10) endereça este problema limitando o universo de predicados e acções aos que podem ser aplicados directamente sobre os dados cifrados.

2.10 Sistemas de ficheiros distribuídos de referência

Nas sub-secções seguintes são apresentados alguns dos sistemas de ficheiros distribuídos de maior importância nos sistemas computacionais de hoje.

2.10.1 Locus

O projecto Locus [15] teve como objectivo o desenvolvimento de um sistema operativo distribuído de raiz. Uma componente fundamental deste sistema é o seu DFS, cujas principais características são delineadas de seguida.

O sistema de ficheiros do Locus oferece um espaço de nomes totalmente transparente em relação à localização dos seus objectos. Aos utilizadores e aplicações é apresentada uma única árvore global de nomes. O sistema suporta a semântica Unix com grande fidelidade, assegurando inclusive as propriedades de partilha de ponteiros de localização dentro de um ficheiro aberto (ver secção 2.3.1). Para aplicações com requisitos mais exigentes de consistência, é também disponibilizada a possibilidade de aplicar trincos lógicos aos ficheiros.

O Locus faz uso da replicação de ficheiros para efeitos de disponibilidade dos ficheiros para acessos de leitura. No caso das escritas, é adoptada uma estratégia de cópia primária, que obviamente não traz benefícios à disponibilidade para escrita. A replicação é transparente aos utilizadores. Para tal, é incluída no sistema uma entidade adicional, *Current Synchronization Site* (CSS), que se dedica às funções de mapeamento entre um nome lógico e a respectiva réplica.

A tolerância a faltas resultantes da separação da rede em duas ou mais partições de rede é abordada pelo sistema, fazendo uso do CSS. Nestas situações, desde que exista uma réplica de um ficheiro numa partição da rede, esse ficheiro pode ser acedido para leitura pelos clientes dessa partição. O sistema só assegura, no entanto, que os dados obtidos sejam os mais recentes no âmbito restrito dessa partição. Isto porque a partição de rede que contenha a cópia primária do ficheiro continuará a servir também as escritas a esse ficheiro. Quando partições de rede se voltam a ligar, existem mecanismos automáticos de reconciliamento que tratam de actualizar as réplicas antigas.

No entanto, a inclusão do CSS tem consequências negativas tanto na tolerância a faltas como na escalabilidade do DFS. Em relação à primeira, o CSS adiciona um ponto extra de falha, uma vez que a abertura de um ficheiro implica agora que tanto o CSS como o servidor estejam disponíveis. Por outro lado, o CSS pode tornar-se um ponto de congestionamento para grupos de clientes elevados, o que se reflecte numa fraca escalabilidade do Locus.

2.10.2 Sun Network File System

O *Network File System* (NFS) [16], desenvolvido pela Sun Microsystems desde 1984, é um dos sistemas de ficheiros distribuídos conceptualmente mais divulgados [7]. O protocolo entre clientes e servidores deste DFS está totalmente descrito em Sun RPC [7], tendo como objectivo a sua utilização por máquinas, sistemas operativos ou arquitecturas de rede heterogéneas [8].

O sistema considera os nós da rede como um conjunto de máquinas com sistemas de ficheiros independentes. A partilha de ficheiros é permitida entre qualquer par de máquinas. Ou seja, qualquer máquina pode ser simultaneamente servidor dos seus ficheiros locais e cliente dos

ficheiros exportados pelas outras, não sendo obrigatória a existência de um servidor dedicado do sistema. Para que uma directoria remota passe a estar acessível numa máquina cliente, o utilizador dessa máquina precisa de efectuar uma operação de *mount* (ver secção 2.2.2). Após efectuar as operações de *mount*, os utilizadores podem aceder aos ficheiros transparentemente em relação à sua localização, como se a directoria remota fosse local. Da versatilidade oferecida pelo NFS resulta que a estrutura de nomes que cada cliente observa pode ser diferente de máquina para máquina. Consequentemente, não há, no NFS, uma noção de sistema de ficheiros partilhado global, tal como acontece com o Locus.

O NFS efectua o *caching* dos blocos dos ficheiros remotos para efeitos de desempenho. No entanto, praticamente todas as operações do sistema[1] implicam a chamada de funções remotas no servidor. Nomeadamente, as operações de leitura e escrita são sempre precedidas de acessos ao servidor para verificar a consistência dos blocos em *cache*. O NFS utiliza uma política de modificação de escrita atrasada, mesmo quando um ficheiro é aberto de forma concorrente por dois clientes. Por esta razão, a semântica de partilha do NFS não cumpre os requisitos da semântica Unix. Por outro lado, esta semântica também não se enquadra em nenhuma das semânticas usuais, apresentadas na secção 2.3.

Toda a responsabilidade de configuração do espaço de nomes, manutenção de estado sobre ficheiros abertos e gestão da *cache* é dos clientes. Os servidores não mantêm estado em memória volátil, o que obriga a que muitas das operações, nomeadamente as leituras e escritas, são idempotentes. Uma das principais vantagens do NFS advém desta característica, pelo que a sua recuperação após uma falha se limita à sua reinicialização, sem necessidade de nenhum protocolo especial de recuperação.

2.10.3 Sprite

O projecto Sprite [17] consiste num sistema operativo distribuído experimental. A sua concepção foi orientada para existência de máquinas clientes dotadas de memórias primárias de grande dimensão, potencialmente sem recursos de armazenamento secundários. Com efeito, o desenho do sistema assume que os clientes conseguirão aceder aos dados na *cache* remota dos servidores de um modo mais rápido do que se o fizessem a partir de um disco local. Esta hipótese baseia-se em ambientes de redes de alto desempenho, como por exemplo redes locais *switched*. Como reflexo desta decisão, o sistema faz uso extensivo da *cache* dos clientes, localizada em memória primária. Aos utilizadores é apresentada uma árvore única de objectos do DFS, que é acessível de forma transparente quanto à sua localização na rede.

A *cache* é organizada em blocos de ficheiros, tipicamente de dimensão de 4kb. Para o tratamento dos blocos modificados em *cache* é utilizada uma política de escrita atrasada. Concretamente, um bloco sujo só é propagado para o servidor quando passarem 30 segundos desde a sua última modificação ou quando for expulso da *cache*. Dado que este DFS disponibiliza uma semântica Unix, a política de modificação de escrita atrasada pode gerar estados inconsistentes quando um ficheiro é acedido para escrita por dois ou mais clientes. Quando esta situação é detectada, o *caching* é desactivado para o ficheiro em causa e eventuais acessos terão de ser feitos passando pelo servidor. Deste modo é garantida a correcção da semântica de partilha do sistema. No entanto, a detecção de situações de acessos concorrentes obriga à manutenção de estado do lado do servidor. Como custo da manutenção desse estado, uma falha do servidor implica o cancelamento de todos as operações sobre ficheiros abertos no servidor.

[1] Com a excepção das operações de abertura e fecho de ficheiros, principalmente.

Outra característica interessante do sistema é o uso de *prefix tables*, tabelas utilizadas para efectuar o mapeamento entre um nome lógico de ficheiro e a sua localização física no DFS. Estas estruturas têm a propriedade de permitir uma forma eficiente de mapeamento de nomes de ficheiros. Este permite, na maioria dos casos, a obtenção imediata da referência física do ficheiro, sem que seja necessária o processamento individual de cada componente (tipicamente directorias) do nome do ficheiro. A manutenção das *prefix tables* é orientada para ambientes de redes locais, sendo baseada em mensagens por difusão para todos os servidores.

2.10.4 Andrew File System

O Andrew File System (AFS) [18] foi inicialmente desenvolvido em Carnegie Mellon University, a partir de 1983. Para o AFS existem duas classes de máquinas: os servidores de ficheiros, onde se executam um conjunto de processos servidores que constituem a base de gestão do sistema e que deverão estar sob uma administração controlada e segura; os clientes que são máquinas que acedem aos serviços do AFS e que poderão não ser geridos de uma forma segura.

Aos clientes é apresentado um espaço de nomes particionado, formado por um espaço local e um espaço partilhado. O espaço partilhado é fornecido pelos servidores do sistema a qualquer cliente como uma hierarquia de ficheiros idêntica e transparente em relação à localização dos ficheiros.

Toda a gestão e configuração do sistema de ficheiros partilhado é feita nos servidores. O espaço de nomes é dividido em células, cada uma correspondendo basicamente a uma organização e dentro de cada célula em volumes. Os volumes são montados nos servidores de cada célula, de forma a oferecer um espaço de nomes consistente para todos os clientes.

A escalabilidade foi uma das principais motivações por detrás da concepção do sistema. Para atingir esse objectivo, o AFS utiliza uma estratégia de *caching* que tenta transpor alguma da carga dos servidores para os clientes. A *cache* dos clientes inclui ficheiros completos e é localizada em disco. Para lidar com ficheiros de dimensões elevadas, versões mais recentes do sistema permitem *caching* de porções de ficheiros de 64kb [8]. A política de validação dos dados em *cache* é iniciada pelo servidor através de um mecanismo designado por *callback*, o que o obriga a manter estado sobre quais os ficheiros abertos pelos clientes. Como contrapartida, o tráfego para efeitos de validação dos dados em *cache* é reduzido consideravelmente. A semântica oferecida pelo AFS é uma variante da semântica de sessão, na qual algumas operações[2] sobre os ficheiros são imediatamente visíveis, mesmo antes do ficheiro ser fechado.

2.10.5 Common Internet File System

O Common Internet File System (CIFS) [10] é uma evolução do Server Message Block (SMB) [19], um sistema de ficheiros distribuído nativo dos sistemas operativos Windows 95, Windows NT e OS/2. Mais que as suas características técnicas, a importância actual do CIFS é devida à sua vasta utilização em ambientes de computadores pessoais *desktop* baseados em sistemas operativos Windows. Adicionalmente, o protocolo CIFS é também utilizado para a partilha de ficheiros e impressoras pelo Samba [20], uma aplicação popular de ambientes Linux.

O mecanismo de localização do CIFS não obriga à definição de pontos de montagem nos sistemas de ficheiros dos clientes para que estes consigam ter acesso aos ficheiros remotos.

[2] Por exemplo, operações de alteração de permissões de ficheiros.

Em alternativa, os ficheiros e directorias remotos são referenciados e acedidos através de nomes de âmbito global, o que simplifica consideravelmente a configuração dos clientes. O espaço de nomes é definido pela Unified Naming Convention (UNC) (ver secção 2.2.2). Esta solução tem a desvantagem de não assegurar transparência de localização, pois o nome de um ficheiro indica explicitamente a sua localização na rede.

O CIFS suporta a selecção de várias semânticas de *locking* de ficheiros (Opportunistic Locks, ExclusiveOplocks, Batch Oplocks e Level II Oplocks) para a prevenção de conflitos quando múltiplos clientes acedem de forma concorrente ao mesmo ficheiro. A especificação de qual a semântica de *locking* permite ao sistema de ficheiros distribuído utilizar os mecanismos de *caching* e os modos de leitura/escrita remota que obtenham melhor desempenho em cada caso, embora mantendo a consistência necessária.

2.10.6 Coda

O Coda [21] é um DFS experimental cujo objectivo fundamental é oferecer aos seus clientes um acesso contínuo aos dados em situações de falhas do servidor ou da rede. Este DFS herda muitas das opções de desenho do AFS. Para atingir o objectivo do acesso contínuo aos dados, o Coda inclui também a funcionalidade de operação desligada.

A operação desligada surgiu como uma necessidade inerente à existência de sistemas computacionais móveis que usavam o sistema de ficheiros distribuído com uma acessibilidade que podia ser intermitente ao longo do tempo. Factores como a impossibilidade de estabelecer fisicamente uma ligação sem fios ou o custo elevado do acesso à rede podem implicar que um cliente do sistema de ficheiros distribuído passe por períodos em que não tem acesso à rede ou em que esse acesso é demasiado lento. A operação desligada permite colmatar estes períodos de acessibilidade ausente ou fraca aos servidores onde se encontram armazenados os ficheiros.

A operação desligada torna um sistema de ficheiros distribuído adaptável a estas variações de acessibilidade, permitindo que um utilizador continue a poder utilizar os seus ficheiros mesmo quando a ligação à rede deixou de existir, desde que uma réplica destes se encontre na *cache* local do cliente. A *cache* do cliente do sistema de ficheiros distribuído é assim aproveitada para, além de melhorar o desempenho do sistema, aumentar a sua disponibilidade.

Um aspecto fulcral do comportamento de um sistema de ficheiros distribuído que aja em operação desligada prende-se com a consistência entre réplicas guardadas localmente na *cache* dos clientes face ao ficheiro correspondente que está armazenado no servidor. Uma vez que as alterações efectuadas sobre as réplicas em *cache* não são imediatamente propagadas para os discos do servidor quando o sistema actua desligado, há que decidir que política utilizar para evitar e tratar eventuais operações conflituosas sobre os dados.

Uma política pessimista, que proíba qualquer escrita sobre dados replicados em partições distintas de uma rede ou que restrinja as leituras e escritas apenas aos ficheiros sobre os quais o cliente adquiriu controlo exclusivo ou partilhado de leitura, é inadequada. Nomeadamente, a primeira opção inibiria o funcionamento do sistema em operação desligada, uma vez que as escritas deixariam de ser permitidas nessa situação. A segunda opção obrigaria um cliente a prever a sua perda de acessibilidade à rede para pedir controlo sobre os ficheiros desejados e possibilitaria também situações indesejáveis no caso de algum cliente pedir o controle sobre um ficheiro e ficar desligado da rede por períodos longos, evitando que outros clientes pudessem ter acesso ao mesmo ficheiro.

Obviamente, os sistemas que disponibilizam operação desligada utilizam uma política de consistência optimista. Nesta estratégia, para obter uma maior disponibilidade, o sistema é

responsável por detectar os conflitos quando a ligação é reposta, resolvê-los automaticamente quando assim for possível e pedir a intervenção aos utilizadores nos restantes casos.

Os clientes podem encontrar-se num de três estados distintos durante a sua execução:

- ligado à rede, em que o servidor fornece os serviços do sistema de ficheiros distribuído de modo normal;
- desligado da rede, em que emula o papel do servidor do sistema, tentando servir as chamadas do sistema de ficheiros sempre que tal seja possível;
- e em reintegração, em que as alterações feitas no estado desligado são propagadas para o servidor do sistema de ficheiros distribuído após a ligação ter sido restabelecida, sendo esta a fase em que eventuais conflitos de consistência são tratados.

Um factor altamente influente na disponibilidade alcançada em operação desligada prende-se com os ficheiros que estavam replicados em *cache* local na altura da perda da ligação à rede [21]. Se o conjunto de ficheiros replicados em *cache* abranger muitos dos ficheiros que o utilizador irá usar no futuro, a disponibilidade do sistema em operação desligada poderá estender-se por longos períodos em que o acesso à rede não é necessário.

Para obter um conjunto de ficheiros replicados em *cache* que tenha alta probabilidade de ser representativo do leque de ficheiros a que o utilizador irá aceder no futuro, o Coda utiliza mecanismos de gestão prioritizada da *cache* local. Concretamente, o cliente tenta manter em *cache* ficheiros que provavelmente serão utilizados no futuro com base em informação implícita, tal como a lista dos ficheiros recentemente acedidos, e em informação explícita, através de uma lista definida pelo utilizador, *hoard database*, com os nomes dos seus ficheiros de interesse.

Para além do funcionamento em operação desligada, o Coda também suporta a adaptação do sistema a situações de fraca acessibilidade. Por situações de fraca acessibilidade entendem-se redes intermitentes, de largura de banda reduzida ou de custo elevado. A operação desligada pode ser vista como um caso extremo de acessibilidade fraca. Perante este tipo de situações, frequente em redes móveis, o Coda assume um funcionamento diferenciado que seja mais apropriado a essas circunstâncias. Entre outras, o funcionamento perante fraca acessibilidade contempla: um modo de validação da *cache* especialmente adaptado para situações de intermitência da rede; a propagação de alterações em *background* em condições de redes lentas; e resolução de *misses* da *cache* com a intervenção do utilizador quando a acessibilidade é fraca.

2.10.7 Low-bandwidth File System

O Low-bandwidth Network File System (LBFS) [22] explora as semelhanças existentes entre ficheiros diferente ou versões diferentes do mesmo ficheiro para poupar largura de banda. Para este fim, os ficheiros são divididos em porções, de largura variável, que são indexados pelo valor resultante da aplicação de uma função de dispersão ao seu conteúdo. A função de dispersão utilizada é a SHA-1 [23], cuja probabilidade dos valores calculados para duas entradas diferentes colidirem é muito inferior à probabilidade de erros em bits ocorrerem devido a falhas do hardware.

O mecanismo de redução do tráfego aplica-se às situações em que o conteúdo de ficheiros do DFS é transferido, pela rede, sendo aplicado tanto a servidores como a clientes. Se se tratar de um acesso de leitura, o computador remoto que receberá o ficheiro trata-se do cliente. Inversamente, se se tratar de um acesso de escrita, o computador remoto é o servidor.

Para usufruir das semelhanças existentes entre porções de ficheiros, o protocolo de transferência de ficheiros do LBFS tenta primeiramente indicar ao computador remoto qual o valor do índice da porção do ficheiro que pretende enviar. No caso do computador remoto encontrar um índice semelhante na sua tabela dos índices dos seus ficheiros locais, considera que a porção correspondente a esse índice é a mesma que se pretende enviar. Por conseguinte, o conteúdo da porção armazenada localmente é usado em vez de se proceder à sua transferência através da rede, reduzindo assim a largura de banda necessária. Apenas no caso de não existir um índice semelhante nos ficheiros armazenados localmente é que a porção é transferida pela rede. Este procedimento é feito tanto do lado do cliente, durante acessos de leitura, como do lado do servidor, durante acessos de escrita.

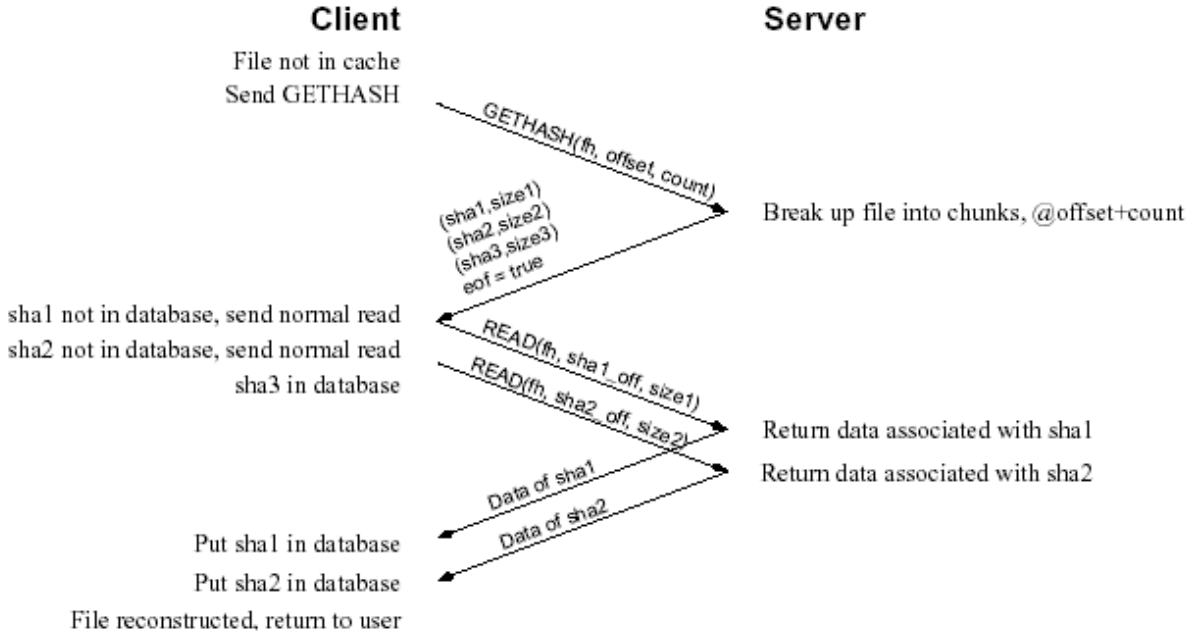

Figura 2: Protocolo LBFS para leitura de ficheiros remotos

Estudos feitos na análise do LBFS mostram que a probabilidade de existência de porções comuns entre ficheiros diferentes ou versões diferentes do mesmo ficheiro é elevada nos sistemas reais. Este facto implica que o LBFS obtenha desempenhos consideráveis face aos sistemas tradicionais em situações de fraca largura de banda.

2.10.8 xFS

O xFS [24][25] foi das primeiras propostas de sistemas de ficheiros distribuídos em que nenhuma máquina tinha responsabilidades diferenciadas de um servidor centralizado. Em alternativa, todas as máquinas funcionam cooperativamente para fornecer os serviços do sistema de ficheiros. Esta solução é orientada para a aplicação em redes locais de elevado desempenho, tirando partido desse meio para obter uma maior escalabilidade e desempenho que os sistemas tradicionais. O xFS explora a alta disponibilidade e desempenho de Redundant Arrays of Inexpensive Disks (RAIDs) [26] e utiliza um Log-structured File System [27][3] para melhor desempenho no uso de RAIDs. Finalmente, o modelo de consistência de *cache* é inspirado nas políticas de consistência de *cache* utilizadas em sistemas multiprocessador, tirando partido do elevado desempenho da rede.

[3] O aprofundamento das tecnologias de Redundant Arrays of Inexpensive Disks ou Log-structured File System não se enquadra no âmbito deste trabalho, pelo que é omitido neste texto. Para mais informações, deverão ser consultados os artigos referenciados.

2.10.9 Bayou

O projecto Bayou [28] fornece uma plataforma de suporte à partilha de um repositório de dados replicado. O ênfase do projecto foi colocado na exploração de mecanismos que permitissem a clientes móveis ler ou escrever sobre um repositório partilhado. Como principal objectivo, o sistema deverá apresentar-se aos utilizadores como um serviço centralizado e de alta disponibilidade que fornece acesso a um repositório de dados partilhado. Deve-se, contudo, referir que o Bayou não é um sistema de ficheiros distribuído. O sistema limita-se a disponibilizar uma API que poderá ser utilizada para conceber soluções de acordo com as necessidades específicas de cada aplicação. Em particular, os seus mecanismos poderão ser aplicados na concepção de um sistema de ficheiros distribuído, como acontece com o OceanStore, descrito na secção 2.10.10.

De seguida são analisados alguns dos aspectos mais relevantes da sua arquitectura, por forma a ilustrar de forma concreta algumas das questões referidas nas secções anteriores.

2.10.9.1 Suporte para computadores com recursos limitados

A arquitectura geral da plataforma prevê uma hierarquia de máquinas clientes e servidoras. Os servidores armazenam uma porção de dados do repositório global, enquanto que os clientes acedem para leitura ou escrita aos dados nos servidores. Obviamente, uma máquina servidora pode também comportar-se como cliente. Esta flexibilidade permite acomodar computadores de recursos extremamente heterogéneos, o que é típico em redes móveis. Os computadores de recursos reduzidos podem actuar como simples clientes, enquanto que os restantes podem actuar como servidores, colaborando na funcionalidade do sistema.

2.10.9.2 Alta disponibilidade para leituras e escritas

Uma das característica distintiva do Bayou é a sua política de replicação *read-any/write-any*, em que um utilizador pode ler ou escrever sobre qualquer réplica existente no sistema. Como consequência, a consistência oferecida é fraca, uma vez que não existe garantia de que as alterações efectuadas sobre uma réplica obtenham um resultado idêntico nas restantes réplicas. A prioridade do sistema é, pois, a de alcançar uma elevada disponibilidade, podendo assim sacrificar a consistência.

2.10.9.3 Propagação das escritas

Os servidores propagam as alterações ocorridas sobre as suas réplicas utilizando um protocolo de anti-entropia [29]. Neste protocolo, os servidores interagem numa filosofia *peer-to-peer*, propagando as suas alterações e evoluindo até um estado em que todos partilhem imagens idênticas das réplicas. Para o conseguir, cada servidor não só tem de receber todas as alterações, como ordená-las consistentemente. A utilização de um protocolo de anti-entropia garante que as diferentes réplicas convirjam para o mesmo estado e que, se não ocorrerem escritas suplementares, conseguirão alcançar esse estado. Um acesso de escrita sobre uma réplica, que originalmente é classificado como *tentative*, passa ao estado *commited* quando a sua posição final na ordenação global das escritas sobre a réplica é finalmente determinada pelo protocolo de anti-entropia. Pode, contudo, suceder que esse acesso seja abortado, em consequência de um conflito de escrita.

A arquitectura *peer-to-peer* de realização do protocolo de anti-entropia traz os benefícios citados na secção 2.4, pois quaisquer máquinas que consigam comunicar poderão propagar mutuamente as suas alterações.

Num esquema de replicação *read-any/write-any*, situações de conflitos de escrita são inevitáveis. O Bayou fornece suporte para a detecção de conflitos e respectiva resolução com

base na especificidade de cada aplicação. A detecção de conflitos é feita pela inclusão de um *dependency set* junto dos dados a serem escritos numa operação de escrita. Um *dependency set* consiste num conjunto de predicados relativos ao estado da réplica que deverão verificar-se quando a escrita for concretizada. Um conflito é detectado se a aplicação de um *dependency set* não se verificar. Em caso de conflitos, a sua resolução é também efectuada com base em procedimentos fornecidos pela aplicação que fez a escrita dos dados. Estes procedimentos, chamados *merge procedures*, são invocados quando um conflito é detectado e retornam um conjunto alternativo de escritas a ser aplicado à réplica.

2.10.9.4 Equilíbrio consistência/disponibilidade

O Bayou fornece alguns meios das aplicações ajustarem o funcionamento do sistema de acordo com os respectivos requisitos de consistência e a sua tolerância a dados inconsistentes.

Um primeiro meio é através do uso de *session guarantees* fornecidas pelo Bayou [30]. A utilização *session guarantee* garante que o estado de uma réplica é mantido consistente no que diz respeito às alterações efectuadas pelo próprio cliente. Sem a utilização de *session guarantees*, é possível a ocorrência de situações inconsistentes mesmo quando existe apenas um cliente. Considere-se, por exemplo, que esse cliente efectua duas escritas à mesma réplica, mas cada uma efectuada num servidor diferente. Após a propagação das escritas, a sua ordenação relativa pode não ser a mesma com que o cliente as efectuou e, desse modo, gerar um estado inconsistente. Existem quatro *session guarantees*[4] ao dispor das aplicações, cada qual para evitar a ocorrência de um determinado tipo de conflito, que podem ser seleccionadas pelas aplicações em função das suas exigências de consistência. Obviamente, a utilização de *session guarantees* reduz a disponibilidade do sistema.

Complementarmente, as aplicações podem decidir se consideram os dados resultantes de escritas *tentative* ou se, alternativamente, apenas utilizam os dados depois de *commited*.

Finalmente, a frequência com que os servidores interagem para propagar as suas alterações pode ser configurável pelas aplicações. Uma maior frequência de propagação de alterações permite reduzir as probabilidades de réplicas inconsistentes.

2.10.10 OceanStore

O sistema OceanStore [31] faz uso dos mecanismos de resolução de conflitos do Bayou (ver secção 2.10.9) para conceber um sistema de ficheiros replicado adaptado para assegurar o armazenamento persistente de informação à escala mundial.

É assim necessário um número elevado de servidores que poderão não ser confiáveis. Por esta razão, o OceanStore adapta os mecanismos de resolução de conflitos para que possam funcionar sobre dados cifrados, de modo a que os servidores nunca tenham acesso à informação em claro dos utilizadores do sistema. Para tal, o OceanStore restringe o conjunto de predicados que podem ser aplicados em *dependency sets* (secção 2.10.9.3) para apenas aqueles que são calculáveis sobre dados cifrados. De igual modo, o conjunto de operações disponíveis aos *merge procedures* (secção 2.10.9.3) é também limitado para aquelas que podem ser executadas correctamente sobre criptogramas.

Outro aspecto interessante do sistema é a utilização da replicação de fragmentos dos ficheiros do repositório partilhado, codificados de forma extremamente redundante, que assegura que só situações de desastres globais possam destruir informação do repositório. A esta propriedade, os autores designam por *deep archival storage*.

[4] As *session guarantees* disponíveis são: *Read Your Writes*, *Monotonic Reads*, *Writes Follow Reads* e *Monotonic Writes*.

Outro objectivo do sistema consiste em maximizar a localidade dos dados face aos clientes que lhes acedem. Para esse efeito utilizam uma política que designam por *promiscuous caching*, que permite que os dados migrem livremente pelos nós servidores para junto dos clientes mais frequentes. Embora esta política dificulte a manutenção da consistência e a localização dos dados, tem compensações a nível de desempenho e disponibilidade que os autores consideram de extrema importância [31].

O OceanStore não considera nenhuma estrutura de nomes global única. Em alternativa, cada utilizador escolhe uma ou mais directorias de raiz que constituem a sua estrutura de nomes personalizada. A identificação dos objectos do sistema é feita por um identificador global único, constituído por um vector de bits de dimensão fixa. Estes identificadores são independentes da localização (secção 2.2.1), permitindo a implementação da política de *prosmicuous caching*. A atribuição de um identificador a um objecto é feita usando uma adaptação do conceito de *self-certifying names* [32]. Nesta adaptação, o identificador de um objecto é obtido pela aplicação a uma função de dispersão segura de um nome textual e legível do objecto e da chave do utilizador que possui o objecto. Este mecanismo evita uma entidade central de gestão dos nomes do sistema, ao mesmo tempo que o previne contra tentativas de acessos ilícitos por parte de utilizadores mal intencionados. Como o cálculo do identificador de um objecto é calculado, não só com base no seu nome textual, mas também com a chave do seu dono, apenas este último deverá conseguir gerar identificadores correctos para os seus objectos.

2.10.11 Rumor

O Rumor [12] é um sistema de ficheiros distribuído de replicação optimista que utiliza um modelo *peer-to-peer* que permite a propagação de escritas entre quaisquer nós que contenham réplicas. O seu objectivo principal é o suporte à computação móvel, tendo herdado muitas das suas características arquitecturais do sistema de ficheiros replicado Ficus [33].

Ambos os sistemas permitem que sejam feitas alterações aos dados do repositório partilhado desde que alguma réplica esteja acessível. A consistência das réplicas baseia-se num algoritmo de reconciliação que é executado periodicamente entre as réplicas que estejam acessíveis entre si.

A reconciliação opera sobre conjuntos de ficheiros designados por volumes. Como tal, cada volume deve ser armazenado por completo em cada réplica e reconciliado no seu todo. Este facto pode ser uma desvantagem significativa para ambientes móveis, onde a capacidade de armazenamento é tipicamente reduzida. Como solução, o Rumor dispõe da possibilidade de replicação selectiva [34], que permite que as réplicas possam armazenar apenas porções de um volume, efectuando uma reconciliação por ficheiro nestes casos.

Numa comparação com o projecto Bayou, o Rumor distingue-se principalmente pela possibilidade de replicação selectiva e pelo facto dos seus mecanismos de detecção e resolução de conflitos serem transparentes para as aplicações.

2.10.12 PAST

Tal como o projecto OceanStore, também o PAST [35][36] pretende oferecer um sistema de ficheiros replicado de larga escala. Do mesmo modo, os seus principais objectivos passam por garantir uma elevada disponibilidade e durabilidade do repositório partilhado. No entanto, ao contrário do OceanStore, o PAST concentra-se na disponibilização de um repositório de dados que deverá ser usado fundamentalmente para o armazenamento de ficheiros estáticos. Deste modo, o PAST apenas fornece uma semântica de ficheiros partilhados imutáveis (secção 2.3.3). Neste aspecto, o PAST assemelha-se a serviços *peer-to-peer* destinados à

partilha de conteúdos na Internet, tais como o Gnutella [37], FreeNet [38] ou Napster [39], entre outros.

O espaço de nomes é constituído por *fileIds* que são gerados como resultado da aplicação de uma função de dispersão ao conteúdo, nome textual e identificador do dono de cada ficheiro. Ou seja, cada *fileId* é associado a uma versão imutável de um ficheiro, pois depende do seu conteúdo.

A distribuição dos ficheiros pelas réplicas do sistema é feita de uma forma aleatória, de modo a garantir que a diversidade do conjunto de réplicas que armazenam cada ficheiro. Pretende-se, deste modo, uma maior durabilidade do repositório partilhado face a desastres não globais, assim como o balanceamento da carga dos servidores de réplicas.

Para tratar os problemas relacionados com a segurança, o PAST baseia-se no uso de *smartcards* que são adquiridos pelos utilizadores para acederem ao repositório distribuído. Este método permite também assegurar a propriedade de anonimicidade dos utilizadores e fornecedores de armazenamento que intervêm no sistema. Por outro lado, os *smartcards* suportam de forma descentralizada um sistema de limitação do espaço do repositório utilizado por cada utilizador.

2.11 Análise comparativa

As secções anteriores permitem ter uma visão global das principais decisões que se colocam perante a concepção de um DFS. Em complemento, os exemplos concretos de soluções existentes, quer de cariz experimental, quer de cariz comercial, mostram como essas decisões foram tomadas em função dos objectivos particulares de cada DFS.

Convém, neste momento, confrontar as opções de desenho aprofundadas nas secções anteriores com as características particulares dos ambientes móveis. Os exemplos concretos de DFSs que foram apresentados são, na sua maioria, orientados para ambientes de redes fixas (casos do Locus, NFS, Sprite, AFS, CIFS ou xFS). No entanto, muitas aspectos das suas estratégias de desenho podem considerar-se interessantes contribuições para a concepção de um DFS para utilização em *info-appliances*, quer em redes móveis com ligação a uma infrastrutura fixa, quer em redes *ad-hoc*.

2.11.1 Semântica de partilha

A escolha da semântica de partilha a suportar por um DFS tem implicações de peso em muitos dos aspectos do desenho do sistema. Em particular, a granularidade (secção 2.5.1), localização (secção 2.5.2) e políticas de modificação (secção 2.5.3) da *cache* são aspectos que estão intimamente ligados à semântica de partilha que é oferecida. Quaisquer decisões tomadas em relação a estes pontos não devem ser dissociadas da semântica de partilha escolhida.

Como foi referido na secção 2.3.5, as semânticas de sessão e Unix são predominantes nos DFSs existentes actualmente. No entanto, as semânticas de sessão arrastam consigo uma realização tipicamente mais simplificada, com ganhos ao nível do desempenho, tolerância a faltas e escalabilidade dos sistemas que as oferecem. Como desvantagem, estes sistemas não oferecem uma consistência tão forte quanto aqueles que suportam a semântica Unix.

No entanto, deve-se ter em conta que a frequência de acessos concorrentes a ficheiros é muito reduzida, de acordo com [7]. Por outro lado, o suporte às semânticas de sessão em muitos DFSs recentes (tais como AFS, Coda, LBFS) sugere que a maioria das aplicações actuais já se tenha adaptado à consistência mais relaxada que esta semântica de partilha assegura.

Deste modo, a semântica de sessão afigura-se como uma solução apropriada para DFSs orientados para ambientes de redes móveis ou *ad-hoc* onde, em particular, o factor desempenho é fundamental.

A escolha desta solução deve, no entanto, assegurar meios que permitam a utilização do sistema de ficheiros por aplicações com requisitos exigentes de consistência. Por exemplo, deverá ser possível a aplicação de trincos lógicos que garantam a exclusividade dos acessos de leitura, escrita ou ambos às aplicações que abram um ficheiro.

No contexto da semântica de sessão, opções como o *caching* de ficheiros inteiros (secção 2.5.1), localização dos dados da *cache* em memória secundária (secção 2.5.2) ou uma política de modificação de *write-on-close* (secção 2.5.3) são apropriados.

Contudo, deverá ser tido em conta que, devido às limitações de memória disponível às *info-appliances* e de largura de banda das redes sem fios, o carregamento em *cache* de ficheiros de grande dimensão poderá não ser aceitável. No caso de ficheiros de grande dimensão devem ser utilizadas políticas de acesso aos dados diferenciadas, tais como:

- o *caching* de partes do ficheiro, em oposição a ficheiros completos; ou
- o acesso directo ao servidor sem utilização de mecanismos de *caching*.

2.11.2 Espaço de nomes

De um modo geral, a transparência de localização e a disponibilização de uma estrutura de nomes única a todos os clientes são propriedades desejáveis a qualquer DFS (secção 2.2).

Nas soluções referidas na secção 2.10, apenas raras excepções não possuem alguma das duas primeiras propriedades. Nomeadamente:

- o CIFS (secção 2.10.5) faz uso de um esquema de nomes que não é transparente quanto à localização, pois uma componente do nome de um ficheiro identifica a máquina onde o ficheiro se encontra; e
- o NFS (secção 2.10.2) possibilita a existência de várias árvores de nomes distintas, pois a versatilidade do seu esquema de nomes oferece a cada cliente a liberdade de incluir directorias remotas em qualquer parte do seu espaço de nomes local.

Embora possam ser apontadas algumas vantagens pontuais a estas soluções[5], as metas da transparência de localização e de uma estrutura de nomes única global são usualmente prioritárias em qualquer DFS.

No caso do CIFS e NFS, essas metas foram colocadas de lado principalmente pelo objectivo de fornecer uma estrutura versátil que permitisse que cada máquina fosse, simultaneamente, cliente e servidor do sistema. Este objectivo tem grande interesse para ambientes em que não existe uma autoridade única que centralize a administração e o controle dos conteúdos disponibilizados pelo DFS. Tipicamente, as redes *ad-hoc* e as redes móveis são excelentes representantes desses ambientes. Este objectivo pretende que qualquer máquina cliente possa ser também servidora de conteúdos de uma ou mais sub-árvores da estrutura global do sistema.

Resumindo, as propriedades da transparência de localização e disponibilização de uma estrutura de nomes única são desejáveis em qualquer DFS, incluindo o caso específico das redes *ad-hoc* ou redes móveis. Por outro lado, a possibilidade de qualquer máquina disponibilizar conteúdos para a estrutura global do sistema é extremamente interessante, se

[5] Ver secções 2.10.5 e 2.10.2, respectivamente.

conjugada com as anteriores. Uma solução para a conjugação destas propriedades é proposta na secção 3.3.

2.11.3 Arquitectura

Como foi discutido na secção 2.4, a arquitectura cliente-servidor é geralmente apropriada para redes fixas ou mesmo redes móveis com acesso a uma infra-estrutura fixa onde residem os servidores. Ou seja, um DFS orientado para *info-appliances* que se apoiem em servidores situados numa rede fixa deverá usar uma abordagem cliente-servidor. Os clientes móveis deverão ser também dotados da autonomia oferecida pela funcionalidade de operação desligada, para que possam continuar a funcionar quando a infra-estrutura fixa não estiver acessível. O Coda (ver secção 2.10.6) é um exemplo de um DFS adaptado a estes ambientes e que usa esta mesma abordagem.

Por seu lado, uma arquitectura *peer-to-peer* parece ser uma solução interessante para um sistema orientado a redes *ad-hoc*. Esta alternativa oferece uma maior independência aos nós do sistema, pois estes deixam de estar dependentes da disponibilidade de um ou mais servidores centrais para operarem (secção 2.4). No caso concreto das redes *ad-hoc*, nas quais é expectável um dinamismo elevado da sua topologia, as máquinas podem tornar-se inacessíveis frequentemente e de uma forma imprevisível. Logo, a autonomia acrescida dos nós de uma arquitectura *peer-to-peer* é um aspecto altamente atraente para um DFS adaptado a redes *ad-hoc*.

No entanto, as características próprias dos ambientes *ad-hoc* (ver secção 1.2) sugerem uma ponderação mais cuidada da tensão entre a autonomia e a interdependência dos seus nós. Por um lado, o dinamismo da topologia das redes *ad-hoc* é um argumento a favor da autonomia dos nós. Mas por outro lado, a possibilidade de existirem nós de recursos reduzidos numa rede *ad-hoc* aconselha a que esses nós deleguem responsabilidades de processamento armazenamento para nós dotados de melhores recursos, o que contraria a filosofia *peer-to-peer*. Uma solução viável deverá estabelecer um equilíbrio entre a autonomia dos nós de uma solução *peer-to-peer* pura e a interdependência entre nós de recursos heterogéneos. Idealmente, esse equilíbrio deverá ser dinâmico, através da capacidade de cada nó móvel se adaptar face às suas circunstâncias actuais [40].

2.11.4 Desempenho da comunicação

Os utilizadores de sistemas de ficheiros distribuídos tradicionais raramente consideram a sua utilização em redes lentas ou wide-area networks, dado que o seu desempenho seria inaceitável face às exigências aplicacionais típicas. As redes móveis, pelas características gerais das suas ligações, também são ambientes pouco adequados ao débito usual de um sistema de ficheiros distribuído. As aplicações pressupõem usualmente larguras de banda do sistema de ficheiros superiores a 1Mbps [22].

Como exemplo concreto, considere-se o caso de um processador de texto que efectua *auto saves* periódicos. De modo a garantir um bom funcionamento, a aplicação assume que cada *auto save* não prejudica a interacção com o utilizador nem consome demasiados recursos. O objectivo de um sistema de ficheiros distribuído a funcionar em situações de baixa largura de banda deverá ser o de assegurar um desempenho próximo do de um sistema de ficheiros local, por forma a que as aplicações o usem transparentemente. Para este efeito, muitos sistemas utilizam mecanismos que tentam obter uma redução da quantidade de dados transferidos de modo a reduzir os requisitos de largura de banda das redes em que são utilizados.

O AFS utiliza *callbacks* dos servidores para informar os clientes quando outros clientes alteraram ficheiros em *cache*. Deste modo, e em comparação com a política de consistência de

cache do NFS, os utilizadores podem aceder aos ficheiros em *cache* sem necessidade de contactar o servidor e assim evitando ocupar os recursos de rede desnecessariamente.

Os *leases* [41] são uma variação dos *callbacks* em que a obrigação do servidor de informar um cliente sobre as alterações de ficheiros em *cache* expira após um período de tempo. Deste modo o estado mantido pelo servidor é reduzido, o número de *callbacks* ocorridos é menor e o tratamento de situações de falha ou de inacessibilidade dos clientes é simplificado.

Contudo, deve-se ter em consideração que tanto a solução de *callbacks* como a de *leases* obrigam à manutenção de estado do lado dos servidores. Como foi referido na secção 2.6.1, um servidor *stateful* implica desvantagens ao nível da tolerância de faltas do sistema, quando em comparação com uma solução *stateless*. Para além disso, os dispositivos móveis são geralmente caracterizados por recursos de memória relativamente reduzidos, o que também aconselha a escolha de uma solução *stateless*. A escolha entre uma destas soluções de invalidação de *cache* face a uma política de invalidação iniciada do lado do cliente, não obrigando manutenção de estado do servidor, deverá ser, pois, ponderada cuidadosamente.

O protocolo NFS4 agrupa múltiplas operações do sistema de ficheiros e transfere-as em lote pela rede. Assim o número de mensagens trocado pelo sistema é reduzido e, consequentemente, diminui-se a quantidade de dados transferidos pela rede.

A compressão de dados é outra abordagem que permite diminuir a dimensão dos dados trocados entre cliente e servidores. Com o aumento do poder computacional dos dispositivos, o que torna o tempo gasto em compressão e descompressão desprezável, esta é uma solução bastante interessante e que pode normalmente ser complementada com outras abordagens.

Finalmente, outra solução interessante é aquela utilizada pelo Low-bandwidth File System, que descrita na secção 2.10.7. No entanto, os excelentes resultados obtidos por esta solução, apresentados em [22], não deverão ser, à partida, considerados como válidos no universo das *info-appliances*. As tarefas desempenhadas pelos dispositivos móveis são tipicamente diferentes daquelas levadas a cabo em estações *desktop*, o que poderá implicar uma utilização do sistema de ficheiros muito diferenciada. Por outro lado, a capacidade de memória dos dispositivos móveis, reduzida quando comparada com a de computadores *desktop*, poderá ter uma grande influência no tipo de ficheiros que estes dispositivos armazenam. Estes factores poderão alterar significativamente a eficácia do mecanismo de redução de tráfego do LBFS em *info-appliances*, pelo que a sua adequação a esses ambientes deverá ser cuidadosamente avaliada.

Concluindo, a compressão de dados e o mecanismo de redução de tráfego usado pelo LBFS são medidas com interesse para um DFS para *info-appliances*. Contudo, o último deverá ser alvo de uma avaliação sob condições típicas de uso de *info-appliances* que permita determinar a sua real eficácia. O mecanismo de validação de *leases* é também apropriado para um bom desempenho do sistema, associado à manutenção de uma quantidade reduzida de estado no servidor. No entanto, a solução de *leases* só poderá ser aplicada em DFSs cujos servidores sejam *stateful*. Finalmente, o envio pela rede de operações agrupadas em lote é, sempre que possível, uma medida vantajosa para a redução do tráfego de comunicação do DFS.

2.11.5 Segurança

Tal como os DFSs sobre redes fixas, os sistemas sobre redes móveis deverão considerar políticas de segurança que assegurem uma utilização segura dos repositórios partilhados por parte de todos os intervenientes. Face às vulnerabilidades das redes móveis (secção 1.2), o modelo de confiança subjacente às políticas de segurança deverá considerar a rede, os servidores e os restantes utilizadores como entidades não confiáveis. Contudo, e tal como no

caso das redes fixas, a implantação de mecanismos de segurança implica geralmente prejuízos de desempenho (secção 2.9) que raramente são desprezáveis. Esses prejuízos deverão ser especialmente notórios em redes de baixo desempenho ou máquinas de recursos computacionais relativamente limitados, o que é expectável em cenários de redes móveis. Em conclusão, a aplicação de mecanismos de segurança deverá ser uma decisão que pondere, por um lado, os requisitos de desempenho do sistema e, por outro, o nível de segurança exigido para os dados partilhados.

3 Arquitectura

Neste capítulo são descritas de modo sucinto as principais opções tomadas que tiveram influência no desenho e realização do sistema de ficheiros distribuído a que este relatório se refere.

3.1 Dois ambientes, duas arquitecturas

Se consideramos o âmbito de utilização de uma info-appliance, tipicamente um dispositivo móvel, podem ser distinguidos dois principais ambientes, conforme foi referido na secção 1. O primeiro ambiente baseia-se na existência de uma infra-estrutura de rede fixa na qual estão situadas as máquinas servidoras, usualmente computadores *desktop* dotados de recursos abundantes. Os dispositivos móveis são integrados nesta infra-estrutura como clientes do sistema, estabelecendo ligações sem fios a pontos de acesso à rede fixa. Finalmente, o segundo ambiente, designado por redes *ad-hoc*, baseia-se apenas nas interacções entre nós móveis que, usando as suas capacidades de comunicação sem fios, formam redes espontâneas entre si.

O objectivo deste projecto é o de conceber um sistema de ficheiros distribuído que seja adequado a qualquer um destes ambientes. No entanto, uma análise individual a cada ambiente permite concluir que o sistema deverá ser baseado em arquitecturas diferentes consoante o ambiente em que se execute, conforme foi discutido na secção 2.11.3.

No caso em que as *info-appliances* são usadas numa rede móvel com acesso a uma infra-estrutura fixa, é usada uma arquitectura cliente-servidor. Os servidores do sistema são computadores de recursos relativamente abundantes, localizados na rede fixa, enquanto que os dispositivos móveis onde se executam as *info-appliances* se comportam como clientes do sistema.

Em contraste, é usada uma arquitectura *peer-to-peer* no caso em que a info-appliance é executada no contexto de uma rede *ad-hoc*.

Associados a cada arquitectura, são também usados dois esquemas de nomes distintos para cada ambiente. No primeiro caso, é utilizado um esquema de nomes integrado, semelhante àquele disponibilizado nos sistemas AFS (secção 2.10.4) ou Coda (secção 2.10.6). Por seu lado, no caso da arquitectura *peer-to-peer* é adoptado o esquema de nomes baseado em *ad-hoc folders*, que será apresentado na secção 3.3.

Idealmente, o sistema deverá fornecer suporte à adaptação aplicacional a cada um dos ambientes de execução citados atrás. Para tal, a API oferecida pelo DFS deverá ser estendida para incluir uma funções que permitam às aplicações ou sistema operativo definir dinamicamente qual o modo de funcionamento pretendido.

3.2 Principais opções de desenho

Em qualquer um dos ambientes citados em 3.1, o sistema apresenta opções semelhantes em relação a vários aspectos do seu desenho. Nos parágrafos seguintes apresentam-se os principais aspectos que caracterizam o desenho do DFS. As decisões que são referidas abaixo são baseadas nas conclusões obtidas pela análise efectuada na secção 2.11.

O espaço de nomes disponibilizado aos utilizadores e aplicações será particionado entre o espaço de nomes local e partilhado. Para tal, o espaço de nomes local continuará intacto, sendo adicionada uma directoria à raiz do sistema de ficheiros que incluirá todo o espaço de nomes partilhado. Esta solução é semelhante àquela adoptada pelo AFS (secção 2.10.4).

Pelas razões invocadas em 2.11.1, a semântica de partilha oferecida será a semântica de sessão (secção 2.3.2). Nomeadamente, esta opção permite obter vantagens ao nível da simplicidade da realização, tolerância a faltas, escalabilidade e desempenho.

O sistema utilizará mecanismos de *caching* do conteúdo dos ficheiros remotos. Como consequência da semântica de sessão, a granularidade da *cache* será ao nível do ficheiro completo (secção 2.5.1) e a sua política de modificação será *write-on-close* (secção 2.5.3). No entanto, o sistema fornecerá mecanismos para lidar com ficheiros cuja dimensão seja demasiado elevada. Nesses casos, o mecanismo de *cache* será omitido, sendo os acessos feitos de forma directa e síncrona sobre o servidor do ficheiro remoto.

Em relação à política de validação, optou-se pela utilização do mecanismo de leases (ver secção 2.11.4). Esta opção implica a manutenção de estado nos nós que sejam servidores, o que traz consequências negativas em relação à tolerância de faltas e ao espaço de memória consumido nesses nós. Estes aspectos são, no entanto, minimizados pelo facto do mecanismo de *leases* ser tolerante a faltas do lado dos clientes[6] e pelo facto do estado mantido no servidor ser reduzido face a outras soluções de invalidação iniciada do lado do servidor[7]. O factor desempenho foi decisivo para a escolha desta política de invalidação, pois o tráfego de rede resultante de mensagens para a validação de *cache*s é reduzido de forma considerável face a outras soluções (secção 2.11.4).

Para fornecer um grau acrescido de autonomia às *info-appliances*, o sistema disporá da funcionalidade de operação desligada, fazendo uso da *cache* de ficheiros completos para esse efeito.

Como medida para obter um desempenho melhorado na transferência do conteúdo dos ficheiros pela rede de comunicação, será utilizada compressão de dados. Adicionalmente, a solução adoptada pelo LBFS deverá ser avaliada no contexto das *info-appliances*, de modo a verificar se a sua eficácia se mantém com os conteúdos e cenários de utilização típicos dos sistemas de ficheiros desses ambientes. Caso essa eficácia seja aceitável, essa solução deverá também ser incluída no DFS a desenvolver.

Os aspectos de segurança não foram ainda considerados na fase do projecto a que este relatório se refere. O sistema a realizar deverá, no entanto, contemplar soluções já existentes para a implantação de mecanismos de autenticação de utilizadores, controlo de acesso, privacidade e integridade dos dados. A aplicabilidade dessas soluções deverá, obviamente, ser ponderada face aos requisitos de desempenho do sistema a desenvolver.

3.3 *Ad-Hoc* folders

Considere-se o seguinte cenário: um conjunto de pessoas trabalha cooperativamente através de um sistema de ficheiros distribuído sobre uma rede *ad-hoc*. Esta equipa cria uma directoria partilhada no DFS onde estarão localizados os ficheiros sobre os quais trabalhará. A esta directoria por todos partilhada pode ser feita uma analogia com a mesa de trabalho comum, onde os elementos da equipa disponibilizam documentos e os alteram.

[6] No entanto, fica por resolver o caso de falhas do servidor ou de comunicação.
[7] Caso do mecanismo de *callbacks* (ver secção 2.10.4), por exemplo.

As realizações convencionais de esquemas de nomes de DFSs têm tipicamente uma granularidade de partilha ao nível do directório[8]. Ou seja, o menor objecto que os servidores podem partilhar é um directório. O conteúdo de cada directório partilhado, acessível por parte dos clientes do sistema, está assim integralmente localizado num único servidor. Ou seja, do ponto de vista semântico, o servidor que partilha um directório é o único dono dos seus conteúdos. Eventualmente, mecanismos de replicação podem descentralizar a localização dos dados para junto dos clientes por razões de disponibilidade e desempenho. Deste modo, e atendendo ao cenário anterior, quando o servidor responsável pela directoria de trabalho saísse fora do alcance da rede ou ficasse sem bateria, todo o ambiente de trabalho ficaria inacessível aos restantes trabalhadores.

Numa primeira análise, duas soluções poderiam minimizar os efeitos da inacessibilidade do servidor da directoria de trabalho:

- A possibilidade de operação desligada por parte dos clientes permitiria que estes continuassem a trabalhar sobre os ficheiros que estivessem replicados na sua *cache* no momento em que o servidor saiu da rede *ad-hoc*. No entanto, é de prever que a maioria dos dispositivos móveis tenha uma capacidade de memória reduzida. Como consequência, é aceitável considerar que esses nós não consigam assegurar a replicação de um número suficiente de ficheiros que permita um funcionamento eficaz em modo desligado. Por outro lado, a operação desligada fornece uma consistência relaxada quando os clientes se exscutam em modo desligado, o que não garante que as alterações feitas sobre as réplicas sejam, de facto, verificadas na cópia primária no servidor (ver secção 2.10.6).
- Outra solução poderia passar pelo particionamento do conteúdo da directoria de trabalho pelos vários nós da rede *ad-hoc*. Este particionamento poderia seguir uma estratégia que associasse, por exemplo, o dono de cada nó aos documentos ligados às suas áreas de responsabilidade. Neste caso, a inacessibilidade de um dos nós implicaria apenas a perda da porção do conteúdo que por ele era disponibilizada. As directorias partilhadas pelos restantes nós continuariam a estar acessíveis. Ao contrário da operação desligada, o conteúdo que continuaria disponível continuaria a ser acedido directamente por pedidos ao servidor. Ou seja, continuaria a ser assegurada uma consistência forte aos clientes. Contudo, esta solução tem a desvantagem óbvia de não ser transparente quanto à localização dos ficheiros. Ou seja, para abrir um ficheiro é necessário conhecer em que dispositivo este se encontra e, com base nessa informação, seleccionar a directoria correspondente.

Nas subsecções que se seguem, é apresentado um esquema de nomes que se pretende que seja uma solução para o cenário descrito acima. Esse esquema de nomes, cujas unidades fundamentais são repositórios de ficheiros designados por *ad-hoc folders*, possui muitas das propriedades que foram consideradas desejáveis na discussão da secção 2.2. Entre elas, é transparente da localização e fornece uma estrutura de nomes única e global aos seus clientes, ao mesmo tempo que permite que qualquer nó disponibilize conteúdos para essa estrutura global. Adicionalmente, a aplicação deste esquema de nomes implica melhorias ao nível da disponibilidade do repositório partilhado, nomeadamente em cenários semelhantes ao referido anteriormente nesta secção.

3.3.1 Esquema de nomes baseado em *ad-hoc* folders

A solução aqui proposta tem como característica fundamental a possibilidade de uma directoria partilhada do DFS conter ficheiros que estejam armazenados remotamente em nós

[8] Casos do Locus, NFS, Sprite, AFS, CIFS, Coda ou LBFS, por exemplo.

distintos. Ou seja, uma directoria partilhada deixaria de ser uma entidade com um servidor responsável associado, para passar a ser uma entidade para a qual qualquer nó pode contribuir com conteúdo, sob a forma de ficheiros que lá são partilhados. Estas directorias partilhadas, que designarei por *ad-hoc folders*, são os blocos constituintes da estrutura de nomes global do sistema de ficheiros e situam-se directamente abaixo da raiz da mesma. Aos nós que contribuem para adicionar conteúdo ao directório partilhado designarei por participantes no directório.

Uma propriedade inerente aos *ad-hoc folders* é o facto do seu conteúdo poder ser altamente dinâmico, como resultado da entrada e saída de participantes na rede *ad-hoc* (secção 1.2). À medida que diferentes participantes se tornam acessíveis ou inacessíveis, também os ficheiros ou directórios por si partilhados ficam disponíveis ou indisponíveis no directório em que participam. Esta característica é, de certo modo, análoga ao dinamismo da topologia de uma rede *ad-hoc*.

3.3.2 Exemplo de utilização dos *ad-hoc* folders

Para melhor compreender a semântica dos *ad-hoc* folders e como os utilizadores poderão tirar o máximo proveito das suas características, as linhas seguintes apresentam um exemplo ilustrativo do seu funcionamento no sistema operativo Windows CE. Considere-se o cenário descrito na introdução da secção 3.3.

- Um dos utilizadores cria, dentro do directório raiz "My Ad-hoc folders", um directório a que dá o nome de "Reunião GSD". O sistema de ficheiros distribuído cria, assim, um novo *ad-hoc folder* com o nome "Reunião GSD". A este *ad-hoc folder* é atribuído um GUID, que será usado para o identificar no âmbito do sistema de ficheiros distribuído.

Figura 3: Criação de um *ad-hoc* folder

- A criação do novo *ad-hoc folder* é propagada aos restantes computadores onde o sistema de ficheiros distribuído está instalado. Os utilizadores desses computadores passam a visualizar, no seu directório "My Ad-hoc Folders", o novo *ad-hoc folder* criado. Possivelmente, o utilizador que criou o directório anuncia verbalmente aos restantes colegas que o directório a ser usado durante a reunião será o "Reunião GSD".

Figura 4: *Ad-Hoc* folder criado passa a estar visível em "My Ad-hoc Folders"

- A partir deste momento, qualquer utilizador da reunião pode copiar, mover ou criar ficheiros ou directórios para o *ad-hoc folder* de modo a partilhá-los com os restantes colegas da reunião. Estas operações são feitas usando a interface do sistema de ficheiros do Windows CE. Os restantes utilizadores passam a poder aceder aos ficheiros colocados por qualquer elemento da reunião no *ad-hoc folder*.
- No caso de algum membro da reunião, incluindo aquele que criou inicialmente o *ad-hoc folder*, se ausentar da sala, tornando-se inacessível pela rede, os seus ficheiros deixarão de estar disponíveis no *ad-hoc folder*. No entanto, os ficheiros partilhados pelos restantes utilizadores continuarão poder ser utilizados.

3.3.3 Âmbito de um *ad-hoc folder* e utilização de GUIDs

Basicamente, a principal diferença que resulta do facto de um *ad-hoc folder* poder conter conteúdos armazenados em diferentes nós da rede relaciona-se com o seu âmbito. Em vez de uma directoria local a um servidor que é partilhada para acesso pelos clientes através do DFS, um *ad-hoc folder* passa a ser visto como um repositório global de objectos do sistema de ficheiros. Isto porque o *ad-hoc folder* deixa de estar directamente associado a algum servidor que o disponibiliza para poder ser constituído por conteúdos associados a qualquer nó que conheça o *ad-hoc folder*. Ou seja, um *ad-hoc folder* pode ser acedido por qualquer nó, em qualquer partição da rede *ad-hoc* que se encontre. Esse acesso pode ser de leitura ou escrita de ficheiros do *ad-hoc folder* que se encontrem disponíveis, porque os seus servidores estão acessíveis na rede, ou de criação de novos ficheiros dentro do *ad-hoc folder*. Para efectuar estes acessos, o nó precisa de conhecer o *ad-hoc folder*, ou seja, ter uma referência para o seu identificador. Essa referência pode ser obtida como resultado da criação de um novo *ad-hoc folder* por parte do próprio nó ou recebida com base em diálogos com outros nós para troca de identificadores de *ad-hoc folders*.

O espaço de nomes lógico é constituído por identificadores globais únicos (*globally unique identifier*, ou GUID) que referenciam os objectos do sistema, incluindo tanto os *ad-hoc folders* como os ficheiros e directorias que neles residam. Estes identificadores consistem em vectores de bits de dimensão fixa, que são gerados de um modo pseudo-aleatório que assegure a unicidade do identificador de cada objecto. A amplitude referencial do espaço de nomes, consequência directa da dimensão dos GUIDs, deverá ser suficientemente grande para que a probabilidade de ocorrência de dois GUIDs idênticos seja desprezável [7]. Uma realização possível para a geração dos GUIDs é aquela utilizada no âmbito do projecto OceanStore (ver secção 2.10.10).

Deve-se referir, no entanto, que o espaço de nomes lógicos composto por GUIDs é utilizado no âmbito do sistema de suporte do DFS. Ou seja, as rotinas remotas tratadas pelos servidores

do DFS recebem GUIDs como argumentos para referenciar os objectos do sistema. Estas rotinas, por sua vez, efectuam o mapeamento de GUIDs para, por exemplo, *file handles*.

Os utilizadores e aplicações que utilizem a interface de programação do DFS continuam a observar os objectos identificados transparentemente pelo seu nome textual e a usar esses nomes quando chamam funções da interface do sistema de ficheiros. Cada cliente do DFS deverá ser responsável por efectuar localmente o mapeamento entre o nome textual do ficheiro e o GUID respectivo quando interagir com o DFS.

A necessidade de identificação dos objectos por GUIDs, em vez de nomes de ficheiros convencionais, resulta do âmbito de utilização pretendido para os *ad-hoc folders*. Qualquer nó deverá poder criar novos *ad-hoc folders* ou ficheiros dentro de *ad-hoc folders* existentes, sob quaisquer condições de acessibilidade da rede. Ou seja, dois nós que se encontrem em partições de rede distintas poderão criar novos *ad-hoc folders* ou mesmo criar ficheiros sobre um *ad-hoc folder* que ambos conheçam. Nestas circunstâncias, é possível que sejam criados, de forma fortuita, objectos com nomes textuais idênticos por ambos os nós. A cada objecto será atribuído um GUID que, por natureza, será único[9], o que evita qualquer conflito da unicidade referencial no espaço de nomes.

Para resolver o problema do conflito de nomes textuais dos objectos, quando dois ou mais objectos com nomes idênticos se encontrarem simultaneamente acessíveis a um utilizador, os seus nomes deverão ser apresentados de forma diferenciada entre si. Por exemplo, considere-se que dois nós não acessíveis entre si criam ambos um *ad-hoc folder* com o nome *Reunião*. Imagine-se agora que esses nós passam a estar acessíveis entre si. A estrutura de *ad-hoc folders* passaria, assim, a ter dois objectos com o mesmo nome textual, pelo que deveria apresentá-los, por exemplo, do seguinte modo: *Reunião(1)* e *Reuniao(2)*.

3.3.4 Consequências no desenho do DFS

Para utilizar o esquema de nomes baseado em *ad-hoc folders*, os nós do DFS deverão assegurar os seguintes pontos:

- Utilizar GUIDs para referenciar os objectos do sistema no âmbito das chamadas às rotinas remotas a outros nós do DFS;
- Localmente, fornecer um mapeamento entre o GUID dos objectos do sistema e um nome textual a eles associados. O nome textual serve apenas para a interacção com os utilizadores e aplicações através da interface do sistema de ficheiros. Em caso de conflitos de nomes textuais de objectos distintos, deverão ser usadas variações do nome textual que permitam distinguir os objectos. Por exemplo, se dois ficheiros do mesmo *ad-hoc folder* tiverem o nome *readme.txt*, o utilizador poderá observá-los como *readme(1).txt* e *readme(2).txt*.
- Permitir que cada nó possa ser, simultaneamente, cliente e servidor do sistema[10].
- Efectuar a resolução de nomes do sistema com base no facto de, ao contrário dos modelos convencionais de partilha de ficheiros de directorias, um *ad-hoc folder* poder estar armazenado em mais que um servidor participante. Ou seja, o cliente deverá manter em memória uma estrutura que associe, a cada *ad-hoc* folder conhecido pelo cliente, um conjunto de servidores que nele participam.

[9] A probabilidade de serem gerados dois GUIDs idênticos para objectos distintos deverá ser suficientemente reduzida para que essa possibilidade seja considerada como impossível

[10] Esta possibilidade está presente em DFSs como o NFS (secção 2.10.2) ou o CIFS (secção 2.10.5).

- Diferenciar os *ad-hoc folders* das restantes directorias do sistema de ficheiros na interface com o utilizador. O conteúdo de um *ad-hoc* folder pode variar frequentemente em função da acessibilidade de outros nós da rede *ad-hoc*, ao contrário das directorias convencionais. Por esta razão, os utilizadores deverão ter um modo claro de saberem que estão a lidar com um *ad-hoc* folder. Uma solução pode passar por localizar obrigatoriamente os *ad-hoc folders* debaixo do directório raiz "My Ad-Hoc Folders". A eficácia da distinção entre os dois tipos de directórios com semânticas diferenciadas é essencial para evitar erros de utilização por parte do utilizador.

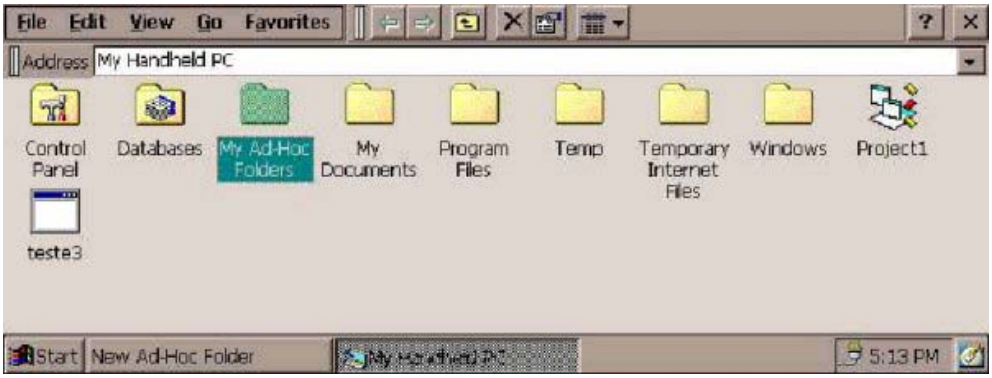

Figura 5: A directoria "My Ad-hoc Folders"

3.3.5 Propriedades do esquema de nomes

De seguida é feito um resumo das principais propriedades do esquema de nomes apresentado atrás.

O esquema de nomes é transparente em relação à localização, uma vez que os nomes dos ficheiros, que são apresentados aos utilizadores, não revelam qual é o servidor onde estão armazenados.

A mesma estrutura de nomes global é observada por qualquer cliente do DFS. Esta estrutura é constituída por uma árvore de raiz única que tem como ramos os *ad-hoc folders* existentes. Obviamente, num dado momento, um cliente apenas observa a porção da árvore global que lhe está acessível através da rede.

Este esquema de nomes permite que qualquer nó contribua com conteúdo seu para a estrutura partilhada, criando novos *ad-hoc folders* ou criando novos ficheiros dentro de *ad-hoc folders* existentes.

O esquema de nomes tem consequências na disponibilidade do repositório partilhado pelo DFS. Concretamente, esta solução permite um particionamento dos dados de cada directoria pelos nós da rede. O acesso aos dados particionados por múltiplos servidores é feito de forma transparente quanto à localização de cada ficheiro. Como resultado do particionamento, a falha ou inacessibilidade de um servidor apenas se reflecte na indisponibilidade de uma porção dos dados de uma directoria. Os restantes ficheiros da directoria que permaneçam disponíveis podem continuar a ser acedidos de forma normal pelos clientes.

3.4 Interfaces do sistema

Cada *peer* pode ser considerado como sendo constituído por uma componente servidora e uma componente cliente que mantêm estado entre si.

A componente cliente deverá ser capaz de responder às funções do sistema de ficheiros que sejam chamadas sobre os directórios partilhados pelo sistema de ficheiros distribuído. O

processamento destas funções poderá implicar, em alguns casos, o pedido de dados a outros *peers*. Nesses casos, o cliente terá de efectuar a invocação de uma ou mais rotinas remotas disponibilizadas pela componente servidora dos restantes *peers*. Para minimizar a utilização dos recursos de rede nestas situações, o cliente deverá fazer uso de *caching* para: o conteúdo dos ficheiros e directorias remotos, assim como da respectiva informação (atributos, datas de acesso, tamanho, etc.).

Por outro lado, a componente servidora deverá fornecer uma interface remota que permita aos restantes *peers* acederem aos ficheiros e directorias por si partilhados. Frequentemente, o processamento das rotinas por detrás dessa interface implica o acesso aos ficheiros e directorias armazenados no sistema de ficheiros local. Esse acesso é feito através da API de funções do sistema de ficheiros fornecida pelo Windows CE.

Analisando a arquitectura do sistema em função das suas interfaces com o exterior e entre *peers*, podemos identificar três interfaces distintas. Primeiramente, as aplicações interagem de modo transparente com a componente cliente utilizando a API das funções do sistema de ficheiros do Windows CE. Complementarmente, a componente servidora utiliza a mesma interface para aceder ao repositório de dados local. Finalmente, as componentes cliente e servidora comunicam entre si utilizando a infra-estrutura de suporte a RPC. A Figura 6 apresenta as interfaces utilizadas pelo sistema de ficheiros distribuído.

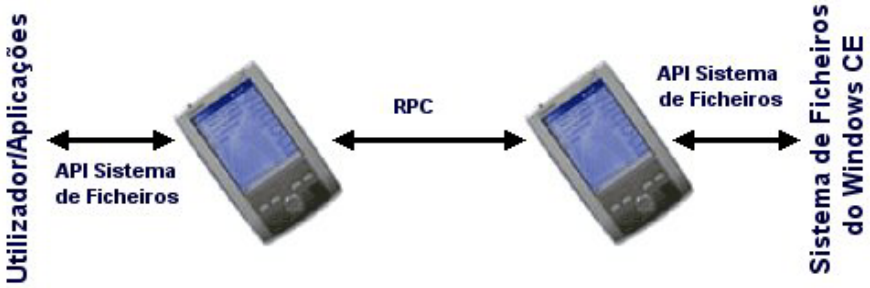

Figura 6: Interfaces entre as componentes do DFS

4 Realização

Este capítulo pretende descrever o estado do trabalho que foi desenvolvido até à data da escrita deste relatório. As próximas secções tentam demonstrar o que foi realizado, assim como as principais dificuldades encontradas durante o processo de desenvolvimento. Na secção 4.1 são descritos os aspectos de realização relacionados com a plataforma do sistema operativo instalada. De igual modo, a secção 4.2 trata da componente relativa à comunicação. A secção 4.3 descreve a realização do sistema de ficheiros distribuído. Finalmente, a secção 4.4 faz a avaliação do sistema concretizado face aos objectivos propostos.

4.1 Plataforma Windows CE

4.1.1 Utilização da plataforma de desenvolvimento do Windows CE

O sistema operativo eleito para a concretização deste trabalho foi o Microsoft Windows CE, versão 3.0. Este sistema operativo é destinado ao uso em sistemas embebidos, pretendendo fornecer uma solução apropriada para equipamentos portáteis como handheld PCs ou palm-sized PCs.

Um factor determinante para esta escolha foi o facto do Windows CE ser um sistema operativo bastante representativo do sector do mercado de dispositivos móveis. Concretamente, um estudo levado a cabo pela empresa Canalys nos mercados europeu, asiático e do médio oriente concluiu que este sistema operativo está instalado em cerca de 34% dos dispositivos móveis vendidos no presente ano, o que revela um aumento de quase 100% face aos resultados do ano passado [42]. Ou seja, num futuro próximo é provável que uma porção considerável dos equipamentos participantes em ambientes de redes *ad-hoc* utilizem este sistema operativo.

Outro aspecto determinante na selecção deste sistema operativo foi o facto de este permitir a incorporação de sistemas de ficheiros adicionais de uma forma extremamente modular e relativamente simples, sem obrigar a modificações no código do sistema operativo. Essa incorporação é disponibilizada por um componente do sistema de ficheiros, o FSD Manager, que permite a adição de Installable File Systems [11].

Adicionalmente, a plataforma de suporte ao desenvolvimento do sistema operativo, o Platform Builder, disponibiliza serviços de debug e de controle da execução do sistema operativo de grande utilidade.

Finalmente, o desenvolvimento de um sistema adaptado a um sistema operativo específico pode permitir um melhor desempenho. Nomeadamente, alguns aspectos do desenho podem ser ajustados para que se adaptem de forma mais optimizada ao funcionamento do sistema operativo. Como contrapartida, a portabilidade do sistema desenvolvido é assim reduzida, sendo mais dificultada a sua aplicação em sistemas operativos diferentes.

4.1.2 Sistema de ficheiros do Microsoft Windows CE

A componente do Windows CE com mais relevância para o TFC é o Object Store. Este termo refere-se aos três tipos de armazenamento persistente que são suportados pelo Windows CE: os sistemas de ficheiros, o registo e as property databases.

No que toca aos sistemas de ficheiros, o Windows CE disponibiliza a possibilidade de criação e instalação de um sistema de ficheiros através do componente FSD Manager [11]. Estes sistemas de ficheiros são referidos por Installable File Systems (IFS).

De um modo resumido, a arquitectura do sistema funciona da seguinte maneira, tal como ilustrado pela figura Figura 7:

- As aplicações executam chamadas ao sistema de ficheiros, através da API do mesmo;
- As chamadas são passadas primeiramente para o sistema de ficheiros incorporado no Windows CE; no caso dessas chamadas se reportarem a um volume dum IFS que tenha sido registado no FSD Manager, estas são-lhe passadas;
- O FSD Manager passa-as para o driver do IFS correspondente;

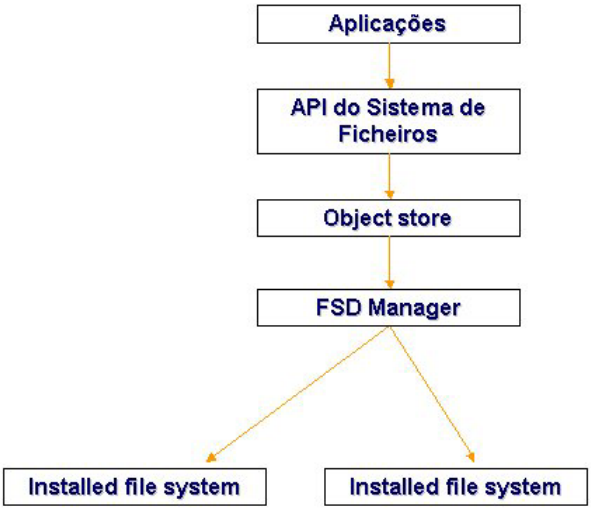

Figura 7: Fluxo de invocações resultante de uma chamada a uma função de um IFS

A criação de um IFS exige o desenvolvimento de um ficheiro dll onde é exportada a funcionalidade do sistema. As interfaces suportadas pelo IFS deverão estar entre as interfaces de sistemas de ficheiros definidas pelo Windows CE. No caso de uma interface do sistema de ficheiros não ser suportada pelo IFS, o sistema retorna um erro ao utilizador a indicar que a funcionalidade não é suportada [43].

Por sua vez, o FSD Manager disponibiliza um conjunto de serviços para uso dos IFSs, que podem ser usados para funções como o registo de volumes e a criação de *file handles* ou *find handles*. Os *file handles* são estruturas retornadas pela função CreateFile e que servem para as aplicações referenciarem os ficheiros abertos[11] na chamada a outras funções do sistema de ficheiros. Por sua vez, os *find handles* são estruturas retornadas pela função FindFirstFile e que são passadas como argumento a chamadas subsequentes à função FindNextFile para obter prosseguir com a pesquisa.

Dadas as vantagens oferecidas pelo uso de IFS, considero que este mecanismo é altamente adequado para o desenvolvimento do sistema de ficheiros distribuído do TFC. Contudo, o estado do trabalho realizado na altura da escrita deste relatório ainda não permite tirar conclusões concretas sobre a facilidade e eficácia deste mecanismo no âmbito dos objectivos do trabalho.

[11] De facto, os *file handles* do sistema de ficheiros do Windows CE servem também para referenciar outros objectos, tais como directorias, *pipes* ou dispositivos de disco [43].

4.1.3 Plataforma Windows CE instalada

Para instalação de uma plataforma baseada no Windows CE, utiliza-se a aplicação MS Platform Builder 3.0, que fornece funcionalidades de compilação de imagens do sistema operativo e respectiva instalação noutra máquina, assim como suporte ao desenvolvimento e execução de componentes e aplicações.

A estação de desenvolvimento consiste num computador pessoal em que se encontra instalado o Platform Builder 3.0. Adicionalmente, uma outra máquina é utilizada, na qual é executada a imagem do sistema operativo Windows CE 3.0 compilado. Esta máquina, com processador Intel Pentium, está configurada de modo a satisfazer os requisitos da *PC-Based Platform* (CEPC), enumerados na documentação de apoio ao utilizador.

A ligação entre a máquina alvo do sistema operativo e a estação de desenvolvimento é efectuada através de um adaptador de rede *Ethernet*. Esta ligação serve para efeitos de carregamento inicial da imagem do sistema operativo e de controle da execução do mesmo, através do serviço CESH do Platform Builder. A Figura 8 ilustra o ambiente de desenvolvimento instalado. Deve-se referir que, adicionalmente, outro adaptador Ethernet foi instalado para ser usado como *product Ethernet card*, que pode ser utilizada pelo sistema operativo e aplicações para acesso geral à rede.

Figura 8: Configuração do ambiente de desenvolvimento

Muitas funções de suporte à execução do sistema operativo disponibilizadas pelo Platform Builder revelam-se de elevada importância no processo de desenvolvimento do sistema de ficheiros distribuído. Nomeadamente, as funções de *debug* tais como a possibilidade de escrita de mensagens de teste para a consola na estação de desenvolvimento, a filtragem das mensagens de teste através do uso de *debug zones*, o controlo da execução por *breakpoints* e a possibilidade de inspecção do estado do sistema quanto a processos, threads e conteúdo da memória.

Por outro lado, algumas ferramentas suplementares fornecidas com o pacote do Platform Builder têm grande utilidade. Entre estas estão ferramentas como Remote Registry Editor, Remote File Viewer ou Remote Zoom In.

4.1.4 Processo de instalação e configuração da *plataforma* Windows CE

A instalação e configuração da plataforma Windows CE foi uma tarefa que ocupou uma porção significativa do tempo investido no trabalho. Tal deveu-se a algumas dificuldades encontradas.

Principalmente, o computador utilizado para a execução da imagem do sistema operativo era composto por alguns componentes de hardware que não faziam parte da lista oficial de periféricos suportados pelo Windows CE 3.0. Nomeadamente:

- A configuração inicial do computador era baseada num processador Intel 486DX2. Este processador, apesar de ser suportado pelo Windows CE 3.0, não era previsto pelos requisitos da PC-Based Platform. Como tal, a sua utilização obrigaria à alteração do código fonte do sistema operativo. Entre o código a alterar incluíam-se algumas rotinas programadas com instruções Assembly não suportadas pelo processador em causa. Algumas dessas rotinas foram corrigidas com sucesso. No entanto, considerou-se que o esforço necessário para realizar as restantes adaptações era pouco compensador em função dos objectivos do trabalho. Por essa razão decidiu-se utilizar uma configuração baseada num processador Intel Pentium, suportada pela PC-Based Plaftorm.
- A placa de rede para debug foi obtida de entre um conjunto de placas antigas, muitas das quais não estavam funcionais. Este facto deveu-se ao facto das especificações do PC-Based Platform aconselharem o uso de uma placa compatível NE2000 baseada na interface ISA. Uma placa com estas características já não é facilmente encontrada em locais de venda de produtos informáticos. Entre as placas que foram tentadas, algumas apresentaram avarias, enquanto que outras não funcionavam devido a conflitos com os restantes periféricos. Finalmente, a placa de rede escolhida utilizava uma configuração que divergia das especificações do PC-Based Platform no valor de IRQ (*Interrupt Request*) usado. Tal obrigou à alteração manual dos ficheiros de configuração da plataforma, tarefa que não é documentada pelo Platform Builder.

Por outro lado, a insuficiência de documentação de suporte ao utilizador do Platform Builder obrigou a esforços adicionais para a resolução de outros problemas simples de configuração da plataforma. Entre outros, os seguintes problemas foram registados:

- As ferramentas suplementares disponíveis no Platform Builder, tais como o Remote Registry Editor ou o Remote File Viewer, necessitaram da configuração da ligação com a máquina onde a imagem do sistema operativo se executava. A configuração do tipo de ligação carecia de qualquer tipo de documentação auxiliar, pelo que obrigou a tentativas exaustivas de várias opções.
- As alterações efectuadas às configurações da imagem do sistema operativo não eram armazenadas de forma persistente. Tal implicava que as alterações tivessem de ser repetidas manualmente após cada inicialização do sistema. No caso particular da *product ethernet card*, o seu funcionamento obrigava a que a sua configuração correcta estivesse disponível durante a inicialização do sistema, o que não era possível com o procedimento de configuração manual. A maior parte dos parâmetros configurados estava guardada no repositório do *Registry* [11]. A solução passou pela análise das entradas relevantes do registry do sistema em execução, utilizando a ferramenta *Remote Registry Editor*, disponível no Platform Builder. As entradas obtidas foram então adicionadas a um dos ficheiros de definição do estado inicial do *registry* da plataforma compilada pelo Platform Builder, de modo a permanecerem persistentes.

4.2 Mecanismo de RPC

O modelo de programação distribuída das interacções entre *peers* que se considerou mais apropriado para o trabalho foi o de RPC [7]. No entanto, o Windows CE 3.0 não fornece serviços de RPC às suas aplicações. Por esta razão, foi desenvolvido de raiz um mecanismo

de suporte à invocação remota de rotinas a partir da interface de programação Winsock fornecida pelo sistema operativo.

Dados os objectivos principais do trabalho a que este relatório se refere, a grande preocupação foi colocada na concepção rápida de serviços de RPC que suportassem minimamente as necessidades essenciais do sistema de ficheiros distribuído. O ênfase foi, sim, colocado num desenvolvimento modular dos serviços, por forma a possibilitar uma fácil incorporação de melhorias posteriores.

O mecanismo de RPC concretizado efectua a chamada das rotinas remotas com uma semântica de execução *may-be* [7]. Após o envio de um pedido de chamada de uma rotina remota, o cliente fica à espera de uma resposta por parte do servidor com os resultados da rotina. É utilizado um temporizador para garantir a libertação do cliente ao fim de um determinado intervalo de tempo, em caso de uma falha na comunicação ou no processamento da rotina por parte do servidor. Deste modo, apesar do cliente saber que existiu uma falha na chamada remota, não pode determinar se a rotina foi correctamente executada no servidor, o que atribui às aplicações responsabilidades adicionais de detecção e resolução de faltas. Esta semântica de execução foi escolhida pela sua realização simples, tendo em conta os objectivos de concepção rápida dos mecanismos de RPC.

A transferência dos pedidos e respostas no âmbito dos serviços de RPC são efectuadas usando o protocolo UDP sobre IP. A escolha deste protocolo de transporte baseou-se principalmente na simplicidade de realização. O facto de não ser orientado à ligação permite que cada cliente ou servidor mantenha utilize apenas um *socket* UDP para a comunicação por RPC. Por outro lado, o facto de não existirem ligações liberta a biblioteca de RPC das funções de recuperação perante a quebra de ligação. Este protocolo de transporte, embora não ofereça garantias de fiabilidade do seu serviço [7], adequa-se à semântica *may-be*. Por outro lado, permite que os serviços de RPC possam ser oferecidos por servidores *stateless*, pois não obriga a manter estado das ligações estabelecidas.

4.3 Sistema de Ficheiros Distribuído

O processo de desenvolvimento do DFS que foi concretizado até à data da escrita deste relatório pode ser dividido em duas fases distintas. Na primeira fase, foi desenvolvido um DFS com uma arquitectura cliente-servidor, seguindo a abordagem discutida na secção 3.1 para o caso de uma info-appliance numa rede móvel que se baseia numa infra-estrutura situada numa rede fixa. Posteriormente, a segunda fase pretendeu incorporar o esquema de nomes baseado em *ad-hoc folders* no DFS desenvolvido anteriormente, de modo a atingir a arquitectura *peer-to-peer* referida na secção 3.1 para a utilização de *info-appliances* em rede *ad-hoc*.

Ambas as versões realizadas (versão cliente-servidor e versão *peer-to-peer* com *ad-hoc folders*) são ainda bastante rudimentares face aos às opções de desenho que foram delineadas na secção 3.2.

Nomeadamente, o sistema concretizado não inclui mecanismos de *caching* de ficheiros. Como tal, qualquer acesso de escrita ou leitura de um ficheiro remoto é sempre servido remotamente no servidor que armazena esse ficheiro. A semântica oferecida é, assim, diferente da semântica de sessão escolhida na secção 3.2, sendo próxima de uma semântica Unix[12] (secção 2.3.1), pois as escritas feitas a um ficheiro são imediatamente visíveis a todos os outros

[12] A única diferença face à semântica Unix é o facto de não garantir a propriedade da partilha do ponteiro de localização dentro de um ficheiro por múltiplos clientes (ver 2.3.1).

clientes. A ausência de *cache* nos clientes permite também que os servidores das versões concretizadas sejam *stateless*, pois não têm de manter estado necessário para a invalidação das *caches*.

O protocolo de comunicação entre clientes e servidores foi baseado no protocolo NFS. A escolha deste protocolo baseou-se principalmente na ampla divulgação e documentação deste protocolo. Por outro lado, este protocolo é orientado para servidores *stateless* e, consequentemente, adequado a ambas as versões do DFS que foram desenvolvidas. Finalmente, as rotinas de leitura e escrita de ficheiros do NFS manipulam blocos de ficheiros, uma vez que o protocolo prevê a utilização de *caches* com granularidade de blocos de ficheiros. Este facto é adequado a ambas as versões do DFS realizadas, pois as funções da API do sistema de ficheiros local também manipulam blocos de ficheiros e a *cache* de ficheiros completos não é ainda contemplada nessas versões. No entanto, foram efectuadas algumas alterações ao protocolo original. Entre estas:

- Na versão com *ad-hoc folders*, utilização de GUIDs como nomes lógicos de ficheiros, em vez dos nomes de ficheiros textuais utilizados pelo NFS.
- Alteração dos resultados retornados pela rotina remota ReadDir (ver Tabela 2, secção 4.3.2) para um funcionamento optimizado com o sistema de ficheiros NFS. Esta rotina passou a incluir também os atributos de cada ficheiro no resultado retornado.

4.3.1 Componente cliente

Na prática, a componente cliente é realizada sob a forma de uma *dynamic link library* (*dll*) que implementa as funções exportadas por um *Installable File System*. As funções do sistema de ficheiros distribuído são executadas no âmbito do espaço de endereçamento do processo do sistema operativo *device.exe*.

O carregamento inicial do sistema de ficheiros distribuído tem de ser efectuado de forma explícita por uma aplicação que chame a função sistema ActivateDevice. Só a partir deste momento é que a *dll* do Installable File System é carregada para o espaço de endereçamento do processo *device.exe* e é feito o *mount* do disco que ficará associado ao sistema de ficheiros distribuído. Como resultado da chamada à função MountDisk da componente cliente, é registado um novo volume, que albergará o espaço de nomes partilhado do sistema. É também nesta fase que é lançada a *thread* da componente servidora.

A componente cliente exporta a totalidade das funções que a interface do sistema de ficheiros do Windows CE põe ao dispor das aplicações. Desta forma é garantida a transparência da interface de programação disponibilizada às aplicações, uma vez que estas podem aceder ao sistema de ficheiros distribuído do mesmo modo que anteriormente utilizavam o sistema de ficheiros local.

Complementarmente, a componente cliente, como Installable File System, exporta também funções específicas para a sua manutenção por parte do FSD Manager. Entre estas estão as funções MountDisk, UnmountDisk e CloseVolume.

A Tabela 1 enumera e resume o significado de cada função exportada por parte do Installable File System. Estas são as funções que, através dos serviços do FSD Manager, suportarão a API do sistema de ficheiros do Windows CE.

MountDisk	Efectua o *mount* de um disco que contém o sistema de ficheiros definido pelo Installable File System. Esta função é chamada pelo FSD Manager quando é detectada a presença de algum novo hardware de armazenamento que esteja

	associado a este sistema de ficheiros. Por exemplo, a inserção de um PC Card com funções de armazenamento pode levar o FSD Manager a chamar esta função ao driver do sistema de ficheiros associado. O *mount* de um disco também pode ser originado por software através de uma chamada à função ActivateDevice. Deste modo é possível a existência de sistemas de ficheiros sem nenhum tipo de hardware associado, como é o caso de um sistema de ficheiros distribuído.
UnmountDisk	Efectua o unmount de um disco sobre o qual foi previamente chamada a função MountDisk. Ocorre quando o FSD Manager detecta a remoção do hardware referente ao disco ou quando a função sistema DeactivateDevice é chamada.
CreateDirectoryW	Cria uma directoria num volume do Installable File System.
RemoveDirectoryW	Elimina uma directoria num volume do Installable File System.
GetFileAttributesW	Obtem os atributos de um ficheiro ou directoria.
SetFileAttributesW	Altera os atributos de um ficheiro ou directoria.
DeleteFileW	Elimina um ficheiro.
MoveFileW	Altera o nome de um ficheiro ou directoria que residam num Installable File System. No caso de uma directoria, move também todos os seus filhos.
DeleteAndRenameFileW	Copia o conteúdo de um ficheiro para um ficheiro de destino, já existente no Installable File System. De seguida apaga o ficheiro de origem.
GetDiskFreeSpaceW	Obtém informação sobre a porção de espaço disponível num volume de um Installable File System.
Notify	Notifica o Installable File System de eventos relacionados com a suspensão/activação do computador.
RegisterFileSystemFunction	Regista junto do Installable File System uma função de notificação que deverá ser chamada pelo mesmo quando ocorrerem alterações de ficheiros ou do conteúdo de directorias. A função de notificação serve para assegurar a consistência de aplicações que apresentem o estado do Installable File System ao utilizador, como é o caso do Windows Explorer.
FindFirstFileW	Pesquisa o conteúdo de uma directoria do Installable File System à procura do primeiro ficheiro ou directoria cujo nome coincida com o nome especificado como argumento. No caso de encontrar, retorna informação sobre o ficheiro e uma search handle para posteriores pesquisas.
FindNextFileW	Continua uma pesquisa iniciada com uma chamada à função FindFirstFileW, retornando o próximo ficheiro ou directório encontrado na directoria do Installable File System.
FindClose	Fecha uma search handle.
CreateFileW	Cria, abre ou trunca um ficheiro do Installable File System, retornando o file handle respectivo. Pode também abrir uma directoria, retornando o seu file handle. O ficheiro ou directoria são especificados pelo seu nome. Esta função pode ser também utilizada com outros objectos, tais como pipes ou dispositivos de disco. No entanto, essas situações não se encontram no âmbito do DFS.
ReadFile	Lê um bloco de dados do ficheiro cujo file handle é passado por argumento. O início do bloco é indicado pelo ponteiro associado ao ficheiro aberto. A dimensão do bloco é indicada como argumento. Após a leitura, o ponteiro associado ao ficheiro é avançado para a posição após o final do bloco lido.
ReadFileWithSeek	Lê um bloco de dados do ficheiro cujo file handle é passado por argumento. O início do bloco e a sua dimensão são indicados como argumento. Após a leitura, o ponteiro associado ao ficheiro é avançado para a posição após o final do bloco lido.

WriteFile	Escreve um bloco de dados do ficheiro cujo file handle é passado por argumento. O início do bloco é indicado pelo ponteiro associado ao ficheiro aberto. A dimensão é indicada como argumento. Após a escrita, o ponteiro associado ao ficheiro é avançado para a posição após o final do bloco escrito.
WriteFileWithSeek	Escreve um bloco de dados do ficheiro cujo file handle é passado por argumento. O início do bloco e a sua dimensão são indicados como argumento. Após a escrita, o ponteiro associado ao ficheiro é avançado para a posição após o final do bloco escrito.
SetFilePointer	Move o ponteiro associado ao ficheiro de acordo com o deslocamento indicado como argumento.
GetFileSize	Retorna o tamanho, em bytes, do ficheiro associado à file handle passada como argumento.
GetFileInformationByHandle	Obtém informação sobre o ficheiro associado à file handle passada como argumento.
FlushFileBuffers	Limpa os buffers associados ao ficheiro indicado, obrigando a que toda a informação neles contida seja escrita de modo persistente.
GetFileTime	Obtém os tempos em que o ficheiro indicado foi criado, acedido e modificado pela última vez.
SetFileTime	Obtém os tempos em que o ficheiro indicado foi criado, acedido e modificado pela última vez.
SetEndOfFile	Move a posição final do ficheiro para a posição do ponteiro associado ao ficheiro aberto.
DeviceIOControl	Envia um código de controle directamente para o driver do dispositivo físico associado ao Installable File System. No caso particular do DFS, esta função não é suportada.
CloseFile	Fecha um file handle.
CloseVolume	Fecha um volume do Installable File System. Chamada para notificar o Installable File System de que um volume seu irá ser removido.

Tabela 1: Funções a exportar pela componente cliente

No momento em que este relatório foi escrito, apenas as funções MountDisk, UnmountDisk, CreateDirectoryW, CreateFileW, CloseFile, RemoveDirectoryW, GetFileAttributesW, SetFileAttributesW, FindFirstFileW, FindNextFileW, FindClose, ReadFile ReadFileWithSeek, WriteFileWithSeek e WriteFile estavam completamente concretizadas. Por motivos de insuficiência de tempo para o desenvolvimento do trabalho, as restantes funções não estão suportadas. Como resultado da chamada a essas funções, o sistema retorna um erro ERROR_NOT_SUPPORTED.

O único tipo de *caching* que é efectuado pela componente cliente é o relativo ao conteúdo das directorias, incluindo informação referente ao nome e aos atributos dos ficheiros e directorias que compõe os últimos directórios acedidos.

O acesso ao conteúdo dos ficheiros remotos, através das funções ReadFile e WriteFile são feitos de forma directa sobre os ficheiros originais remotos, sem qualquer utilização de *caching*. Esta decisão prende-se com razões de limitações de tempo para o desenvolvimento do trabalho.

4.3.2 Componente servidora

Como referido anteriormente, a componente servidora deve fornecer uma interface que seja o mais próximo possível daquela presente no NFS. As rotinas disponibilizadas pela componente servidora encontram-se apresentadas na tabela seguinte. Das rotinas incluídas na especificação do protocolo NFS [16], as rotinas relacionadas com a manutenção de *hard links* e *symbolic links* foram ignoradas, pois a semântica do sistema de ficheiros do Windows CE

não contempla esses conceitos. Também as rotinas que eram consideradas obsoletas para a versão 3 do protocolo foram descartadas.

GetAttr	Obtém os atributos de um ficheiro remoto, a partir do seu *file handle*.
SetAttr	Altera os atributos de um ficheiro remoto, a partir do seu *file handle*.
LookUp	Retorna um *file handle* e os atributos do ficheiro remoto cujo nome é dado como argumento.
Read	Lê uma porção do conteúdo de um ficheiro remoto, começando no *offset* indicado e com a dimensão indicada. Os atributos do ficheiro após a leitura também são retornados.
Write	Altera uma porção do conteúdo de um ficheiro remoto, começando no *offset* indicado e com a dimensão indicada. Os atributos do ficheiro após a escritatambém são retornados.
Create	Cria um novo ficheiro na directoria remota especificada como argumento.
Remove	Elimina um ficheiro remoto.
Rename	Altera o nome de um ficheiro remoto.
MkDir	Cria uma nova directoria remota dentro da directoria indicada como argumento.
RmDir	Elimina uma directoria remota.
ReadDir	Retorna um número variável de entradas do conteúdo de uma directoria remota dada como argumento. O número máximo de entradas a retornar é indicado como argumento. No caso de ser necessário efectuar outras chamadas à rotina ReadDir para consultar o restante conteúdo da directoria, é fornecido um *cookie* que permite ao servidor determinar em que ponto terminou a última pesquisa.
StatFS	Retorna atributos do sistema de ficheiros, tais como o tamanho óptimo de transferência de dados nas rotinas Read e Write, número de blocos utilizados no sistema de ficheiros ou número de blocos livres no sistema de ficheiros.

Tabela 2: Rotinas remotas disponibilizadas pela componente servidora

Mais uma vez, por limitações de tempo para a concretização do trabalho, algumas rotinas não foram completamente realizadas. Entre estas encontram-se as rotinas LookUp, Create, Remove, Rename, MkDir, RmDir e StatFS.

A componente servidora é iniciada quando é feito o *mount* do sistema de ficheiros distribúido, isto é, durante a execução da função MountDisk por parte da componente cliente. Esta componente é lançada como uma *thread* do processo do sistema operativo *device.exe*, que tem privilégios de se executar em modo núcleo. Isto é possível porque a execução das funções do Installable File System é feita sob o espaço de endereçamento do processo *device.exe*. Logo, a criação de uma *thread* durante a chamada MountDisk cria uma *thread* desse processo.

Para servir muitas das rotinas enumeradas atrás, a componente servidora precisa de efectuar chamadas do sistema de ficheiros local para aceder aos dados partilhados que aí estão armazenados. Como a *thread* da componente servidora tem privilégios para correr em modo núcleo, a chamada das funções do sistema de ficheiros local pode ser efectuada directamente e sem a utilização das rotinas de interface que existem normalmente para permitir a chamada de funções sistema a partir do modo utilizador [44]. Esta optimização permite ganhos consideráveis no desempenho da componente servidora.

4.3.3 Dificuldades do processo de desenvolvimento do sistema

Muitas das decisões da arquitectura do sistema que foram planeadas não alcançaram a respectiva concretização até à data da escrita deste relatório. Tal deve-se sobretudo a limitações de tempo que inviabilizaram a realização de todos os pontos previstos no Relatório Intercalar.

Os contratempos de um processo de instalação e configuração da plataforma Windows CE relativamente demorado e, principalmente, da necessidade de realização de uma camada de suporte ao RPC obrigaram a iniciar o desenvolvimento efectivo do sistema de ficheiros distribuído com um atraso significativo.

Para além deste facto, o processo de instalação e teste de cada nova versão do sistema de ficheiros distribuído revelou-se extremamente lento. Em particular, cada tentativa de compilação e carregamento de uma nova versão do projecto implicava períodos de espera de cerca de 4min50seg até que os testes pudessem ser iniciados. Tal deve-se ao facto de esta tarefa incluir as fases de carregamento de uma nova imagem do sistema operativo e da respectiva inicialização.

Posteriormente no processo de desenvolvimento, foi utilizada outra abordagem para o teste de novas versões do projecto que permitiu reduzir substancialmente os tempos de carregamento do sistema. Em vez de efectuar o carregamento de uma nova imagem do sistema operativo, apenas é feita a reinicialização da imagem já carregada. Nesta solução, os ficheiros executáveis do projecto não eram incluídos na imagem do sistema operativo. Em vez disso, ficavam armazenados na estação de desenvolvimento e eram acedidos através da funcionalidade do Platform Builder de carregamento dinâmico de ficheiros pela rede através da componente de *debug*, *CESH*. Deste modo, a compilação de novas versões do trabalho apenas se reflectia nos ficheiros armazenados na estação de trabalho, não obrigando à utilização de uma nova imagem do sistema operativo.

Finalmente, a inicialização do *Installable File System* realizado trouxe algumas dificuldades que obrigaram um esforço de alguns dias para a sua resolução. Em concreto, a arquitectura prevista para os *Installable File Systems* pressupõe a sua associação a um dispositivo físico (por exemplo, uma *pc card*), onde é localizado o repositório físico do sistema de ficheiros. Como consequência, o FSD Manager recebe, como argumento de inicialização do *Installable File System*, uma referência para o *driver* do dispositivo físico correspondente. No caso particular de um sistema de ficheiros distribuído, não existe nenhum dispositivo físico que lhe esteja directamente associado. Para solucionar a ausência de uma *driver* de um dispositivo físico, foi utilizada a *RAMDisk block driver* para invocar a inicialização do *Installable File System*. Esta *driver*, embora seja inicializada, nunca é utilizada pelo *Installable File System*. Deste modo consegue-se uma inicialização e funcionamento correcto do *Installable File System* através do FSD Manager.

Obviamente, tal como será indicado no capítulo 5, a evolução do DFS ao longo do próximo ano deverá colmatar as lacunas acima indicadas.

4.4 Avaliação

O sistema que foi realizado não tem todas as funcionalidades tipicamente suportadas pelos sistemas de ficheiros. Em concreto, algumas funções da interface de programação do sistema de ficheiros do Windows CE 3.0 não foram completamente desenvolvidas, pelo que não são disponibilizadas à camada aplicacional. Nestes casos, o FSD Manager encarrega-se de

devolver um erro de ERROR_NOT_SUPPORTED às aplicações que efectuaram a chamada à função não disponível no sistema de ficheiros.

Por esta razão, muitas das aplicações de uso frequente existentes no Windows CE, tais como o processador de texto Pocket Word ou o navegador WWW Internet Explorer, não puderam ser testadas correctamente sobre o sistema de ficheiros distribuído que foi realizado.

Uma avaliação do funcionamento do sistema realizado sob cargas que reflectissem uma utilização realista de um sistema de ficheiros foi assim impossibilitada. Os testes que puderam ser efectuados sobre o sistema consistiram em interacções simples que apenas fizessem uso do subconjunto das funções disponíveis. A maioria destes testes foi feito através da linha de comandos do sistema operativo, onde se executavam comandos simples de manipulação e leitura do sistema de ficheiros. As funcionalidades de criação de ficheiros e directorias remotos, listagem do conteúdo de directorias remotas e acesso de escrita e leitura a ficheiros remotos foram correctamente verificadas em ambas as versões do DFS (versão para redes móveis ligada a infra-estrutura fixa e versão para redes *ad-hoc*) que foram realizadas.

5 Conclusões e trabalho futuro

As recentes evoluções tecnológicas, quer ao nível do poder computacional e da capacidade de armazenamento dos computadores móveis, quer ao nível das tecnologias de comunicação sem fios, tornam o paradigma da computação móvel numa realidade prometedora. A possibilidade de partilha de dados a partir de qualquer local, em particular através de sistemas de ficheiros distribuídos, é um objectivo altamente atractivo. Contudo, o universo computacional definido pelos computadores móveis é bastante diferente daquele que é tipicamente encontrado em redes fixas. As características específicas dos ambientes de computação móvel arrastam consigo novos desafios para os quais os DFSs convencionais, orientados para redes fixas, não são soluções eficazes. É necessário, pois, a utilização de DFSs que possuam conjunto de propriedades que os tornem adaptados aos ambientes móveis.

O presente documento apresentou, como resultado de uma análise crítica a soluções de DFSs já existentes, um conjunto de opções de arquitectura e desenho que deverão contribuir para a concepção de um DFS adequado às características inerentes aos ambientes móveis. Os resultados obtidos basearam-se numa análise que abordou dois ambientes distintos de utilização de dispositivos móveis em rede. Por um lado, considerou-se a utilização de *info-appliances* baseadas na existência de uma infra-estrutura fixa onde residem os servidores do sistema. Por outro lado, foi ponderada a utilização das *info-appliances* em redes *ad-hoc*.

Para o caso do funcionamento em redes *ad-hoc*, foi também proposto um esquema de nomes lógicos que possui um conjunto de vantagens que o tornam apropriado para um DFS orientado para esses ambientes.

Finalmente, foi desenvolvido um protótipo sobre o sistema operativo Windows CE que pretendeu incorporar as soluções que foram propostas para os ambientes móveis. Devido a limitações de tempo, associadas principalmente a dificuldades registadas na instalação do ambiente de desenvolvimento e na realização do protótipo, a solução concretizada apenas reflecte algumas das funcionalidades planeadas. Por esta razão, a avaliação efectuada ao protótipo não permite tirar conclusões quanto à eficácia das soluções propostas.

Como trabalho futuro, a completar no âmbito do último ano de mestrado integrado, pretendo completar a realização do protótipo do sistema planeado. Essa realização consistirá no desenvolvimento de um DFS que possua as opções de desenho citadas na secção 3, nomeadamente em relação aos aspectos de arquitectura, esquema de nomes, gestão de *cache* e segurança. O protótipo obtido deverá ser alvo de uma avaliação cuidada que verifique a eficácia do sistema desenvolvido sob cargas e condições típicas de ambientes móveis. Por fim, deverão ser estudadas e aplicadas eventuais optimizações ao sistema, nomeadamente ao nível de:

- adaptabilidade a ambientes móveis;
- políticas de gestão de *cache* adaptadas para ambientes móveis, incluindo aspectos como granularidade dos dados em *cache*, política de modificação, validação ou localização;
- suporte à autonomia dos nós através do funcionamento em operação desligada; e
- mecanismos de redução do tráfego de comunicação entre nós do DFS.

6 Referências

[1] G. Moore, Cramming more components onto integrated circuits. *Electronics, Volume 38, Nº 8, 19 de Abril de 1975.*

[2] Brief Tutorial on IEEE 802.11 Wireless LANs. Intersil, Fevereiro de 1999.

[3] Bluetooth Special Interest Group. Specification of the Bluetooth System, Version 1.1. 22 de Fevereiro de 2001.

[4] A. Kaminsky. Infrastructure for Distributed Applications in Ad hoc Networks of Small Mobile Wireless Devices. 22 de Maio de 2001.

[5] M. Singh. Peering at *Peer-to-peer* Computing. *IEEE Internet Computing, Janeiro, Fevereiro de 2001.*

[6] S. Corson. Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations.

[7] J. Marques e P. Guedes, Tecnologia de Sistemas Distribuídos, 2ª Edição, FCA, 1998.

[8] E. Levy e A. Silberschatz, Distributed File Systems: Concepts and Examples. *ACM Computing Surveys, Vol. 22, Nº 4, Dezembro 1990.*

[9] A. Barak et al. AFS, BFS, CFS … or Distributed File Systems for UNIX. *Em European UNIX Users Group Conference Proceedings.* 22-24 de Setembro de 1986.

[10] P. Leach, and D. Perry. CIFS: A Common Internet File System.

[11] J. Murray. Inside Microsoft Windows CE.

[12] Richard Guy, Peter Reicher, David Ratner, Michial Gunter, Wilkie Ma, and Gerald Popek. Rumor: Mobile Data Access Through Optimistic *Peer-to-peer* Replication. *Em Proceedings: ER'98 Workshop on Mobile Data Access.* 1998

[13] H. Yu, and A. Vahdat. Design and Evaluation of a Continuous Consistency Model for Replicated Services.

[14] J. Neuman et al. Kerberos: An Authentication Service for open Network Systems. *Em Proceedings Winter Usenix.* 1998, pág. 191-202.

[15] M. Weinstein et al. Transactions and synchronization in a distributed operation system. *Em Proceedings of the 10th Symposium on Operating Systems Principles.* Dezemvro de 1985.

[16] RFC1094, NFS: Network File system specification, www.faqs.org/rfcs/rfc1094.html

[17] M. Hill et al. Design Discussions in SPUR. *IEEE Computing, 19.* Novembro de 1986.

[18] J. Howard et al. Scale and performance in a distributed file system. *Em ACM Trans. Comp. Syst. 6.* Fevereiro de 1988.

[19] SMB & CIFS Protocol. Zzap White Paper – SMB.

[20] SAMBA - Opening Windows to a Wider World. http://www.samba.org. Abril 2000.

[21] J. Kistler, and M. Satyanarayanan. Disconnected Operation in the Coda File System.

[22] Muthitacharoen, B. Chen, and D. Mazières. A Low-bandwidth Network File System.

[23] FIPS 180-1. Secure Hash Standard.

[24] T. E. Anderson et al. Serverless Network File Systems.

[25] R. Y. Yang, and T. E. Anderson. XFS: A Wide Area Mass Storage File System.

[26] D. Patterson et al. A Case for Reduntant Arrays of Inexpensive Disks (RAID).*em International Conference on Management of Data.* Junho de 1988.

[27] J. Hartman e J. Ousterhout. The Zebra Striped Nelwork File System. *Em ACM Trans. on Computer Systems.* Agosto 1995.

[28] Demers et al. The Bayou: Support for Data Sharing among Mobile Users.

[29] A. Demers et al. Epidemic algorithms for replicated database maintenance. *Em Proceedings Sixth Symposium on Principles of Distributed Computing.* Agosto de 1987.

[30] D. B. Terry et al. Session Guarantees for Weakly Consistent Replicated Data.

[31] J. K. et al. Oceanstore: An Architecture for Global-Scale Persistent Storage.

[32] D. Mazières et al. Separating key management from file system security. *Em Proceedings of ACM SOSP.* 1999.

[33] R. Guy et al. Implementation of the Ficus replicated file system. *Em Usenix Conference Proceedings.* Junho de 1990.

[34] D. Ratner. Selective replication: Fine-grain control of replicated files. Technical Report CSD-950007, University of California, Los Angeles. Março de 1995.

[35] P. Druschel, and A. Rowston. PAST: A large-scale, persistent *peer-to-peer* storage utility.

[36] P. Druschel, and A. Rowston. PASTRY: Scalable, distributed object location and routing for large-scale *peer-to-peer* systems.

[37] The Gnutella protocol specification. 2000.

[38] I. Clarke et al. Freenet: A distributed anonymous information storage and retrieval system. *Em workshop on Design Issues in Anonymity, unobservality, and pseudonymity – a proposal for terminology.* Abril de 2001.

[39] Napster. http://www.napster.com.

[40] M. Satyanarayanan. Mobile Information Acces. *IEEE Personal Communications*. Fevereiro de 1996.

[41] Cary G. Gray e David R. Cheriton. Leases: An efficient fault-tolerant mechanism for distributed file *cache* consistency. *Em Proceedings of the 12th ACM Symposium on Operating Systems Principles.* Dezembro de 1989.

[42] Research release: Palm retains mobile lead by units, Compaq top by value, http://www.canalys.com/pr/r2002041.htm

[43] Microsoft Windows CE Platform Builder 3.0 Library.

[44] J. Alves Marques e P. Guedes, Fundamentos de Sistemas Operativos, 1992, Editorial Presença.

[45] Bakre e B. Badrinath, Reworking the RPC paradigm for mobile clients.