UNIVERSIDADE ESTADUAL DE CAMPINAS
Faculdade de Engenharia Elétrica e de Computação
Departamento de Engenharia de Computação e Automação Industrial

FEEC
Faculdade de Engenharia Elétrica e de Computação

Um Sistema de Arquivos Distribuído para Computadores Maciçamente Paralelos Virtuais

Autor:
Juan Carlos Hidalgo Costa

Orientador:
Prof. Dr. Marco Aurélio Amaral Henriques

Banca Examinadora:
Prof. Dr. Ricardo Ribeiro Gudwin
Prof. Dr. José Raimundo de Oliveira
Prof. Dr. Célio Cardoso Guimarães

Dissertação apresentada à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas como parte dos requisitos necessários para a obtenção do título de Mestre em Engenharia Elétrica.

Campinas, São Paulo, Brasil
6 de Agosto, 2002

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

H53u

Hidalgo Costa, Juan Carlos
Um sistema de arquivo distribuído para computadores maciçamente paralelos virtuais / Juan Carlos Hidalgo Costa.--Campinas, SP: [s.n.], 2002.

Orientador: Marco Aurélio Amaral Henriques.
Dissertação (mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Arquivo. 2. Processamento paralelo (Computadores). 3. Java (Linguagem de programação de computador). 4. Internet (Redes de computação). I. Henriques, Marco Aurélio Amaral. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Resumo

Os computadores conectados pela Internet oferecem em conjunto um grande poder de cômputo, o qual deve continuar crescendo nos próximos anos. Eles podem ser vistos como um **Computador Maciçamente Paralelo Virtual** com memória distribuída que pode ser usado na resolução de problemas de grande porte. Existem várias propostas que visam tirar proveito da Internet como um computador virtual, utilizando Java como linguagem independente de plataforma. Entretanto, a maior parte destes projetos não trata, ou trata de forma superficial, a necessidade de se ter um sistema de arquivos que garanta a viabilidade e eficiência do processamento paralelo. Este trabalho propõe um sistema de arquivos distribuído baseado em grupos de servidores e voltado a plataformas para o processamento maciçamente paralelo na Internet. São propostos mecanismos para atender os requisitos fundamentais de sistemas deste tipo, eliminando as principais deficiências dos sistemas de arquivos convencionais. São apresentados e discutidos os resultados obtidos nos testes de uma implementação de referência do sistema de arquivos sobre JOIN, uma plataforma de processamento maciçamente paralelo virtual baseada na Internet. Esta implementação se mostrou confiável, robusta e aumentou a versatilidade da plataforma JOIN.

Palavras-chave: sistema de arquivos, processamento distribuído, Java, Internet

Abstract

The total computing power offered by all computers connected to Internet is huge and increasing. Since the birth of the World Wide Web, new proposals have been made on how to take advantage of the enormous computing power represented by these computers. The proposals are based on the availability of WWW browsers to access resources distributed in the network and on the proliferation of Java as a platform independent language. The computers are grouped in a kind of massively parallel virtual computer, aimed at solving large problems. File systems are a necessary - and often forgotten - feature of these virtual machines. File systems for worldwide virtual machines should be efficient, highly available and fault tolerant. The virtual machines proposed so far either have no File System or provide very simple and inefficient solutions. This work proposes a distributed file system based on Groups of Servers, which was implemented and tested on top of JOIN, a platform for massively parallel processing on Internet. The test results showed that the file system is reliable, robust and makes the parallel platform more versatile.

Keywords: file systems, distributed processing, Java, Internet

Agradecimentos

Quero agradecer a todos que, de uma forma ou de outra, ajudaram na realização deste trabalho.

- Aos meus pais, por sempre terem incentivado minha superação profissional e por compreenderem minha decisão de vir estudar tão longe de casa.
- Aos meus amigos Eduardo e Sahudy, porque sem a ajuda deles nunca poderia ter chegado a este país e por me oferecerem seu apoio incondicional em todos os momentos.
- Ao meu orientador Marco Aurélio, que, apesar das suas múltiplas obrigações e excesso de trabalho, sempre conduziu com muita seriedade e dedicação esta pesquisa.
- À minha família em Cuba, que apesar da saudade, tem me dado sempre o maior apoio e carinho.
- A todos os meus amigos, pela paciência e grande amizade que demonstraram sempre, tornando mais tolerável a separação da família.
- À FAPESP, pelo apoio financeiro, sem o qual nada disto poderia ter sido feito.
- À Faculdade de Engenharia Elétrica e de Computação da UNICAMP, especialmente aos laboratórios do DCA, DSEE e SIFEEC, por me oferecerem seus recursos para a realização dos testes necessário.
- À minha esposa Daynet, por sempre estar ao meu lado e me suportar com muito amor e paciência.
- E por último, ao meu filhinho Edú, por ser o que há de mais importante na minha vida e por me dar a força necessária para continuar adiante.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 A Internet como um Computador Maciçamente Paralelo Virtual

A indústria da computação e das tecnologias de informática, apesar de ser muito jovem se comparada com outras (automobilística, têxtil, etc.), teve um progresso bastante significativo. Além disso, nos últimos anos, as tecnologias de redes também têm se desenvolvido aceleradamente, aumentando a velocidade dos enlaces e melhorando os softwares de rede e a qualidade do serviço oferecido. Os fatos acima mencionados fizeram com que os sistemas computacionais experimentassem mudanças radicais após seu surgimento. Como exemplo tem-se que os sistemas centralizados, formados por um computador só (geralmente um enorme e custoso *mainframe*), deram lugar aos sistemas distribuídos, que exploram os recursos oferecidos pelas redes e pelos computadores a elas conectados. Segundo Tanenbaum [Tan95], "um sistema distribuído é uma coleção de computadores independentes, mas que são vistos pelos usuários do sistema como se fosse um só computador". A consolidação das redes de computadores introduziu no mundo da computação este novo paradigma do processamento distribuído, como uma conseqüência lógica da possibilidade de compartilhar recursos através de uma rede. Os recursos necessários para a realização de uma determinada tarefa não precisam mais estar localizados fisicamente no computador que executa a tarefa; basta que sejam acessíveis a partir desse computador. A tarefa que solicita os recursos é chamada de cliente, e a tarefa que atende a solicitação é chamada de servidor. Este paradigma de acesso a recursos é conhecido como cliente-servidor, e ainda constitui a base da maioria dos sistemas distribuídos atuais. Entretanto, ele não explora completamente as capacidades de um conjunto de computadores conectados em rede. Em particular, ele não oferece uma plataforma que facilite o uso de todos esses computadores para a solução de um único problema de escala maior.

Na solução de problemas computacionais de grande porte adota-se geralmente o processamento paralelo [AG94]. Este processamento é baseado no fato de algumas aplicações poderem ser divididas em um conjunto de tarefas menores que podem ser executadas em paralelo em processadores localizados em um mesmo computador ou em computadores diferentes, aumentando o desempenho da aplicação. Um tipo particular de processamento

paralelo é o processamento maciçamente paralelo. Aplicações maciçamente paralelas são formadas por um grande número de tarefas e geralmente requerem computadores paralelos também com um grande número de processadores. Exemplos de aplicações deste tipo são aplicações de previsão numérica do tempo, cálculos de engenharia, computação gráfica, criptoanálise, seqüenciamento de DNA, dentre outras. Apesar do grande poder computacional que pode ser obtido com máquinas paralelas, muitas vezes elas não conseguem resolver satisfatoriamente alguns destes problemas. Além disso, elas são extremamente caras, requerem conhecimentos técnicos específicos para serem usadas, suas aplicações raramente são portáveis e seu hardware fica obsoleto em pouco tempo.

Uma alternativa à construção de computadores paralelos é a utilização de vários computadores conectados entre si por uma rede. Apesar das taxas de comunicação neste caso serem geralmente menores que na comunicação entre processadores dentro de um computador paralelo, a razão custo-benefício torna-se bastante atraente pelo fato de se usar uma infra-estrutura de hardware já disponível. Já são comuns os programas que viabilizam o uso conjunto de computadores conectados em rede como um computador paralelo, o que é normalmente chamado de computador paralelo virtual. A maioria destes programas tem sua utilização restrita a redes locais e/ou a um número limitado de computadores, o que dificulta a implementação de aplicações maciçamente paralelas.

A rede de computadores mais conhecida e que envolve o maior número de computadores é a Internet. De acordo com pesquisas de MIDS (*Matrix and Information Directory Services, Inc.*) [mat02], o número de computadores conectados à Internet está na ordem de dezenas de milhões e continua crescendo aceleradamente. Outro estudo, realizado por Fox e Furmansky [FF97], revela que o desempenho conjunto potencial de todos os computadores conectados à Internet é superior ao desempenho oferecido pelo computador paralelo real mais avançado. Estes dados indicam o potencial de se utilizar a Internet na resolução de aplicações maciçamente paralelas.

Uma rede de computadores como a Internet pode ser vista como um Computador Maciçamente Paralelo Virtual (MPVC - *Massively Parallel Virtual Computer*) com memória distribuída, formado por computadores independentes, autônomos, heterogêneos, distribuídos geograficamente e conectados de forma irregular mediante uma rede lenta e pouco confiável. A diferença fundamental destes computadores virtuais com os computadores paralelos reais é que os primeiros têm uma latência alta nas comunicações, o que dificulta a execução de aplicações que precisem de muitas comunicações entre seus componentes. No entanto, os MPVCs podem oferecer um poder de cômputo muito grande, que aumenta de forma natural à medida que novos computadores são adicionados ao computador virtual e/ou seus processadores e enlaces são melhorados. Além disso, a construção de um MPVC usando computadores conectados pela Internet é uma alternativa barata, pois usa recursos que já existem. Muitas das deficiências nas comunicações são conseqüências do curto tempo de vida da Internet e se atenuarão à medida que novas tecnologias e softwares forem desenvolvidos. Como exemplo tem-se a utilização da infra-estrutura da televisão a cabo [Sha98] para levar a Internet até os computadores pessoais, os quais constituem uma parcela significativa do total de computadores conectados na rede mundial. Portanto, o uso de Internet como rede de interconexão para um MPVC está se tornando uma alternativa

cada vez mais atraente.

1.2 Arquiteturas para o Processamento Paralelo

Existem várias propostas de processamento paralelo usando computadores conectados em rede. As mais conhecidas são geralmente usadas em redes locais, com implementações que conseguem obter bons resultados, e algumas delas tornaram-se padrões muito utilizados. Destacam-se sistemas como NOW [TEAtNt95], PVM [GBD+94, GKP96], MPI [SOHL+96, KH95], Condor [TL95, ELD+96], Legion [GW96, LG96], ATLAS [BBB96, BJK+95] e Globus [FK98, CFK+98]. No entanto, de forma geral, estas propostas são dependentes de plataformas, exigem a instalação e manutenção de softwares específicos em todos os computadores participantes, requerem que todos eles sejam de um mesmo tipo ou que suas versões de sistemas operacionais sejam compatíveis. Além disso, estes sistemas foram projetados para computadores com um certo nível de sofisticação (geralmente sistema operacional Unix, conexão permanente à rede, alto desempenho), o que restringe significativamente o número de computadores que podem participar em tais sistemas. Isto faz com que seu uso para resolver problemas de grande escala seja difícil.

Mais recentemente, com o surgimento da WWW (*World Wide Web*) [BLCL+94] e da linguagem de programação independente de plataforma Java [Nau97, HSW97], novos sistemas têm sido desenvolvidos para utilizar a Internet como um computador virtual, tirando proveito do uso de navegadores para WWW e de suas possibilidades de executar *applets Java* [Sun97b]. Tais navegadores estão disponíveis em praticamente todos os computadores ligados à Internet, os quais podem formar um grande MPVC sem exigir nenhum tipo de instalação de software extra por parte de seus proprietários. Sempre que um computador se conecte ao MPVC, ele poderá receber automaticamente todo o software necessário para que possa se comunicar com os demais integrantes do sistema paralelo e executar as tarefas que lhe forem atribuídas. Destacam-se sistemas como ParaWeb [BSST96], SuperWeb [AISS97a], PopCorn [NLRC97, RN98], KnittingFactory [BKKMK98], JavaParty [PZ97], JPVM [Fer98], Javelin [ONOCCES99, CCI+97] e JoiN [Hen99].

1.3 Arquitetura de Referência

A maior parte dos sistemas citados não trata, ou trata de forma superficial, da necessidade de se ter um sistema de arquivos que garanta a viabilidade e eficiência na execução das aplicações paralelas. ATLAS, por exemplo, provê um sistema de arquivos global, baseado em URLs e com cache local coerente, mas que somente permite acessos de leitura em seus servidores, o que restringe muito a utilidade do sistema. Em processamento maciçamente paralelo, um sistema de arquivos capaz de manipular de forma eficiente as informações geradas pelas aplicações é um elemento imprescindível.

Há uma clara tendência em se explorar o poder computacional existente na Internet, especialmente através dos novos recursos oferecidos pela WWW. Como existem inúmeras maneiras de se fazer esta exploração, também são diversas as opções para se chegar a um

sistema de arquivos para dar suporte a tais iniciativas. Entretanto, alguns pontos já se evidenciam, como o uso da linguagem Java para atenuar as dificuldades de se trabalhar com computadores de arquiteturas distintas. É recomendável que uma proposta de sistema de arquivos para um computador maciçamente paralelo virtual baseado na Internet seja também baseada nesta linguagem e tenha um potencial de uso maior por parte dos novos sistemas paralelos que têm surgido.

Apesar da adoção de uma linguagem como Java significar também a adoção de uma arquitetura (representada pela máquina virtual Java, JVM - *Java Virtual Machine*) e de um paradigma de programação (programação orientada a objetos), ela não é suficiente para definir todas as possibilidades. É necessário que um modelo básico de funcionamento do MPVC esteja delineado para facilitar o planejamento do sistema de arquivos para o mesmo. Como um MPVC baseado em Internet (na WWW) se propõe a agregar o poder de cômputo de muitas máquinas, é desejável que estas possam se incorporar ao computador virtual de uma forma bem simples e transparente. Uma possível forma é aquela apresentada na Figura 1.1.

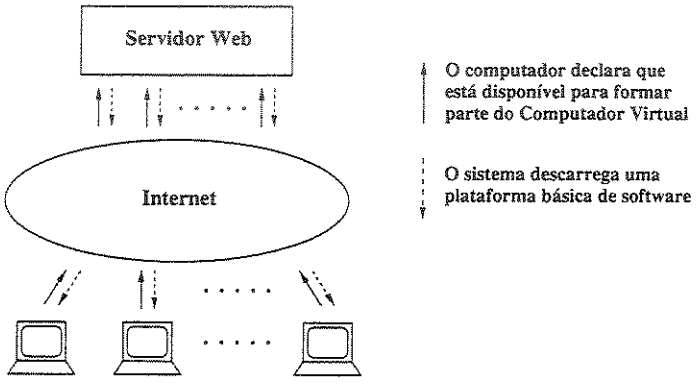

Figura 1.1: Arquitetura básica de um computador virtual.

Um Servidor Web, dedicado ao sistema maciçamente paralelo virtual, serve como portal de entrada para este. Se o proprietário de um computador quiser participar do computador virtual, basta estabelecer uma conexão com este Servidor Web e declarar que seu computador está disponível para tomar parte no processamento paralelo. Nesse momento, do servidor é descarregada uma pequena plataforma básica que fornece as facilidades mínimas para a comunicação com outros componentes do sistema e para a execução de aplicações paralelas no computador. O computador torna-se então parte do MPVC e pode receber tarefas das aplicações que serão executadas no computador paralelo virtual. Estas aplicações paralelas estão formadas por um conjunto de tarefas, que uma vez submetidas ao sistema, são distribuídas entre os diferentes computadores que o formam. O proprietário pode retirar seu computador do MPVC quando quiser. Neste ponto, todas as tarefas que estiverem sendo executadas nesse computador têm que migrar para outros computadores disponíveis para continuar sua execução normal ou começar desde o início.

Esta arquitetura é baseada na ampla disponibilidade de interfaces simples para se ter acesso aos recursos da Internet (navegadores para WWW) e na proliferação de Java como

linguagem independente de plataforma. Praticamente todos os computadores conectados à Internet têm navegadores para WWW e a tendência atual é que todos possam executar *applets Java*. Desta forma, qualquer computador conectado à Internet tem a possibilidade de participar do computador paralelo virtual.

1.4 Trabalhos Afins

Uma das áreas com grande atividade de pesquisa em computação é a área de sistemas de Arquivos Distribuídos [CDK94]. Atualmente existem muitas propostas destes sistemas, algumas delas amplamente usadas e que, em muitos casos, constituem padrões de fato. Tais sistemas podem ser classificados de acordo com a rede onde são utilizados [Tan96]. Esta classificação leva a três categorias principais, mostradas a seguir.

Sistemas de arquivos distribuídos para redes locais [1]:
sistemas de arquivos com ótimo desempenho, cujos enlaces são de alta velocidade e a latência nas comunicações é desprezível. Neste tipo de rede eles são rápidos, eficientes e simples de serem usados. No entanto, estes sistemas não são muito escaláveis e, portanto, inadequados para a manipulação de arquivos em uma escala maior. Destacam-se sistemas como NFS (*Network File System*) [MBKQ96, SGK+85], AFS (*Andrew File System*) [How88, CD93], DFS (*Distributed File System*) [HKM+88], Novell Netware [Nov98d, Nov98e] e Windows NT [SC98, Moh99].

Sistemas de arquivos distribuídos para redes geograficamente distribuídas [2]:
sistemas de arquivos para redes mais lentas e instáveis. Geralmente são sistemas mais escaláveis, tolerantes a falhas, apresentam um espaço de nomes global e implementam mecanismos de replicação de servidores e uso intensivo de cache nos clientes. Estes mecanismos permitem manipular um maior número de arquivos de uma forma mais eficiente. Entre os sistemas de arquivos mais destacados temos Coda [SKK+90, KS91], Bayou [DPS+95], Sprite [Wel90, BO91] e Ficus [GHM+90, Guy91].

Sistemas de arquivos distribuídos para a Internet :
sistemas de arquivos geralmente baseados nos protocolos HTTP (*HiperText Transfer Protocol*) [BLFF96] e FTP (*File Transfer Protocol*) [PR91]. O objetivo destes sistemas é aproveitar o potencial da Internet, tentando ser tão ou mais eficientes que os protocolos acima citados. A Internet já tem um espaço de nomes global, o URL (*Universal Resource Locators*) [BLMM94], o qual pode ser usado no sistema de nomes dos arquivos. Podemos citar sistemas de arquivos como WebNFS [Sun97a, Sun96], WebFS [VEA96, YCE+97], Ufo [AISS97b], Jade [Rao91] e IFS (*Internet File System*) [RC95].

Um estudo detalhado dos sistemas de arquivos acima citados pode ser encontrado no Apêndice A. Por este estudo pode se concluir que nenhum dos sistemas de arquivos dis-

[1] LAN - Local Area Network
[2] WAN - Wide Area Network

tribuídos citados satisfaz todas as exigências de um MPVC baseado na Internet. Suas deficiências principais são mostradas a seguir.

- Muitos deles não têm a escalabilidade suficiente para um processamento maciçamente paralelo, onde milhares de computadores podem trabalhar em conjunto para resolver um mesmo problema.
- Necessitam da instalação, manutenção e configuração de softwares específicos nos computadores que formam a máquina paralela, o que exige maiores conhecimentos técnicos de seus usuários, dificultando uma utilização ampla e irrestrita.
- Não oferecem reconfiguração automática quando um computador entra ou sai do MPVC.
- A maioria das implementações são realizadas sobre Unix e não podem ser usadas em outras plataformas, o que diminui o número de computadores candidatos a participar no sistema.

1.5 Objetivos e Estrutura da Dissertação

O objetivo principal deste trabalho é conceber um sistema de arquivos distribuído que satisfaça de forma eficiente as necessidades de um Computador Maciçamente Paralelo Virtual formado por computadores conectados pela Internet, como o mostrado na Figura 1.1. Este sistema de arquivos deve garantir a viabilidade de se executar aplicações maciçamente paralelas com bons resultados. O problema principal a se enfrentar é o fato de que os dados a serem processados e os resultados do processamento têm que ser armazenados em um lugar acessível por todas as tarefas das aplicações paralelas distribuídas nos computadores que formam o MPVC. Um ponto possível seria o Servidor Web. No entanto, abordagens distribuídas foram consideradas, baseadas na possibilidade de alguns dos computadores do computador virtual oferecerem seus recursos por um determinado período de tempo, evitando que apareçam pontos de gargalo no sistema.

O restante do texto está organizado como descrito a seguir. No Capítulo 2 são descritos os principais requisitos que um sistema de arquivos para este tipo de plataforma tem que satisfazer. São apresentadas e comparadas as principais propostas consideradas neste trabalho. Com base nesta comparação, é escolhida a proposta mais apropriada para o sistema de arquivos desejado. O capítulo também descreve as especificações gerais do sistema de arquivos distribuído proposto. O Capítulo 3 descreve os detalhes de uma implementação feita para JOIN, um sistema de processamento maciçamente paralelo na Internet. A descrição dos testes feitos usando esta implementação, assim como uma discussão dos resultados obtidos nos mesmos, são mostrados no Capítulo 4. Por último, o Capítulo 5 contém as conclusões do trabalho, mostrando suas principais contribuições e aspectos que ainda podem ser trabalhados no futuro. Além disso, foram adicionados 4 apêndices. O Apêndice A apresenta um resumo dos principais sistemas de arquivos distribuídos existentes, mostrando suas principais vantagens e desvantagens e fazendo uma análise sobre a

possibilidade de usá-los em plataformas maciçamente paralelas baseadas na Internet. No Apêndice B se encontra o diagrama de classes do sistema de arquivos implementado, mostrando todas as classes criadas e a relação existente entre elas. No Apêndice C tem-se uma descrição da interface oferecida pelo sistema de arquivos implementado. O Apêndice D apresenta as publicações derivadas deste trabalho.

Capítulo 2

Propostas de Sistemas de Arquivos

No capítulo anterior foram citados os principais sistemas de arquivos existentes, sendo que alguns constituem padrões de fato. No entanto, nenhum deles atende satisfatoriamente as exigências de um computador paralelo virtual baseado na Internet. Neste capítulo mostraremos as principais características que um sistema de arquivos necessita para este tipo de computador. Três abordagens são propostas e analisadas em detalhes para determinar a mais adequada para este tipo de plataforma. Uma especificação detalhada da proposta escolhida também pode ser encontrada no capítulo.

2.1 Características Gerais do Sistema de Arquivos

Um elemento vital para qualquer sistema de computação é o sistema de arquivos. Basicamente, o tipo de sistema de computação determinará que tipo de sistema de arquivos será necessário. Em particular, para o desenvolvimento de um MPVC é necessário ter-se um sistema de arquivos capaz de manipular de forma eficiente o volume de informação geralmente manipulado por problemas maciçamente paralelos. Além disso, as aplicações paralelas que serão executadas nestes MPVCs se caracterizam por serem compostas de um grande número de tarefas que podem precisar ter acesso a um mesmo arquivo simultaneamente. Por isso, para que um sistema de arquivos seja apropriado para o processamento maciçamente paralelo sobre a Internet, ele tem que satisfazer vários requisitos. A seguir são analisados os mais importantes.

Portabilidade: permitir o seu uso transparente em arquiteturas heterogêneas, incluindo os computadores pessoais que não tenham uma conexão permanente à Internet.

Disponibilidade e tolerância a falhas: os dados devem ficar disponíveis o maior tempo possível para as aplicações, independentemente de falhas ocorridas em alguns dos seus componentes.

Consistência: ter uma alta consistência nos dados, garantindo a coerência de toda a informação espalhada pelo sistema.

Escalabilidade: permitir o acesso simultâneo de um grande número de tarefas a um mesmo arquivo ou conjunto de arquivos sem que o sistema fique sobrecarregado.

Reconfiguração automática: se reconfigurar automaticamente quando um computador entrar ou sair do sistema.

Eficiência: diminuir o tempo das operações de entrada e saída minimizando, sempre que possível, as comunicações através da rede.

Espaço de nomes: ter um espaço de nomes global e uniforme para facilitar a denominação dos arquivos.

Segurança: garantir a segurança dos dados gerados no processamento e dos dados armazenados nos computadores que participam do MPVC.

Instalação: não requerer instalação e manutenção de softwares específicos por parte dos usuários dos computadores participantes.

2.2 Propostas Preliminares de Sistemas de Arquivos para MPVC

A seguir serão apresentadas em detalhes as principais abordagens consideradas neste trabalho. A eleição da melhor proposta foi baseada nas vantagens e desvantagens de cada uma delas, e de como foram satisfeitos os requisitos antes mencionados. O problema principal a se enfrentar no momento de projetar um sistema de arquivos para um computador maciçamente paralelo virtual baseado na Internet é o fato de que tanto os dados a serem processados como os resultados obtidos no processamento têm que ser armazenados em um lugar acessível para todas as tarefas das aplicações executadas no computador virtual. Voltando à plataforma de referência utilizada (ver Figura 1.1), um dos pontos acessíveis a partir de qualquer lugar do MPVC é o Servidor Web. No entanto, alguns dos computadores do computador virtual poderiam oferecer os seus recursos para o sistema de arquivos e assim melhorar a distribuição de carga e a tolerância a falhas. Até mesmo os computadores que estejam usando *applets* Java como forma de interação com o resto do MPVC podem oferecer seus recursos ao sistema de arquivos, fazendo uso de *applets* assinados, que permite a um *applet* descarregado em um computador usar melhor os recursos por ele oferecidos (em particular, podem manipular livremente os arquivos no computador descarregado).

2.2.1 Proposta 1: SAC - Sistema de Arquivos Centralizado

A proposta mais simples para o sistema de arquivos desejado seria uma abordagem centralizada. Nesta abordagem, um Servidor de Arquivos Central armazenaria todos os arquivos usados no MPVC. Este servidor poderia estar localizado no próprio Servidor Web da plataforma de referência e assim toda a informação ficaria centralizada em um único lugar do sistema, como mostra a Figura 2.1. Todas as tarefas executadas nos computadores que

formam o MPVC interagiriam diretamente com este Servidor de Arquivos Central, fazendo uso dos arquivos que necessitassem.

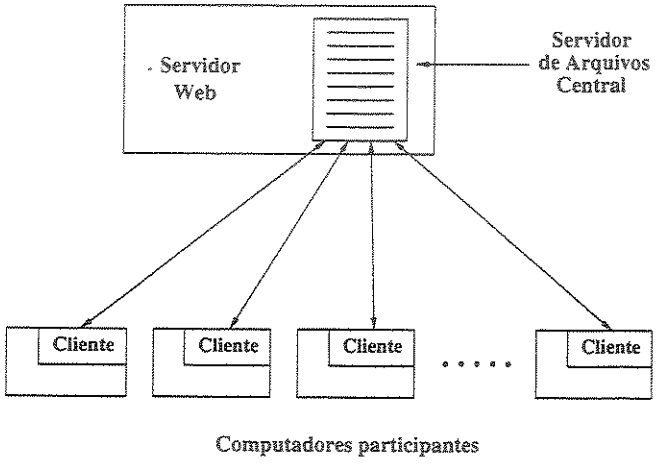

Figura 2.1: Proposta 1 - Sistema de Arquivos Centralizado

A principal vantagem desta abordagem centralizada é a sua simplicidade. Geralmente os sistemas centralizados são notavelmente mais simples e eficientes que os sistemas distribuídos, já que não precisam dos mecanismos, às vezes complexos, de comunicação entre os diferentes componentes que estes últimos precisam. Além disso, esta proposta mantém uma consistência estrita dos dados nos arquivos, já que não se necessita de replicação de informação. Qualquer mudança feita em um arquivo por uma tarefa é imediatamente percebida pelas tarefas restantes que o estão utilizando.

No entanto, pelo fato da informação estar disponível somente no Servidor de Arquivos Central, o mesmo torna-se um ponto vulnerável para todo o sistema, e a disponibilidade da informação fica comprometida. Como resultado disto, esta solução não será tolerante a falhas, inviabilizando a implementação de mecanismos de reconfiguração automática. Se o Servidor de Arquivos Central ficar fora do ar, todas as tarefas que estiverem usando os serviços do sistema de arquivos ficarão impossibilitadas de continuar seu trabalho.

Além disso, considerando que estamos tratando com sistemas maciçamente paralelos, o número de pedidos ao servidor pode ser grande, aumentando com o número de tarefas no sistema. Desta forma, devido à pouca escalabilidade do Servidor de Arquivos Central, ele pode se converter em um gargalo para o sistema.

Embora esta abordagem não seja a mais adequada para ser usada como sistema de arquivos para um MPVC, ela é útil como ponto de referência para avaliar a eficiência das outras propostas.

Uma forma de aumentar a disponibilidade do sistema nesta abordagem é oferecer replicação de servidores para diminuir a carga de trabalho, mas o sistema ficaria mais complexo e a manutenção da consistência estrita dos dados já não seria trivial, eliminando assim as principais vantagens da abordagem centralizada.

2.2.2 Proposta 2: SAD_GC - Sistema de Arquivos Distribuído baseado em Grupos de Computadores

Para minimizar o trabalho do Servidor de Arquivos Central na proposta anterior e impedir que ele represente um gargalo no sistema, pode-se usar o esquema mostrado na Figura 2.2.

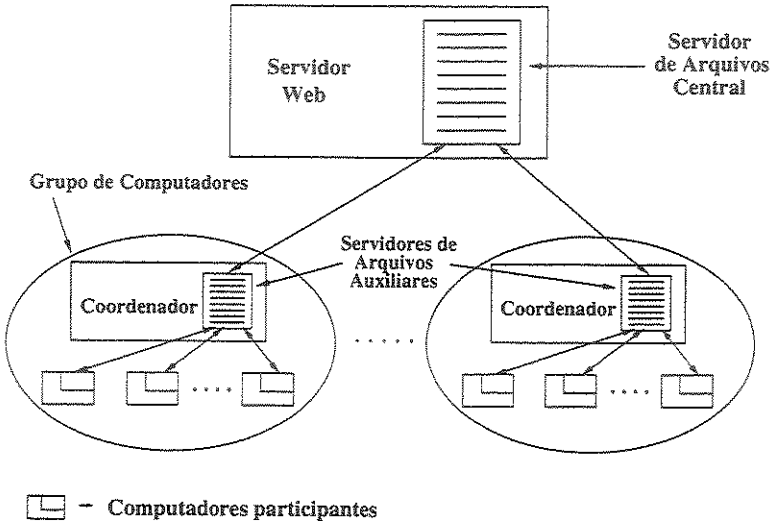

Figura 2.2: Proposta 2 - Sistema de Arquivos Distribuído baseado em Grupos de Computadores

Nesta proposta tem-se um Servidor de Arquivos Central contendo todos os arquivos usados no MPVC, que pode estar localizado no próprio Servidor Web da plataforma de referência utilizada. Os computadores do MPVC são organizados em grupos, com um coordenador para cada grupo criado. Para cada grupo tem-se um Servidor de Arquivos Auxiliar, onde poderá haver uma cópia de todos os arquivos utilizados por qualquer uma das tarefas nos computadores desse grupo. Estas cópias dos arquivos usados pelo grupo podem ser feitas sob demanda, isto é, somente quando uma das tarefas sendo executada nos computadores do grupo precisa de um determinado arquivo é que o mesmo é copiado do Servidor de Arquivos Central para o servidor do grupo. Desta forma, diminui-se a quantidade de acessos destas tarefas ao Servidor de Arquivos Central. Os servidores de arquivos de cada grupo seriam replicados, para obter um maior nível de tolerância a falhas e melhor desempenho.

Diferentemente da proposta centralizada, esta abordagem é escalável, já que podem ser usados vários Servidores de Arquivos Auxiliares por grupo e assim distribuir a carga de trabalho, aumentando a eficiência do sistema de arquivos. Além disso, a disponibilidade da informação é maior, já que as tarefas das aplicações executas no MPVC podem ter acesso aos arquivos em vários pontos do sistema, aumentando a tolerância a falhas e possibilitando a implementação de mecanismos de reconfiguração automática quando algum (ou alguns) dos servidores não estiver disponível.

O principal problema desta proposta está em manter a consistência das informações espalhadas pelos diferentes grupos que formam o MPVC. Uma determinada aplicação for-

mada por milhares de tarefas pode estar espalhada por vários grupos, e assim todos os arquivos utilizados por estas tarefas deverão estar replicados em todos os grupos utilizados. Se o número destes grupos for grande, o esforço necessário para manter a consistência das réplicas dos arquivos usados seria muito alto.

2.2.3 Proposta 3: SAD_GSA - Sistema de Arquivos Distribuído baseado em Grupos de Servidores para Aplicações

A última proposta, com uma estrutura mais geral e mais independente do MPVC, visa limitar o número de servidores para cada arquivo, principal desvantagem da proposta anterior. Esta proposta baseia-se no fato que aplicações diferentes, salvo raras exceções, não compartilham arquivos em sua execução. Assim, os arquivos usados no MPVC podem ser divididos em conjuntos disjuntos, cada um dos quais armazena os arquivos necessários para uma aplicação. Também nesta proposta têm-se um Servidor de Arquivos Central contendo todos os arquivos usados no MPVC, além de Grupos de Servidores para Aplicações que contêm os arquivos de uma ou mais aplicações. Os computadores participantes do MPVC interagem com os servidores apropriados de acordo com as tarefas que estão executando. Além destes servidores, nesta abordagem é necessária a existência de um Resolvedor de Endereços que determine a partir do nome do arquivo, o servidor onde ele está localizado. Quando uma tarefa necessita acessar um arquivo, pergunta ao Resolvedor de Endereços, que lhe indica o endereço do servidor apropriado dentro de um dos Grupo de Servidores para Aplicações, possibilitando assim a interação entre eles. Este Resolvedor de Endereço deve ser replicado para evitar que falhas nesse componente comprometam a tolerância a falhas do sistema como um todo. A Figura 2.3 mostra maiores detalhes desta proposta.

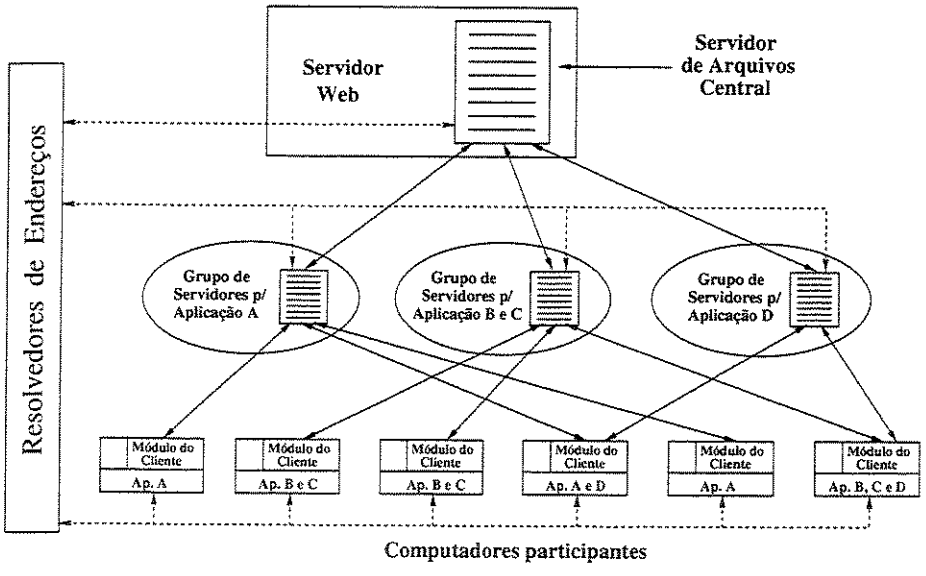

Figura 2.3: Proposta 3 - Sistema de Arquivos Distribuído baseado em Grupos de Servidores para Aplicações

A idéia é ter mais de um servidor por cada arquivo usado no sistema. O número de servidores será limitado (por exemplo, três servidores por arquivo), para facilitar a manutenção da consistência da informação dentro de um grupo. Este número poderá ser adaptado às necessidades e à quantidade de computadores do MPVC. Portanto, esta abordagem é mais escalável devido a esta capacidade de adaptação. Além disso, é viável a implementação de mecanismos de reconfiguração automática dentro de cada grupo de servidores, aumentando a tolerância a falhas dos mesmos e do sistema como um todo.

O maior problema desta abordagem é a sua complexidade, além de ser necessário um número maior de servidores. Esta proposta, em principio, também não prevê aplicações que compartilhem arquivos durante a execução, precisando ser estendida, caso necessário, para permitir este compartilhamento.

2.3 Comparação das Propostas

A Tabela 2.1 mostra uma comparação entre as três propostas para escolher a mais adequada para um computador maciçamente paralelo virtual baseado na Internet segundo a referência básica utilizada. Esta comparação está baseada em alguns dos requisitos apresentados na seção 2.1.

	SAC	SAD_GC	SAD_GSA
Disponibilidade	baixa	alta	alta
Tolerância a falhas	baixa	alta	alta
Escalabilidade	baixa	alta	alta
Reconfiguração automática	não	sim	sim
Manutenção de consistência	N.A.	difícil	média

Tabela 2.1: Comparação das propostas preliminares de Sistemas de Arquivos

De acordo com os dados da comparação, foi escolhida a terceira proposta como a de maior potencial para o sistema de arquivos desejado. Esta proposta oferece uma alta disponibilidade, possibilitando a implementação de mecanismos de reconfiguração automática do sistema ante eventuais mudanças de carga ou possíveis problemas dos servidores, o que aumenta a tolerância a falhas do sistema. Além disso, estes mecanismos podem ser implementados tomando como base os Grupos de Servidores para Aplicações, tornando os mesmos mais eficientes. A abordagem também é escalável, devido à possibilidade destes grupos de se adaptarem às necessidades das aplicações e do MPVC.

Esta proposta torna mais viável a principal desvantagem da segunda proposta: a manutenção da consistência entre as diferentes réplicas dos arquivos espalhadas no sistema. O SAD_GSA limita o número de servidores para cada arquivo, facilitando assim a resolução do problema da inconsistência.

Um dos problemas desta proposta é o fato dela não prover, em princípio, suporte a aplicações que compartilhem arquivos durante a execução. No entanto, como veremos mais na frente neste capítulo, ela pode ser estendida para prever esses casos.

O restante do capítulo apresenta uma especificação mais detalhada da proposta escolhida. Esta especificação inclui um detalhamento da arquitetura desta abordagem e de cada um de seus componentes, mostrando suas características, funções, relações entre eles, dentre outras questões. A especificação também inclui os serviços oferecidos pelo sistema de arquivos, a estrutura destes arquivos, como eles são usados pelos usuários do sistema, as características de todos os servidores usados nesta proposta, a interface do sistema, os principais mecanismos implementados, dentre outros aspectos fundamentais.

2.4 Componentes Principais

A Figura 2.4 mostra a arquitetura do SAD_GSA e as possíveis interações entre seus principais componentes.

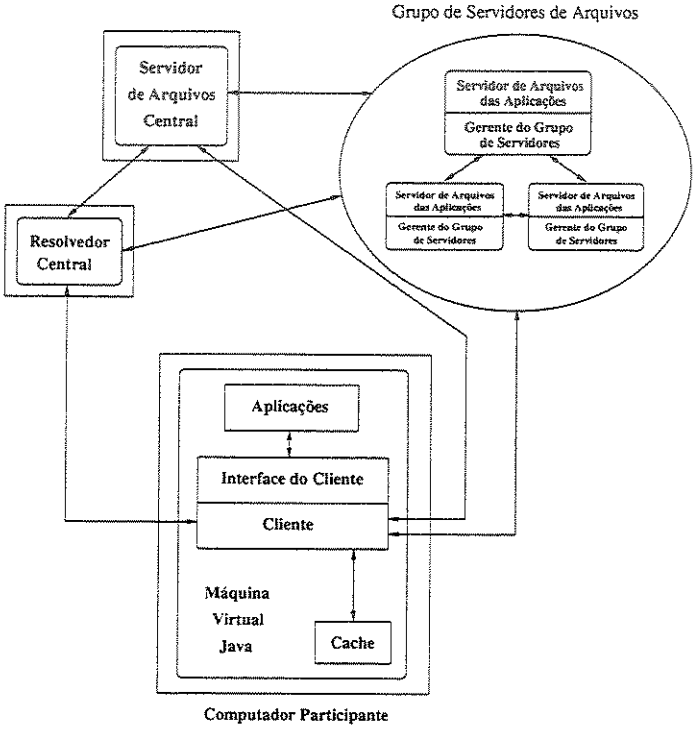

Figura 2.4: Arquitetura do SAD_GSA

Cada componente mostrado na figura tem uma função bem definida dentro do sistema de arquivos distribuído. A seguir são mostradas as funções dos componentes mais importantes.

2.4.1 Servidor de Arquivos Central

Devido à estrutura do MPVC, todos os dados devem ser armazenados em um lugar acessível por todas as tarefas das aplicações paralelas distribuídas nos computadores participantes, garantindo desta forma a disponibilidade de pelo menos uma cópia atualizada de todos os

arquivos. Por isto, o primeiro componente do sistema de arquivos é o Servidor de Arquivos Central, que armazena uma cópia de todos os arquivos usados no computador virtual. Dado o caráter dinâmico do MPVC, onde qualquer computador pode entrar ou sair em qualquer momento, este é um dos componentes mais importantes do sistema de arquivos. No entanto, ele é raramente utilizado para atender diretamente os pedidos das aplicações. Este Servidor de Arquivos Central somente é usado caso os servidores que atendem diretamente os pedidos das aplicações (réplicas do Servidor de Arquivos Central, descritos na seção 2.4.3) falhem, para evitar que o mesmo seja um gargalo para o sistema. Ele deve estar localizado em um dos pontos mais visíveis, estáveis e confiáveis do sistema, além de ter um bom desempenho. Este servidor pode ser replicado para aumentar a disponibilidade dos dados.

2.4.2 Resolvedor de Endereços

O Resolvedor de Endereços também deve se localizar em um dos pontos mais visíveis, estáveis e confiáveis do sistema, devido a sua importância. Ele é encarregado de manter os endereços de todos os servidores disponíveis, além da informação sobre suas características (processador, quantidade de memória, velocidade do enlace, etc.) e sua carga de trabalho. Assim que um servidor de arquivos do sistema, incluindo o Servidor de Arquivos Central, é executado, ele registra todas estas características no resolvedor.

Desta forma, qualquer mudança ocorrida nos servidores, seja por que um servidor saiu do ar, um novo servidor foi incorporado ao sistema ou um servidor mudou seu estado, causa uma atualização no Resolvedor de Endereços. A forma como estas atualizações são feitas encontra-se melhor detalhada na próxima seção. Sendo assim, tem-se pelo menos uma cópia atualizada com a disponibilidade e a carga de trabalho de todos os servidores do sistema. Este resolvedor é o primeiro ponto de contato com o sistema de arquivos e sua função principal é fornecer todas as informações necessárias sobre os servidores de arquivos disponíveis aos novos computadores que entram no MPVC. Com estas informações, estes computadores, através do software cliente (que chamaremos de Módulo do Cliente, explicado mais detalhadamente na seção 2.4.4), decidem quais servidores atenderão os pedidos das aplicações que estão sendo executadas neles. Uma vez que esses computadores obtiverem os dados dos servidores, não precisarão mais se comunicar com o resolvedor. Todas as futuras mudanças ou atualizações destes servidores serão obtidas nos próprios servidores, evitando sobrecarregar o Resolvedor de Endereços.

Além de manter o endereço e as características de todos os servidores disponíveis no sistema, o Resolvedor de Endereços implementa um mecanismo para determinar se algum desses servidores ficou fora do ar. Os Grupos de Servidores para Aplicações também implementam um mecanismo semelhante. Com ajuda deste mecanismo o sistema pode se reconfigurar automaticamente ante possíveis falhas nos servidores. Na seção 2.9 tem-se maiores detalhes de como esta reconfiguração é feita.

2.4.3 Grupos de Servidores para Aplicações

Para aumentar a escalabilidade, a disponibilidade e a tolerância a falhas do serviço oferecido pelo SAD_GSA, a informação armazenada no Servidor de Arquivos Central é replicada.

O sistema de arquivos cria grupos de servidores, chamados de Grupos de Servidores para Aplicações. Estes grupos estão formados pelos Servidores de Arquivos para Aplicações, encarregados do armazenamento destas réplicas dos dados. Para garantir a eficiência e o bom desempenho do sistema, os Servidores de Arquivos para Aplicações devem ser computadores estáveis, confiáveis e de alto desempenho. Todos os pedidos das tarefas das aplicações executadas no MPVC serão resolvidos diretamente por estes servidores, isto é, as aplicações interagem diretamente com eles. Sendo assim, todos os arquivos usados por uma aplicação paralela são armazenados em um único Grupo de Servidores para Aplicações e todas as requisições de entrada/saída dessa aplicação são enviadas aos servidores desse grupo. Além disso, todos os arquivos usados por essa aplicação são replicados em cada um dos servidores do grupo. Desta forma, qualquer um deles pode atender qualquer pedido das aplicações, aumentando a disponibilidade da informação e a tolerância a falhas do sistema. Para tornar isto possível, é preciso implementar um mecanismo para a manutenção da coerência entre todas as cópias dos dados espalhadas no grupo, evitando possíveis inconsistências. Uma outra alternativa seria dividir todos os arquivos usados por uma aplicação entre os diferentes servidores do grupo, já que o volume de informação pode ser grande. No entanto, se cair um dos servidores do grupo, todos os arquivos nele armazenados não poderiam ser mais acessados pelas tarefas, ficando comprometidas tanto a escalabilidade como a tolerância a falhas do sistema. Na seção 2.7.3 tem-se uma especificação completa do mecanismo de replicação do sistema de arquivos proposto, que trata da manutenção da coerência entre as diferentes réplicas e do protocolo de atualização, dentre outras questões.

Têm-se varias formas diferentes de associar uma aplicação a um Grupo de Servidores para Aplicações, e dentre outros aspectos, esta associação depende das características e de como o sistema de arquivos é implementado. Por exemplo, pode ser criado um Grupo de Servidores para Aplicações para cada aplicação executada no computador virtual, ou um mesmo grupo atender os pedidos de várias aplicações. No SAD_GSA foi escolhida uma abordagem baseada nos seus usuários. Para cada usuário que estiver interagindo com o sistema de arquivos será criado um Grupo de Servidores para Aplicações. Todas as aplicações executadas por esse usuário vão interagir diretamente com os servidores do grupo criado.

Outra característica dos Servidores de Arquivos para Aplicações é que eles adotam a semântica de sempre responderem a todas as requisições feitas pelas tarefas das aplicações, seja com a resposta ao pedido, com o envio do mesmo a outro servidor menos carregado que o responderá ou com uma indicação de que não pode atendê-lo, caso ele e todos os demais servidores do grupo estejam carregados. Desta forma o sistema garante que as tarefas recebam sempre uma resposta em um período de tempo razoavelmente pequeno. A seção 2.8.1 mostra mais detalhes desta características dos servidores e como ela é usada para implementar um mecanismo de balanceamento de carga no sistema.

Supervisor do Grupo de Servidores para Aplicações

A informação sobre a disponibilidade e carga de trabalho dos servidores de arquivos disponíveis no sistema não é mantida somente no Resolvedor de Endereços. Cada Servidor de Arquivos para Aplicações cria um subcomponente, chamado de supervisor do Grupo de

Servidores para Aplicações, que replica as informações de todos os servidores que pertencem ao seu grupo. Assim, esta informação não fica centralizada em um componente só, aumentando a disponibilidade da mesma.

Estes supervisores dos grupos têm duas funções principais. Em primeiro lugar, eles são os encarregados de gerenciar a carga de trabalho de cada Servidor de Arquivos para Aplicações. Assim, se o servidor mudar o seu estado, este supervisor detecta o fato e o informa aos restantes supervisores do grupo, incluindo o Resolvedor de Endereços. Com toda esta informação disponível e atualizada em cada servidor de um grupo torna-se viável, sob determinadas condições (por exemplo, aumento da carga de trabalho de um servidor do grupo), devolver aos Módulos dos Clientes não só os dados dos pedidos realizados, mas também o estado de todos os servidores do grupo que estão atendendo uma determinada aplicação. Desta forma, os Módulos dos Clientes ficarão sabendo de mudanças na disponibilidade e na carga dos servidores sem necessidade de se comunicar com o Resolvedor de Endereços, diminuindo o número de mensagens enviados pela rede (estes dados podem viajar de carona nas respostas aos pedidos) e evitando que o Resolvedor de Endereços se torne um ponto de gargalo. Portanto, estes supervisores participam do mecanismo de balanceamento de carga dos servidores disponíveis no sistema. Este mecanismo será explicado com maiores detalhes na seção 2.8.1.

A outra tarefa é gerenciar o Grupo de Servidores para Aplicações com o qual está relacionado. Eles implementam um mecanismo para determinar se algum dos servidores do grupo ficou fora do ar. Caso isto aconteça, as mudanças são espalhadas pelo grupo e pelo Resolvedor de Endereços. Assim, o grupo fica atualizado, evitando fornecer aos Módulos dos Clientes informações inconsistentes. Este mecanismo é totalmente distribuído, isto é, não se tem um componente centralizado do supervisor do Grupo de Servidores para Aplicações e sim várias cópias espalhadas pelos diferentes grupos do sistema (seção 2.9).

2.4.4 Módulo do Cliente

O Módulo do Cliente é o componente do sistema descarregado nos computadores participantes no MPVC. Este componente permite a interação entre as aplicações e os servidores de arquivos. Ou seja, ele é o encarregado de, dado um nome de arquivo, encontrar a localização física do mesmo e enviar o pedido da aplicação ao servidor escolhido.

Para isto, cada Módulo do Cliente mantém os endereços e as características dos servidores de arquivos disponíveis. Estas informações são obtidas inicialmente no Resolvedor de Endereços e atualizadas freqüentemente com as informações provenientes dos Servidores de Arquivos para Aplicações. Sendo assim, em cada Módulo do Cliente, para cada aplicação sendo executada no computador, é escolhido um Servidor *Default* dentro do Grupo de Servidores para Aplicações associado à aplicação. Este servidor atenderá todas as requisições dessa aplicação. O Servidor *Default* é escolhido segundo as características e carga de trabalho dos servidores do grupo, podendo mudar de acordo com futuras mudanças no grupo. Com esta abordagem, parte da funcionalidade do sistema de arquivo passa dos servidores para os Módulos dos Clientes, aumentando a eficiência do sistema e diminuindo a comunicação através da rede. Na seção 2.8.1 tem-se uma especificação mais formal do mecanismo de seleção destes servidores.

Sempre que os computadores participantes do computador virtual possuam espaço de disco disponível e permitirem acessos de escrita nele, o conceito de cache é utilizado de forma intensiva no Módulo do Cliente. Ele replicará os arquivos utilizados pelas aplicações executadas no seu computador, minimizando o tráfego na rede e aumentando a eficiência do sistema. A seção 2.10.3 mostra como o cache é utilizado no sistema, como os arquivos são replicados no Módulo do Cliente, dentre outras características.

Interface do Sistema

A Interface do Sistema é a interface do serviço oferecido pelo sistema de arquivos para as aplicações que serão executadas no computador virtual. Nas seções a seguir são explicados em detalhes a forma como são definidos os arquivos, os serviços oferecidos pelo sistema de arquivos, como os arquivos são usados pelas aplicações, dentre outras questões de interesse.

A especificação completa da interface do SAD_GSA encontra-se no Apêndice C.

2.5 Serviços do Sistema de Arquivos

Um sistema de arquivos deve oferecer dois serviços fundamentais: o serviço de arquivos e o serviço de diretório. O primeiro destes serviços está relacionado com as operações realizadas sobre arquivos independentes, como por exemplo, ler e escrever dados. O segundo serviço diz respeito às operações de criar, apagar e gerenciar diretórios, criar e apagar arquivos, manipular as permissões de acesso destes diretórios e arquivos, dentre outras operações. A seguir serão discutidas as características destes serviços no sistema de arquivos proposto.

2.5.1 Serviço de Arquivos

O serviço de arquivos oferece operações sobre arquivos independentes, como ler ou escrever dados. Mas antes é preciso definir o que é um arquivo no SAD_GSA.

Representação dos Arquivos

De forma geral tem-se duas formas de definir um arquivo.

Como uma seqüência de bytes: o significado e a estrutura dos dados no arquivo é completamente definida pelas aplicações e o sistema de arquivos não tem que se preocupar com estes detalhes. Ele simplesmente considera o arquivo como uma seqüência de bytes.

Como uma seqüência de blocos: a informação é organizada em blocos e identificadas por uma chave. O sistema de arquivos lê os dados do arquivo em blocos, usando para isto as chaves que os identificam. Ou seja, o bloco definido é a menor unidade de acesso ao arquivo.

O sistema de arquivos proposto definirá os seus arquivos como uma seqüência de bytes, para desta forma obter uma maior flexibilidade no uso dos mesmos. A maioria dos sistemas distribuídos atuais tratam os seus arquivos desta forma.

Atributos dos Arquivos

Outro aspecto fortemente relacionado com os arquivos são os seus atributos. Um conjunto de atributos foi definido e associado a cada arquivo. A relação destes atributos e suas funções são mostradas a seguir.

name: indica o nome do arquivo. Identifica de forma única um arquivo no sistema.

type: indica se é um arquivo ou um diretório.

owner: indica o usuário que é dono do arquivo.

group: indica o grupo a que pertence o usuário que é dono do arquivo. É criado pelo sistema um grupo especial: ADMIN. Os usuários que pertencem a este grupo têm total acesso ao sistema de arquivos.

size: indica o tamanho do arquivo, em bytes.

creationDate: indica quando o arquivo foi criado.

lastAccessDate: indica quando foi o último acesso ao arquivo.

lastModificationDate: indica quando foi a última modificação do arquivo.

access: indica as permissões de acesso ao arquivo. Estas são de dois tipos: leitura e escrita. Estas permissões estão divididas em três partes: as permissões de acesso do usuário que criou o arquivo, as permissões de acesso dos usuários que pertencem ao mesmo grupo do proprietário do arquivo e as permissões de acesso dos restantes usuários do sistema (esquema similar ao empregado no Sistema de Arquivos de Unix).

status: indica a forma como o arquivo está sendo usado. Os possíveis estados são fechado, que indica que o arquivo não esta sendo usado por aplicação nenhuma; aberto para leitura, que indica que o arquivo esta aberto somente para leitura, e aberto para escrita, que indica que o arquivo esta aberto somente para escrita.

Estes atributos são armazenados em um arquivo diferente, ou seja, em um arquivo serão armazenados os dados e em outro arquivo serão armazenados os atributos. Desta forma, embora sejam gerados um maior número de arquivos, a manipulação separada de dados e atributos é uma abordagem mais simples para o sistema. Além disso, a possibilidade de importar de e exportar para o sistema de arquivos proposto é uma tarefa trivial que não implica em modificações dos arquivos de dados.

Forma de Utilização dos Arquivos

Atualmente está muito difundido o uso dos *streams* para ler e escrever informação em qualquer dispositivo de armazenamento. Isto faz com que os programadores estejam se acostumando a esta nova forma, aproveitando suas facilidades do ponto de vista de programação.

Um dos objetivos do sistema de arquivos que está se implementando é a sua facilidade de uso por parte das aplicações. Por isto, foram usados os *streams* como forma de entrada e saída da informação no sistema de arquivos proposto. A seguir são mostradas as formas diferentes como as aplicações podem usar os arquivos.

Como um *stream* de entrada: as aplicações lêem bytes do arquivo de forma seqüencial.

Como um *stream* de saída: as aplicações escrevem bytes no arquivo de forma seqüencial.

Com esta diferenciação dos *streams* de entrada e de saída o sistema pode dar um tratamento diferenciado a cada tipo de arquivo, aumentando assim a sua eficiência. Na seção 2.10.3 é mostrada a forma como o sistema manipula os diferentes tipos de arquivos.

2.5.2 Serviço de Diretórios

O serviço de diretório permite criar e apagar diretórios, criar e apagar arquivos, listar o conteúdo de um diretório e controlar o acesso das aplicações tanto aos diretórios como aos arquivos contidos neles, dentre outras operações. Portanto, é função do serviço de diretório manter toda a informação sobre a hierarquia de diretórios do sistema. Diretórios são tratados pelo sistema da mesma forma como são tratados os arquivos comuns. Eles têm os mesmos atributos que os arquivos (com exceção do estado) e seu conteúdo provê informações sobre seus arquivos e subdiretórios (nome e tipo de cada arquivo ou diretório). Esta abordagem torna mais simples a implementação do sistema de arquivos.

Representação dos Nomes

Além das operações acima, neste serviço é definida a composição dos nomes dados aos arquivos e diretórios. A Figura 2.5 mostra o espaço de nomes oferecido pelo sistema de arquivos proposto.

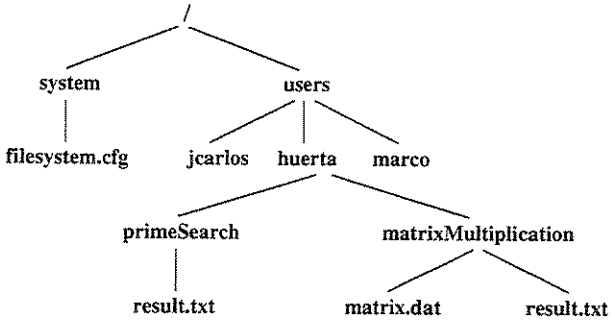

Figura 2.5: Espaço de nomes do SAD_GSA

O sistema possui inicialmente dois diretórios, *system* e *users*. No diretório *system* são armazenadas todas as informações utilizadas diretamente pelo sistema de arquivos, tais

como arquivos de configuração, lista de usuários, dentre outras. No diretório *users* são armazenadas as informações dos diferentes usuários. Para cada usuário do sistema de arquivos é criado um diretório, chamado de Diretório do Usuário (*home directory*). Nesse diretório o usuário armazena todas as informações usadas pelas suas aplicações. Para cada aplicação que o usuário implemente, ele deve criar um novo diretório com o mesmo nome da aplicação. Nesse diretório são armazenados todos os arquivos usados por essa aplicação específica. Por exemplo, se o usuário **huerta** implementa uma aplicação chamada de *primeSearch*, ele deve criar um diretório com esse nome em seu Diretório de Usuário.

Deve ser ressaltar que o esquema mostrado acima não é obrigatório e os usuários do sistema de arquivos podem escolher livremente como esta estrutura de diretórios será criada. No entanto, com o uso desta convenção é possível definir uma organização mais eficiente do sistema de arquivos proposto, o que facilita o processo de replicação de informação nos servidores de arquivos, como será mostrado em detalhes na seção 2.7.3.

De forma geral, as aplicações usarão as seguintes notações para denominar os arquivos.

Arquivos de sistema:
/system/nome do arquivo

Arquivos comuns (arquivos criados pelos usuários do sistema):
/users/nome do usuário/nome da aplicação/nome do arquivo

Esta notação é chamada de nome absoluto do arquivo e usa o caminho completo até o arquivo desejado, a partir do diretório raiz. Por exemplo, se uma aplicação do usuário **huerta** quiser usar os arquivos *filesystem.cfg* e *result.txt*, mostrados na Figura 2.5, deverá usar as notações */system/filesystem.cfg* e */users/huerta/searchPrime/result.txt*, respectivamente, que representam os nomes absolutos dos arquivos. Estes exemplos mostram como é possível para uma aplicação usar da mesma forma arquivos do sistema e arquivos comuns.

Para facilitar o uso dos arquivos comuns pelas aplicações, também podem ser usados seus nomes relativos. Voltando ao exemplo da aplicação do usuário **huerta** que usa o arquivo *result.txt*, nota-se que ela pode usar os nomes mostrados a seguir.

1. **/users/huerta/searchPrime/test.txt:** nome absoluto do arquivo.
2. **~huerta/searchPrime/test.txt:** nome relativo ao usuário **huerta**.
3. **searchPrime/test.txt:** nome relativo ao diretório atual do usuário que está executando a aplicação. Inicialmente, o diretório atual do usuário é o seu Diretório do Usuário (*home directory*), mas o sistema oferece funções para que as aplicações possam modificá-lo.

Na seção 2.2.3 foi mostrado que uma desvantagem do SAD_GSA era não suportar aplicações que compartilham arquivos durante a execução. No entanto, com o uso desta notação a desvantagem fica atenuada. Uma outra aplicação do usuário **huerta** ou mesmo de um outro usuário qualquer do sistema pode ter acesso ao arquivo *result.txt* usando as duas primeiras notações definidas, bastando para isso que ele tenha as permissões de acesso

necessárias para completar a operação. De forma geral, uma aplicação usará a notação 2 para se referir a arquivos de outras aplicações (do mesmo usuário ou de usuários diferentes), e a notação 3 para se referir aos seus próprios arquivos. Além disso, como será explicado na seção 3.5.2, o sistema oferece comandos para importar e exportar arquivos entre sistemas de arquivos diferentes, copiar arquivos entre aplicações, dentre outros.

Características do Espaço de Nomes Definido

A notação definida pelo sistema de arquivos proposto tem as principais características desejadas na hora de definir o espaço de nomes em um sistema de arquivos, as quais são citadas a seguir.

Global e uniforme: Qualquer arquivo pode ser usado por qualquer aplicação e, para cada arquivo, é usada sempre a mesma notação (notação absoluta ou relativa).

Transparência de localização: O nome do arquivo não oferece informação alguma da localização física do arquivo, isto é, da localização do computador onde ele está armazenado.

Independência de localização: O nome do arquivo não muda se o arquivo muda de localização, ou seja, se muda o computador onde ele está armazenado.

Assim, é total responsabilidade do sistema de arquivos garantir que as aplicações não tenham que se preocupar com estes aspectos, aumentando a flexibilidade na denominação dos arquivos e na sua manipulação.

2.6 Características dos Servidores

Uma das características que definem um servidor é a forma como ele manipula os seus arquivos. Tem-se dois tipos de servidor, o servidor *stateful* e o servidor *stateless*.

Os servidores *stateful* são aqueles servidores que mantêm informações sobre as requisições dos clientes, isto é, uma vez que um arquivo é aberto, todas as informações referentes ao arquivo e ao cliente que está usando-o são mantidas em uma tabela no servidor. O cliente somente recebe um descritor que identifica o arquivo. Este descritor é usado pelo cliente em requisições futuras. Quando uma operação é solicitada ao servidor, ele procura o arquivo associado ao descritor na sua tabela e faz a operação. Este tipo de servidor tem um desempenho maior e as mensagens usadas são pequenas, minimizando o tráfego na rede. No entanto, ele tem dois problemas fundamentais. O primeiro problema é de escalabilidade, que surge quando aumenta o número de clientes. Isto aumenta o tamanho da tabela mantida no servidor, fazendo com que sua manipulação já não seja uma tarefa trivial, diminuindo a sua eficiência e limitando o número de arquivos abertos. O outro problema é que neste tipo de servidor a recuperação ante possíveis falhas, tanto no servidor como nos clientes, é difícil, sendo necessária a implementação de mecanismo complexos e demorados. Se o servidor cair, perde-se toda a informação mantida na sua tabela, fazendo com que futuros pedidos dos clientes não possam ser respondidos. Para evitar isto, todos estes dados

devem ser armazenados em disco e recuperados em caso de falha. Além disso, o servidor também tem que implementar um mecanismo para saber quando um cliente cai, para assim conseguir liberar da tabela todos os arquivos que o mesmo estava usando.

Os servidores *stateless* são aqueles que não mantêm informação nenhuma do cliente, isto é, cada pedido é totalmente independente dos demais e contém toda a informação necessária para resolvê-lo. Para cada pedido, o servidor abre o arquivo requisitado e realiza a operação, fechando-o assim que a mesma é completada. Uma vez que um pedido é resolvido, o servidor apaga todas as informações associadas ao mesmo, não ficando nenhum dado nem do pedido nem do cliente no servidor. Estes servidores, embora com menor desempenho que os servidores *stateful* (não utilizam informações de pedidos anteriores e as mensagens são maiores), não precisam de tabela nenhuma para armazenar informação, o que faz com que sejam mais escaláveis e mais tolerantes a falhas.

A Tabela 2.2 resume as vantagens e desvantagens destes tipos de servidores.

	Servidores *Stateful*	Servidores *stateless*
Vantagens	- Melhor desempenho - Mantêm informações dos pedidos anteriores - Mensagens menores	- Escalável - Tolerante a falhas - Não limita o número de arquivos abertos
Desvantagens	- Não é escalável - Não é tolerante a falhas	- Precisa abrir e fechar os arquivos em cada operação - Mensagens maiores

Tabela 2.2: Comparação dos tipos de servidores

Fica claro que os servidores tipo *stateful*, apesar de serem os mais eficientes, não são muito apropriados para um sistema de arquivos de uma plataforma para o processamento maciçamente paralelo na Internet. Eles têm duas características que impedem o uso dos mesmos. Primeiro, haverá um número muito grande de tarefas que precisarão ter acesso simultâneo aos arquivos. Isto implica em que a tabela necessária para manter o estado de todos estes clientes e os arquivos que eles usam vai ter um tamanho considerável e sua manipulação será um problema sério para o servidor. Além disso, o fato de usar a Internet, uma rede pouco confiável, faz com que qualquer cliente ou mesmo um servidor possa ficar fora do ar, gerando novos problemas na manutenção desta tabela. O servidor precisará da implementação de mecanismos complexos de recuperação de falhas. É por isso que todos os servidores usados no sistema de arquivos proposto são do tipo *stateless*, incluindo o Servidor de Arquivos Central.

Para melhorar o desempenho dos servidores, eles vão manter em memória uma tabela de arquivos abertos com os últimos arquivos que foram utilizados, mas sem guardar informações sobre os clientes, mantendo assim sua condição de *stateless*. O tamanho desta tabela é fixo, evitando assim que aumente seu tamanho com o aumento do número de requisições. Quando um pedido chegar ao servidor, ele procura o arquivo referenciado na tabela de arquivos abertos. Se ele é encontrado, podem ser utilizados os dados armazenados

nele, evitando ter que procurá-lo no disco. Se o arquivo não é encontrado, ele é aberto e inserido na tabela pelo servidor. Se a tabela estiver cheia, um dos arquivos armazenados nela é eliminado, isto é, ele é fechado. Para eliminar um arquivo da tabela é usada uma política LRU (*Least Recently Used*), ou seja, serão eliminados os arquivos menos recentemente usados, procurando fazer com que a taxa de acerto – *hit rate* – seja elevada. Para um cliente que esteja fazendo uso de um arquivo de forma freqüente, é muito provável que sempre seja encontrado o arquivo na tabela de arquivos abertos depois da sua primeira requisição. Maiores detalhes podem ser encontrados na seção 3.5.3.

Esta proposta mantém as vantagens dos servidores *stateless*. Ela continua sendo tolerante a falhas, já que se um servidor cair e for reiniciado, ele só precisará criar uma nova tabela e começar o trabalho desde o início. Por outro lado, se um cliente fica fora do ar, os arquivos que ele estiver usando serão fechados quando suas entradas na tabela forem utilizadas por outros arquivos usados por outros clientes. Além disso, o servidor consegue atender, teoricamente, qualquer quantidade de clientes, já que o tamanho da tabela é fixo e não cresce com o aumento da quantidade de arquivos que são abertos ou com a quantidade de clientes, diferentemente do que acontece nos servidores *stateful*.

2.7 Replicação de Servidores

A Internet é uma rede complexa e seus enlaces podem não estar disponíveis em todo momento. Por estas características, qualquer sistema que deseje usar a Internet como rede de interconexão para um computador virtual deve garantir um alto nível de disponibilidade dos componentes que o formam, fornecendo mecanismos de tolerância e recuperação ante possíveis falhas que ocorram nos computadores do MPVC ou nas suas conexões à rede.

A principal técnica para se aumentar a disponibilidade dos componentes de um sistema é a replicação, definida em [Tan95] como "a manutenção de múltiplas cópias de informação, cada uma em um ponto diferente do sistema". Os motivos principais da replicação de informação são melhorar o desempenho do sistema, distribuindo a carga de trabalho entre os componentes replicados, aumentando assim a disponibilidade e a tolerância a falhas do serviço oferecido por estes componentes replicados. Isto aumenta a possibilidade de, caso um ou mais desses componentes caiam ou percam sua conexão com o sistema, ter-se pontos alternativos onde obter acesso aos dados. Um outro aspecto relacionado à replicação é a transparência. Uma replicação é transparente se os clientes dos serviços não têm que se preocupar com a localização física da informação, isto é, em qual das réplicas está a informação que eles precisam. Todos estes detalhes ficam por conta do sistema.

Assim como a maioria dos sistemas atuais que tratam o problema da tolerância a falhas usam a replicação dos componentes principais do sistema para obter a disponibilidade desejada, o sistema de arquivos proposto também oferece mecanismos de replicação de servidores, para melhorar a qualidade do serviço oferecido.

2.7.1 Modelos de Replicação

Existem vários modelos de replicação. No entanto, eles podem ser generalizados em três modelos principais [Tan95], os quais são mostrados a seguir.

Replicação explícita: qualquer operação de escrita em um arquivo é feita simultaneamente em todos os servidores (Figura 2.6). As operações de leitura podem ser feitas em qualquer um deles.

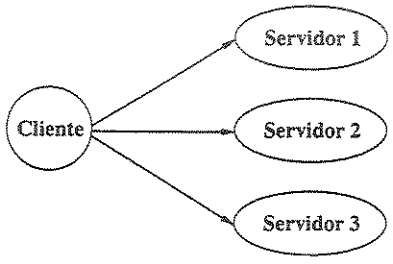

Figura 2.6: Replicação explícita

Nesta abordagem, como a nome indica, os clientes precisam explicitamente enviar as operações de escrita a todos os servidores. A sua principal vantagem é o fato de que todas as cópias sempre estarão atualizadas e qualquer uma delas poderá ser usada pelos clientes nas operações de leitura. A desvantagem mais importante desta abordagem é o fato de que em cada operação de escrita todas as réplicas tenham que ser atualizadas diretamente pelos clientes, tornando este tipo de operação mais lenta. A operação não é concluída até que todos os servidores não sejam atualizados.

Replicação atrasada: as operações de escrita nos arquivos são realizadas somente em um dos servidores, encarregado de enviar estas modificações, algum tempo depois, às demais réplicas (Figura 2.7). No entanto, as operações de leitura podem ser realizadas em qualquer um dos servidores.

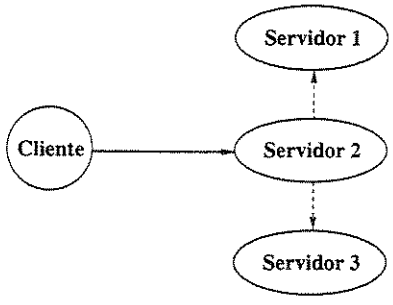

Figura 2.7: Replicação atrasada

Na replicação atrasada os clientes simplesmente enviam os pedidos a um servidor, que espalhará as mudanças nos servidores restantes. Esta abordagem resolve o problema

da demora nas operações de escrita da abordagem anterior. No entanto, usar a replicação atrasada no sistema pode gerar inconsistências entre as distintas réplicas.

Grupo de comunicação: todos os servidores formam um grupo de comunicação. As operações de escrita em um arquivo são enviadas ao grupo, encarregado de espalhá-las pelos diferentes servidores (Figura 2.8). As operações de leitura também são enviadas ao grupo para serem processadas.

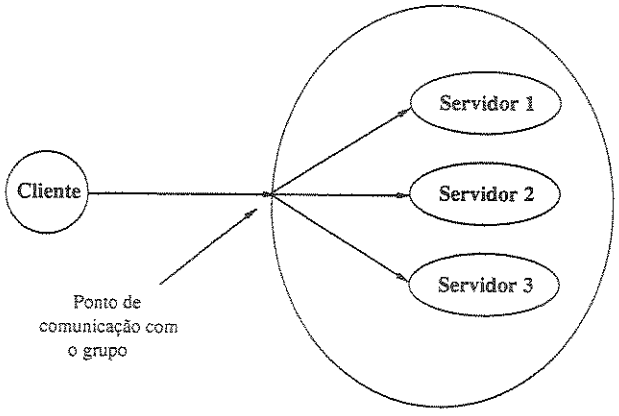

Figura 2.8: Grupo de comunicação

Usando os grupos de comunicações os clientes enviam todas suas operações ao grupo, que é o encarregado de atualizar todos os servidores. Geralmente são implementados mecanismos que permitem diminuir o tempo de espera dos clientes e manter a consistência entre os diferentes servidores.

2.7.2 Protocolos de Atualização

Diferentes protocolos de atualização são usados na replicação de servidores [Tan95]. A seguir são mostrados os mais importantes.

Vetores: geralmente usado para a implementação da replicação explícita. A idéia do algoritmo é ter dois vetores para cada unidade de replicação. O primeiro vetor armazena o identificador de todas as réplicas dos arquivos (RV - *Replication Vector*) e o segundo vetor armazena somente as réplicas disponíveis nesse momento (ARV - *Accessible Replication Vector*). Geralmente, as operações de escrita são espalhadas por todos os servidores do ARV e é escolhido um servidor preferencial para cada cliente que atenderá as suas operações de leitura. Este servidor preferencial é escolhido em cada usuário tendo em conta a carga dos servidores, proximidade física, dentre outras características. O RV e o ARV são iguais na ausência de falhas. Quando algum dos servidores ficar fora do ar, ele é eliminado do ARV e, logo após ser reiniciado e atualizado, ele é agregado novamente ao vetor.

Esta abordagem aumenta a disponibilidade da informação, mas as operações de escrita são lentas.

Cópia primária de replicação: um dos servidores é designado como servidor primário, os restantes serão servidores secundários. As operações de escrita são realizadas somente no servidor primário, que propagará mais tarde as mudanças aos servidores restantes. As operações de leitura podem ser realizadas tanto no servidor primário como nos secundários.

A desvantagem principal deste algoritmo é que, se o servidor primário ficar fora do ar, não é possível realizar atualizações no sistema. Além disso, possíveis inconsistências podem surgir entre as diferentes réplicas.

Votação: neste algoritmo, para realizar qualquer operação de escrita ou leitura, precisa-se da permissão de vários servidores. O número de servidores a serem usados depende da implementação realizada. Por exemplo, pode-se definir que é necessário receber autorização da metade dos servidores para realizar qualquer operação em um arquivo. Uma generalização deste algoritmo é conhecida como *quorum*. Ela consiste em prefixar quantos servidores serão usados para as operações de escrita (*quorum* de escrita) e quantos para as operações de leitura (*quorum* de leitura). A soma de ambos deve ser maior que o número total de servidores para garantir um bom funcionamento do algoritmo.

Este algoritmo elimina a desvantagem principal do protocolo anterior. No entanto, ele precisa de mais comunicação entre os diferentes componentes em cada uma das operações e os mecanismos para manter consistência são mais complicados, aumentando a complexidade da implementação.

Read any/write any: este é um algoritmo otimista. Ele pressupõe que todas as réplicas estão atualizadas, e portanto, qualquer operação pode ser realizada sobre qualquer uma delas. Para obter este resultado, existem diferentes mecanismos para garantir que todas as réplicas sempre estejam atualizadas, como por exemplo um protocolo de reconciliação, que sincroniza todas as réplicas do sistema caso elas fiquem desatualizadas. Este processo de reconciliação geralmente é realizado entre duas réplicas e cada réplica usa um vetor que armazena as versões dos arquivos. No processo de reconciliação, baseado nestes vetores, os arquivos são atualizados.

A principal desvantagem deste algoritmo é a dificuldade de se manter a consistência entre as diferentes cópias e garantir que todas sempre tenham a versão mais atualizada dos dados. É por esta razão que protocolos deste tipo são usados em sistemas que precisam de uma consistência fraca ou onde o número de réplicas é reduzido.

2.7.3 Replicação de Servidores no SAD_GSA

De acordo com as características do sistema de arquivos distribuído proposto (Figura 2.3), foram definidos três níveis de replicação. O primeiro nível é formado pelos Cache nos Módulos dos Clientes. No segundo nível estão os Grupos de Servidores para Aplicações que atendem diretamente os pedidos das aplicações. O terceiro nível é formado pelo Servidor de Arquivos Central, que mantém uma cópia de todos os arquivos do sistema, como mostra a Figura 2.9.

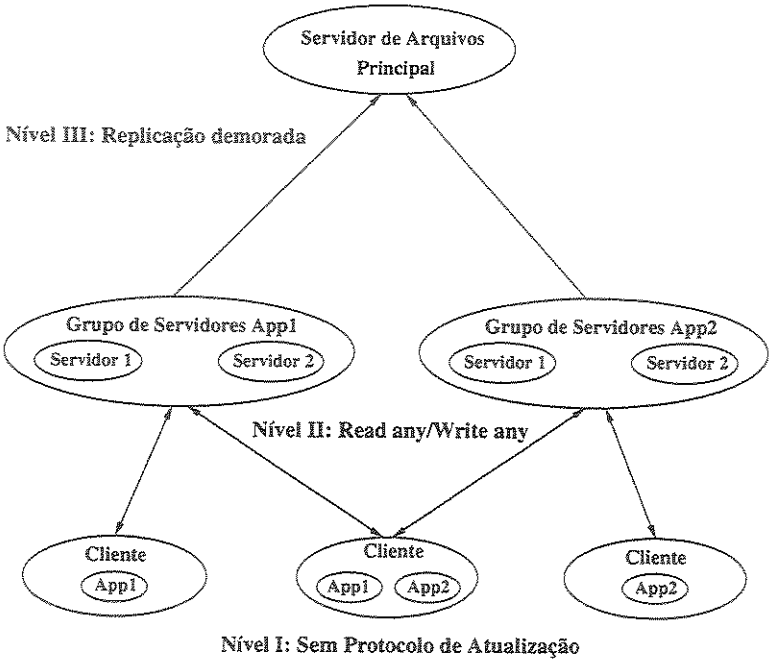

Figura 2.9: Níveis de replicação no SAD_GSA

No primeiro nível não é necessário implementar nenhum mecanismo para manter a consistência dos dados replicados, já que somente uma cópia de cada arquivo é criada no Módulo do Cliente (ver seção 2.10.3).

Devido ao número geralmente pequeno de servidores em um grupo, o que facilita a manutenção da coerência entre a informação que eles armazenam, é usado no segundo nível um modelo de Grupo de comunicação, baseado em um protocolo de atualização do tipo *Read any/ Write any*. Este é um protocolo otimista que pressupõe que todas as réplicas estão atualizadas e, portanto, qualquer operação pode ser realizada sobre qualquer uma delas. Isto é, qualquer um dos servidores pode receber qualquer requisição das tarefas da aplicação, aumentando a disponibilidade do sistema. Foi implementado um protocolo para manter a consistência dos dados espalhados pelos servidores do grupo que será detalhado na próxima seção.

No terceiro nível é usado um modelo de replicação atrasada. Somente a cada certo intervalo de tempo é que todas as mudanças que foram realizadas nos diferentes Grupos de Servidores para Aplicações são enviadas ao Servidor de Arquivos Central, diminuindo-se o tráfego na rede e a carga deste último. Como os arquivos mantidos no Servidor de Arquivos Central não são utilizados diretamente pelas aplicações paralelas, não existem restrições fortes de consistência, o que permite fazer as atualizações de forma atrasada.

Protocolo de Replicação

O protocolo de replicação envolve os servidores que formam cada Grupo de Servidores para Aplicações. Cada um destes grupos utiliza este protocolo para manter atualizadas

as informações mantidas nos seus servidores. Assim, qualquer um dos servidores em um grupo pode atender qualquer requisição feita pelas aplicações a ele associadas.

Cada Servidor de Arquivos para Aplicações, uma vez iniciado, realiza as funções mostradas a seguir.

- Comunica-se com o Resolvedor de Endereços para obter o endereço do Servidor de Arquivos Central e dos restantes servidores do seu grupo, se existirem.

- Cria o supervisor do Grupo de Servidores para Aplicações. Este subcomponente gerencia a carga de trabalho do novo servidor iniciado. Além disso, ele usa as informações obtidas dos demais servidores para gerenciar o grupo e difundir eventuais mudanças.

- Comunica-se com o Servidor de Arquivos Central para replicar os arquivos que serão utilizados. Na seção 2.5.2 vimos o espaço de nomes definido. O sistema de arquivos cria um diretório para cada usuário (*home directory*) e o usuário cria um diretório para cada aplicação por eles implementada. O diretório criado tem o mesmo nome da aplicação. Uma primeira solução seria que o servidor replicasse todos os arquivos contidos nesse *home directory*. No entanto, esta abordagem não é muito eficiente se o volume de informação do usuário é grande. Por isso, o sistema de arquivos usa uma replicação sob demanda. Cada vez que o usuário executa uma aplicação, somente os arquivos armazenados no diretório criado para a mesma são replicados nos servidores do grupo. Esta abordagem é mais eficiente e evita replicação desnecessária de arquivos.

- Por último, o novo Servidor de Arquivos para Aplicações registra-se no Resolvedor de Endereços, passando a formar parte do Grupo de Servidores para Aplicações. Este grupo atenderá todos os pedidos da aplicação ou das aplicações associadas a ele. O número de servidores dentro do grupo adapta-se às necessidades das aplicações que ele atende e às características do MPVC. Na seção 2.8.1 ter-se maiores detalhes desta característica dos Grupos de Servidores para Aplicações.

Logo depois de criado o Grupo de Servidores para Aplicações com pelo menos um Servidor de Arquivos para Aplicações, as tarefas das aplicações associadas ao grupo podem começar a enviar suas operações de leitura e escrita para serem resolvidas.

Para as operações de leitura os Módulos dos Clientes podem usar qualquer um dos servidores do grupo. O servidor escolhido (Servidor *Default* da aplicação, explicados em detalhes na seção 2.8.1) não precisa se comunicar com as demais réplicas, já que o protocolo de replicação garante que a réplica usada está atualizada. As operações de leitura são uma simples interação entre o Módulo do Cliente e o servidor escolhido.

Para as operações de escrita os passos a seguir são outros, já que as mudanças no arquivo no Servidor *Default* devem ser espalhadas pelos demais servidores do grupo, mantendo assim a consistência entre eles.

A seguir é mostrado o protocolo de atualização, que é executado quando uma aplicação faz uma operação de escrita como ilustrado na Figura 2.10.

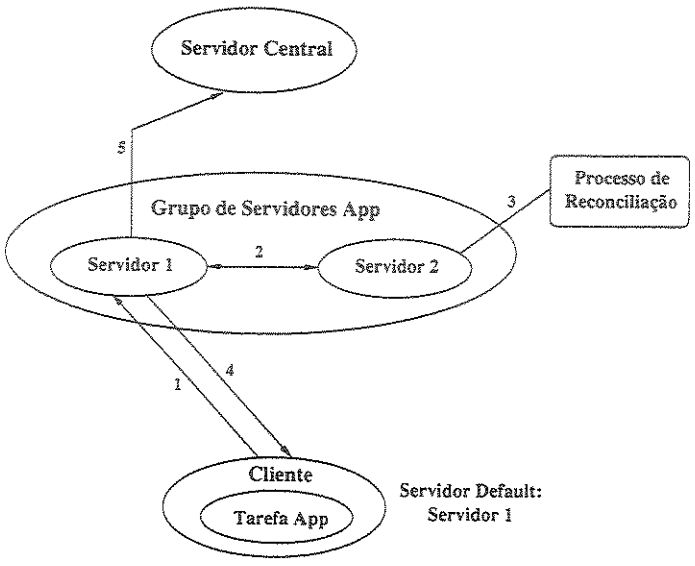

Figura 2.10: Protocolo de atualização para as operações de escrita

1. Uma tarefa da aplicação App envia uma mensagem com uma requisição de escrita para o Servidor *Default* da aplicação (Servidor 1).

2. O Servidor 1 recebe a mensagem e faz duas operações em paralelo: começa a processar o pedido e difunde a mensagem pelos demais servidores do grupo (Servidor 2).

3. O Servidor 2 recebe o pedido, e antes de processá-lo, analisa se existe uma possível inconsistência entre a sua cópia local do arquivo e a cópia do outro servidor. Se existir um conflito, o Servidor 2 começa um Processo de Reconciliação, que corrige o problema. Este processo é o encarregado de determinar a melhor solução para cada possível erro. Como veremos com mais detalhes na próxima seção, neste processo cada servidor pode resolver os conflitos sem a necessidade de se comunicar com os demais servidores do grupo. Também nessa seção tem-se maiores detalhes dos tipos de inconsistência que podem surgir e como elas são solucionadas.

4. Assim que for concluída a operação de escrita no Servidor 1, a resposta é enviada ao Módulo do Cliente. O Servidor 1 fica aguardando novos pedidos para serem processados.

5. Em intervalos regulares, todas as mudanças feitas nos arquivos mantidos no Grupo de Servidores para Aplicações são enviadas ao Servidor de Arquivos Central. Esta operação é explicada mais a frente.

Como o Módulo do Cliente recebe uma resposta assim que o Servidor *Default* processa o pedido, o sistema de arquivos responde rapidamente às operações de entrada/saída das aplicações, minimizando o tempo de espera das mesmas, sem esperar a difusão pelo grupo. Se houver algum problema de consistência entre as diferentes cópias espalhadas

no grupo, o Processo de Reconciliação será o encarregado de corrigí-lo, mas sem interferir significativamente no tempo de resposta do pedido em questão.

Processo de Reconciliação

O Processo de Reconciliação é executado por um servidor quando detecta que algum dos seus arquivos está desatualizado. Uma possível inconsistência pode surgir quando um mesmo arquivo é modificado simultaneamente em dois ou mais servidores. Para detectar esta inconsistência é usada a hora da última modificação do arquivo, que é um dos seus atributos. Portanto, seria necessário que os relógios de todos os servidores do grupo estivessem sincronizados. Sincronizar exatamente a hora dos relógios é um problema muito difícil de ser resolvido e considerando o fato de que a Internet é usada como rede de interconexão, o problema torna-se mais complexo. No entanto, no Processo de Reconciliação não é necessário manter uma sincronização estrita dos relógios dos servidores envolvidos no processo; basta somente saber entre dois eventos qual ocorreu primeiro. Portanto, é usado o Algoritmo de Sincronização de Lamport [Lam78, Lam79], que utiliza o conceito de relógio lógico para sincronizar os relógios dos servidores em um grupo.

Cada Servidor de Arquivos para Aplicações cria seu relógio lógico. Em cada comunicação entre dois servidores de um grupo são atualizados os seus relógios usando o Algoritmo de Lamport. Cada servidor, ao se comunicar com um outro servidor do grupo, inclui na sua mensagem a hora lógica do seu computador (obtida do seu relógio lógico). Quando o outro servidor recebe a mensagem, compara a hora lógica de seu computador com a hora enviada pelo outro servidor. Se a sua hora lógica for maior que a hora enviada na mensagem, ele não atualiza seu relógio. Mas se o sua hora lógica for menor que a hora enviada na mensagem, ele atualiza seu relógio lógico da forma mostrada a seguir.

```
hora lógica = hora enviada na mensagem + 1
```

O acréscimo do valor 1 procura refletir o fato de que a transmissão de uma mensagem pela rede sempre leva algum tempo. Desta forma, é possível obter uma sincronização relativa entre todos os servidores de um grupo usando um mecanismo muito simples, fácil de implementar e que não gera tráfego a mais na rede. Mais informações de como estes relógios foram implementados encontram-se na seção 3.5.4.

Para melhor descrever o Processo de Reconciliação, será usado o mesmo exemplo da Figura 2.10. O Servidor 1 recebe uma requisição de escrita em um arquivo e começa a processá-la, modificando o arquivo. Em paralelo à modificação do arquivo, o pedido é enviado aos demais servidores do grupo (Servidor 2). Simultaneamente, o Servidor 2 também recebe uma requisição de escrita diferente sobre o mesmo arquivo antes de receber a mensagem do Servidor 1 com as atualizações feitas nele. O Servidor 2, também em paralelo, começa a modificar a sua cópia do arquivo e espalha o pedido recebido no grupo. Vamos supor que as mensagens com as requisições foram recebidas pelo Servidor 1 no tempo t_1 e pelo Servidor 2 no tempo t_2, com $t_1 < t_2$. Quando o Servidor 1 recebe o pedido enviado pelo Servidor 2, ele o processa normalmente, já que ocorreu depois do pedido que ele mesmo recebeu do Módulo do Cliente. Mas quando o Servidor 2 receber o pedido

enviado pelo Servidor 1 detecta uma possível inconsistência entre os arquivos mantidos em ambos servidores. Está inconsistência é devida ao fato de que ele recebeu uma requisição em t_2 e depois disso, recebe a operação enviada pelo Servidor 1, que ocorreu em t_1, ou seja, é anterior à requisição enviada pelo seu Módulo do Cliente. Nesta situação as cópias mantidas por ambos os servidores podem estar inconsistentes.

Um caso particular da situação acima ocorre quando ambos os servidores modificam suas cópias do arquivo exatamente no mesmo instante, sendo impossível determinar qual deles começará o Processo de Reconciliação. Para solucionar este problema, o sistema de arquivos faz o seguinte. Cada Servidor de Arquivos para Aplicações tem uma posição dentro do grupo ao qual pertence. Esta posição pode ser determinada segundo vários critérios, como a ordem em que foram registrados, as características dos computadores onde são executados, dentre outras. No sistema de arquivos proposto é usada a ordem em que foram registrados os servidores. Assim, cada servidor usa a sua ordem relativa dentro do grupo para determinar qual é a sua prioridade, isto é, qual é a cópia que será considerada. O servidor que tiver a posição mais alta iniciará o Processo de Reconciliação.

A Figura 2.11 traz um exemplo de como trabalha o Processo de Reconciliação.

A seguir são mostradas as operações realizadas no Processo de Reconciliação.

- O servidor que inicia o Processo de Reconciliação verifica se ambas as operações modificam alguma parte comum do arquivo. Se não for o caso, o servidor simplesmente processa a operação enviada pelo outro servidor, já que a ordem em que elas são processadas não influencia no estado final do arquivo.

- Se as operações modificarem uma parte comum do arquivo, o servidor que iniciou a reconciliação não pode processar o pedido enviado pelo outro servidor, já que as duas cópias ficariam diferentes. Neste caso, ele teria que determinar o estado final do arquivo no outro servidor para atualizar sua própria cópia. O servidor faz as seguintes operações a seguir.

    1. Calcula a interseção de ambas as operações, isto é, determina quais são os bytes no arquivo que ambos os servidores modificam. Esses bytes precisam ser corrigidos;
    2. Determina, da operação que chegou do outro servidor, quais bytes não estão incluídos na interseção calculada. Esses bytes não precisam de correções.
    3. O servidor corrige na cópia do arquivo por ele mantida apenas os bytes contidos na interseção calculada, nas posições corretas (copiando os bytes corretos da mensagem que chegou primeiro).

Atualização do Servidor de Arquivos Central

Em intervalos regulares, todas as mudanças nos Servidores de Arquivos para Aplicações são enviadas ao Servidor de Arquivos Central, evitando que o mesmo fique desatualizado. A forma mais simples de atualizar o Servidor de Arquivos Central é que cada Servidor de Arquivos para Aplicações enviasse suas modificações de forma independente. No entanto,

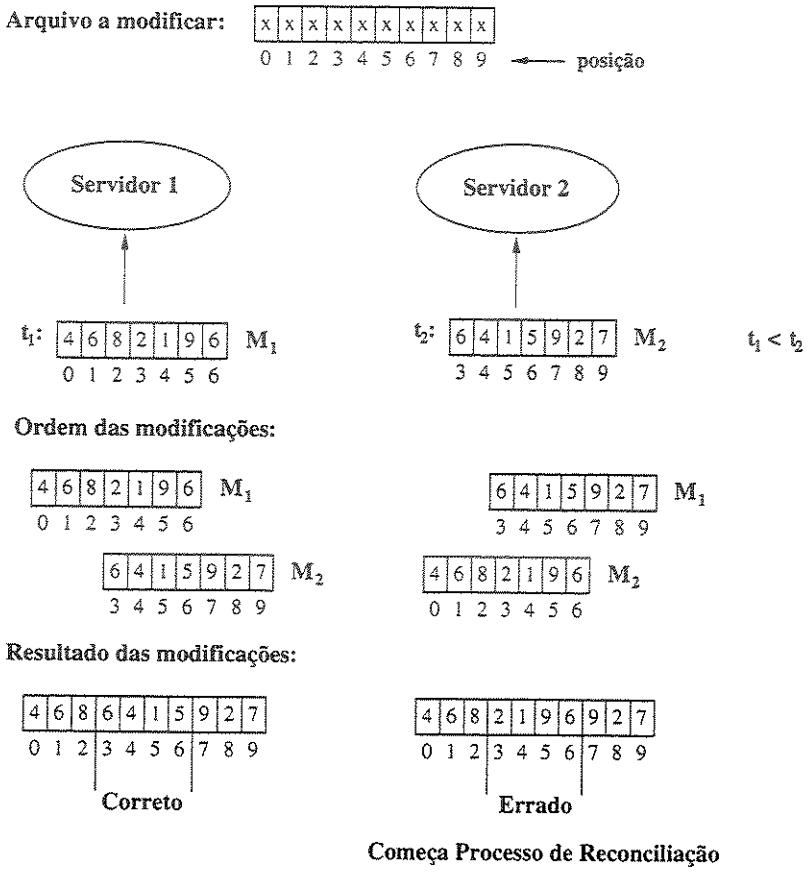

Figura 2.11: Processo de Reconciliação de Servidores

embora esta seja uma solução simples, o sistema não pode garantir que as mudanças sejam feitas na mesma ordem em que foram enviadas devido a atrasos imprevisíveis das mensagens na rede. Por exemplo, vamos considerar um Grupo de Servidores para Aplicações formado por dois servidores, S_1 e S_2. S_1 recebe uma requisição e momentos depois S_2 recebe outra requisição (ambas são propagadas pelo grupo). Se os servidores enviarem estas mudanças para o Servidor de Arquivos Central de forma independente, o sistema não pode garantir que as operações sejam recebidas na ordem adequada, embora sejam enviadas pelos servidores do grupo na ordem certa.

Para solucionar este problema, também é usada a ordenação dos servidores dentro de um grupo para determinar qual dos servidores enviará as mudanças realizadas nos arquivos ao Servidor de Arquivos Central. Como todas as mudanças são espalhadas pelo grupo, qualquer servidor pode fazer a atualização, em particular o servidor que ocupar a primeira posição. Desta forma, o sistema pode garantir que todas as operações sejam realizadas na mesma ordem em que foram recebidas.

2.8 Balanceamento de Carga

Para que a utilização da replicação de servidores garanta um aumento na eficiência do sistema, é necessário definir também um mecanismo de balanceamento de carga de trabalho entre os diferentes servidores.

2.8.1 Balanceamento de Carga no SAD_GSA

O sistema de arquivos proposto define um mecanismo de balanceamento totalmente distribuído. Para obter um bom balanceamento entre os diferentes servidores existe uma estreita relação entre três dos principais componentes do sistema de arquivos: o Resolvedor de Endereços, os Módulos dos Clientes e os Grupos de Servidores para Aplicações.

O ponto de partida do mecanismo de balanceamento é o Resolvedor de Endereços. Como já foi explicado em seções anteriores, todos os servidores de arquivos se registram neste componente para ficarem disponíveis para o sistema. Cada servidor registra o endereço IP do computador onde está sendo executado, as características do computador (tipo de processador e quantidade de memória), a velocidade do enlace, o seu estado, dentre outras informações. O estado do servidor é determinado pela carga de trabalho. Fica claro que, no momento do registro, o servidor está descarregado, já que ainda não começou a receber requisições das aplicações.

Uma vez registrado, o servidor começa a formar parte de um Grupo de Servidores para Aplicações, compartilhando a carga de trabalho do mesmo. Cada Servidor de Arquivos para Aplicações, como foi explicado em seções anteriores, criará um subcomponente chamado de supervisor do Grupo de Servidores para Aplicações e encarregado de gerenciar o grupo ao qual o servidor pertence. Umas das funções deste supervisor é detectar mudanças na carga de trabalho do Servidor de Arquivos para Aplicações que o criou e informá-las aos demais servidores do grupo. Para determinar esta carga, a primeira coisa que o supervisor faz quando é criado (momento no qual a carga de trabalho do servidor é baixa, devido a

que ainda não está recebendo pedidos das aplicações) é se comunicar com o servidor para determinar o tempo que ele demora em responder. Este tempo é chamado de tempo padrão inicial. Uma vez que o servidor forme parte de um Grupo de Servidores para Aplicações e comece a receber pedidos, o supervisor enviará, em intervalos regulares, uma mensagem de teste ao servidor, que responderá o mais rápido que puder. Dependendo do tempo que o servidor demore em responder, o supervisor determina a carga do Servidor de Arquivos para Aplicações. Como este servidor e o supervisor do Grupo são executados no mesmo computador, o tráfego na rede não afeta esta medida de tempo. Toda vez que o supervisor detecta uma mudança na carga de trabalho do Servidor de Arquivos para Aplicações, ele a divulga para o grupo e para o Resolvedor de Endereços. Desta forma, sempre que um servidor mude sua carga de trabalho, todos os demais ficam sabendo das mudanças. A seguir são mostrados os possíveis estados dos Servidores de Arquivos para Aplicações.

Descarregado: o servidor tem uma carga de trabalho muito baixa ou nenhuma carga. Isto é verificado por um tempo de comunicação até X% acima do tempo padrão inicial. O valor de X dependerá da implementação, mas valores típicos estarão entre 0 e 20%.

Carregado: o servidor tem uma carga de trabalho alta, mas que ainda pode gerenciar com eficiência. Neste caso o tempo de comunicação deverá estar entre X e Y% acima do tempo padrão inicial (Y aproximadamente 50%).

Sobrecarregado: o servidor tem uma carga maior que a que ele pode atender, isto é, o tempo de comunicação ultrapassa o tempo padrão inicial em mais de Y%.

Servidores *Default* nos Módulos dos Clientes

Quando um novo computador entra no MPVC, é descarregada nele uma plataforma básica de software. Esta plataforma permitirá às aplicações usar os recursos oferecidos pelo computador virtual. Junto com esta plataforma básica é descarregado um outro componente do sistema de arquivos, o Módulo do Cliente (seção 2.4.4). Este componente permitirá às aplicações interagir com o serviço de arquivos. Cada aplicação executada em um computador utilizará um determinado Grupo de Servidores para Aplicações. Além disso, em cada Módulo do Cliente, para cada aplicação sendo executada no MPVC, será escolhido um dos servidores do Grupo de Servidores para Aplicações a ele associado que atenderá todos os seus pedidos. Este servidor é chamado de Servidor *Default* e pode variar dependendo de futuras mudanças na carga dos servidores do grupo. É aqui que a implementação de um bom mecanismo de balanceamento de carga poderá aumentar a eficiência do sistema, diminuindo o tempo das operações de entrada/saída.

Assim que uma aplicação é submetida ao sistema, o Módulo do Cliente se comunica com o Resolvedor de Endereços para obter as informações de todos os servidores disponíveis no grupo que vai atender seus pedidos. O Módulo do Cliente recebe todas estas informações e determina dentre eles qual será o seu Servidor *Default*. Uma vez que os Servidores *Default* sejam escolhidos, as aplicações poderão interagir com eles para processarem seus pedidos. A medida que as tarefas das aplicações espalhadas pelos diferentes computadores que formam o MPVC enviem seus pedidos aos servidores, a carga de trabalho destes variará e será

preciso modificar os Servidores *Default* de alguns dos Módulos dos Clientes para manter a carga entre todos os servidores equilibrada. Os Módulos dos Clientes não precisarão de uma nova comunicação com o Resolvedor de Endereços para esta atualização, já que cada servidor em um grupo mantém as informações dos demais integrantes do grupo e podem oferecer aos Módulos dos Clientes estas informações. Desta forma, evita-se comunicações excessivas com o resolvedor, que o tornaria um possível ponto de gargalo do sistema. Além disso, são os Módulos dos Clientes que tomam as decisões na escolha do Servidor *Default*. O Resolvedor de Endereços ou os Servidores de Arquivos para Aplicações somente fornecem os dados da carga de trabalho do grupo; é o Módulo do Cliente o encarregado de determinar qual será o melhor servidor que atenderá seus pedidos.

Para cada operação recebida pelos Servidores de Arquivos para Aplicações, dependendo da sua carga de trabalho, são realizadas as seguintes funções.

Servidor descarregado: o Servidor de Arquivos para Aplicações processa o pedido e envia a resposta ao Módulo do Cliente.

Servidor carregado: o Servidor de Arquivos para Aplicações processa o pedido, mas, se existe no grupo outro servidor com carga menor (descarregado), envia de carona com a resposta a lista de estados dos servidores do grupo, já que provavelmente a carga mudou e o Módulo do Cliente ainda não percebeu essas mudanças. É decisão do Módulo do Cliente modificar ou não seu Servidor *Default*.

Servidor sobrecarregado: O Servidor de Arquivos para Aplicações procura um outro servidor no grupo com menor carga. Duas situações são possíveis:

Tem-se pelo menos um servidor com carga menor: neste caso o servidor reenvia o pedido para o servidor com menor carga. Esse outro servidor processa o pedido e junto com a resposta também envia a lista de estados dos servidores para que o Módulo do Cliente determine um novo Servidor *Default*. Neste caso, é obrigatória a modificação do Servidor *Default*.

Todos os servidores estão carregados: neste caso, o servidor não atende o pedido e responde ao Módulo do Cliente que todos os servidores estão carregados. O Módulo do Cliente envia a resposta à aplicação, que determinará o que fazer nesse caso. O mais conveniente é repetir a operação passado um certo tempo arbitrário, já que o fato de todos os servidores ficarem carregados será detectado pelo sistema, que automaticamente criará um novo Servidor de Arquivos para Aplicações para formar parte do grupo, diminuindo a carga do mesmo.

2.8.2 Função de Distribuição de Carga

A base deste mecanismo de balanceamento está em uma função, que chamaremos de Função de Distribuição de Carga, encarregada de determinar, de uma lista de servidores, qual responderá melhor aos pedidos de uma aplicação. Esta função baseia sua decisão fundamentalmente na carga relativa de trabalho dos servidores e na latência nas comunicações. A seguir será explicado como ela trabalha.

Cada vez que a função é chamada, ela recebe uma lista de servidores como parâmetro. Esta lista é percorrida e determina-se o conjunto de servidores com capacidade para atender o pedido (servidores descarregados ou carregados). Se o conjunto está formado por um servidor só, ele é retornado como melhor servidor da lista. Se o conjunto está formado por dois ou mais servidores, o processo é mais complexo e novos aspectos são considerados, tais como as características dos computadores onde os servidores são executados e a proximidade física. Neste caso é aplicada uma fórmula matemática (baseada em pesos) para determinar o melhor servidor. São considerados dois fatores: características do computador (CPU - velocidade do processador, mem - quantidade de memória) e velocidade medida do enlace (*link*). Cada um destes fatores recebe um peso dentro da fórmula, como é mostrado a seguir. A escolha destes pesos dependerá da implementação realizada.

$$v = (a * CPU) + (b * mem) + (c * link) \qquad com \;\; a + b + c = 1 \tag{2.1}$$

Para cada servidor na lista aplica-se a equação 2.1, obtendo-se um determinado valor. Valores maiores significam melhores servidores.

Mas este método tem uma falha. Analisemos a seguinte exemplo. Vamos supor que 16 Módulos dos Clientes vão escolher o seu Servidor *Default* e todos recebem a lista de todos os servidores disponíveis. Vamos supor também que esta lista estará formada por dois servidores com a mesma carga de trabalho. Cada Módulo do Cliente aplicará a fórmula acima para escolher o melhor servidor. Como todos Módulos dos Clientes usam a mesma fórmula com os mesmos dados iniciais, todos vão escolher o mesmo servidor. Portanto, o sistema vai ter 16 Módulos dos Clientes usando o servidor mais rápido e nenhum utilizando o outro servidor que, embora seja mais lento, tem uma certa capacidade e uma melhor distribuição de carga aumentaria a eficiência do sistema.

Voltando ao exemplo, vamos supor novamente que o primeiro servidor, que chamaremos de S_1, obtém um valor de $v_1 = 30$ no cálculo da equação 2.1, e o segundo servidor, que chamaremos de S_2, obtém um valor de $v_2 = 10$. Ou seja, o valor obtido por S_1 é 3 vezes maior que o valor obtido por S_2. Um balanceamento de carga mais adequado neste exemplo ocorreria se, por cada Módulo do Cliente que escolhesse S_2 como Servidor *Default*, 3 Módulos dos Clientes escolhessem S_1 como o melhor servidor. No exemplo dos 16 Módulos dos Clientes, 12 deles deveriam escolher S_1 e os 4 restantes, S_2, distribuindo melhor a carga entre os dois servidores.

Mas como cada Módulo do Cliente aplica sua fórmula e escolhe seu Servidor *Default* sem se comunicar com os demais, é impossível que cada um deles saiba a escolha feita pelos outros. Por isso, a Função de Distribuição de Carga, além de aplicar a equação 2.1, utiliza o algoritmo seguinte para escolher qual será o Servidor *Default* para cada Módulo do Cliente.

$$v_t = \sum_{i=1}^{n} v_i \tag{2.2}$$

$$f = \textit{número aleatório escolhido entre } 0 \textit{ e } v_t \tag{2.3}$$

$$Servidor\ Default = \begin{cases} Servidor\ 1,\ se\ 0 \leq f \leq v_1 \\ Servidor\ 2,\ se\ v_1 < f \leq v_1 + v_2 \\ \ldots\ldots \\ Servidor\ n,\ se\ v_1 + v_2 + \ldots + v_{n-1} < f \leq v_t \end{cases} \quad (2.4)$$

Desta forma, tem-se uma distribuição probabilística na escolha do Servidor *Default*. Os servidores cujos valores na equação 2.1 são maiores terão mais possibilidades de serem escolhidos como Servidores *Default* pelos diferentes Módulos dos Clientes do sistema. Mas os servidores com menores valores na equação 2.1 também terão a possibilidade de serem escolhidos, resultando em uma melhor distribuição de carga entre os servidores disponíveis.

2.9 Reconfiguração Automática

Quando um servidor fica fora do ar, cada um dos componentes relacionados com ele ficam sabendo da falha. A seguir será explicado o que cada componente do sistema faz para detectar uma falha em um servidor e como reage ante a ocorrência da mesma.

Na seção 2.8.1 mostrou-se o mecanismo de balanceamento implementado e a função do supervisor do Grupo de Servidores para Aplicações, que é o subcomponente que todo Servidor de Arquivos para Aplicações possui para gerenciar o Grupo de Servidores para Aplicações ao qual pertence. Umas das tarefas deste supervisor é determinar quando um servidor do grupo ficou fora do ar e informar o fato ao Servidor de Arquivos para Aplicações que o criou. Para isto, em intervalos regulares, o supervisor envia uma mensagem de teste a cada servidor do grupo e espera por resposta. Se todos os servidores respondem, está tudo bem no grupo. Se algum servidor não responder, o supervisor repete o processo com este servidor mais duas vezes. Caso o servidor continue sem responder, o supervisor considera que está fora do ar e comunica-se com o Resolvedor de Endereços para cancelar o registro anteriormente feito. Note que o supervisor não necessitará se comunicar novamente com os demais servidores, já que os supervisores de cada um deles fazem a mesma coisa e detectam a falha por si mesmos.

O Resolvedor de Endereços também implementa um mecanismo semelhante ao implementado pelos supervisores para saber se todos os servidores registrados estão em ordem. Tem-se duas situações para as quais é necessário implementar este mecanismo no Resolvedor de Endereços.

Grupo de Servidores para Aplicações formado por um servidor só: neste caso, se o único servidor do grupo fica fora do ar, ninguém pode propagar a falha para o Resolvedor de Endereços, ficando este desatualizado.

Falhas simultâneas: se todos os servidores de um determinado Grupo de Servidores para Aplicações saírem do ar simultaneamente, ninguém pode avisar ao resolvedor, ficando este desatualizado.

O resolvedor, para cada servidor registrado, armazena a hora da última comunicação entre eles. Se em um determinado período de tempo (definido pelo sistema) não houver

uma nova comunicação entre eles, o resolvedor envia uma mensagem para determinar se o servidor ainda está sendo executado. Se não receber resposta, depois de repetir o processo mais duas vezes, o resolvedor determina que servidor está fora do ar e o elimina da lista de servidores registrados.

Outra forma de se implementar este mecanismo, tanto no Resolvedor de Endereços como nos supervisores dos Grupos de Servidores para Aplicações, é fazer com que o sistema gere eventos sob determinadas circunstâncias, como por exemplo, quando um determinado servidor deixa de estar disponível. No entanto, para isto é necessário que a plataforma de software básico utilizada forneça suporte para a geração e captação de eventos.

Por último, os Módulos dos Clientes também podem detectar que um servidor está fora do ar se tentarem se comunicar com ele e não obtiverem respostas. Mas, neste caso, o Módulo do Cliente simplesmente muda o seu Servidor *Default* (segundo o protocolo explicado anteriormente). Se todos os servidores do grupo que atendem os pedidos de uma determinada aplicação estiverem fora do ar, o Módulo do Cliente pode enviar suas requisições diretamente ao Servidor de Arquivos Central.

2.10 Uso de Cache nos Clientes

O mecanismo mais utilizado para aumentar a eficiência dos sistemas de arquivos é o cache no cliente. Usando cache no cliente diminui-se o tempo das operações de escrita e leitura, o tráfego na rede e a carga dos servidores. No entanto, ele gera um problema difícil de ser resolvido: manter todas as cópias dos arquivos atualizadas. Criar cópias dos arquivo nos diferentes caches dos clientes somente para leitura é muito simples e não gera inconsistências. O problema começa quando um dos clientes decide modificar o arquivo. A seguir são apresentados os tipos de escritas que são normalmente usados para minimizar o problema da consistência [Tan95].

2.10.1 Tipos de Escritas

Escrita síncrona: quando uma operação de escrita é realizada, as mudanças feitas são atualizadas no cache do cliente e enviadas ao servidor imediatamente, isto é, uma operação de escrita não retorna até que os dados sejam atualizados no cache e no servidor. Esta é a abordagem mais consistente, já que o servidor sempre tem a cópia mais atualizada dos dados. Não entanto, é a mais lenta e o cliente tem que esperar que os dados sejam atualizados no servidor para realizar uma nova operação. Além disso, ela aumenta o tráfego na rede.

Escrita atrasada: neste caso, a operação de escrita retorna logo que as mudanças são feitas no cache do cliente. Algum tempo depois, as mudanças são feitas no servidor. Esta abordagem diminui o tempo de espera dos clientes e evita a escrita no servidor caso o arquivo seja apagado um tempo depois da sua criação (arquivo temporário), diminuindo o tráfego na rede. No entanto, esta abordagem pode criar inconsistências entre as diferentes cópias mantidas por vários clientes e, se o cliente cair antes de enviar os dados ao servidor, os dados são perdidos.

Escrita assíncrona: esta é uma abordagem intermediária entre as duas anteriores. Quando uma operação de escrita é realizada, as mudanças feitas no arquivo são escritas no cache e enviadas ao servidor. Entretanto, a operação de escrita retorna imediatamente após os dados serem armazenados no cache do cliente. Esta abordagem minimiza a possibilidade de perda de dados, mas não minimiza a carga do servidor e o tráfego na rede.

2.10.2 Protocolos de Cache

A seguir são introduzidos os principais tipos de protocolos para implementar o cache no cliente.

- Cache só de leitura.
- Protocolos *stateful*:
  1. *Callbacks*.
  2. *Leases*.
- Protocolos *stateless*:
  1. Baseados em tempo.
  2. *Polling*.
  3. Baseados em *tokens*.

Os protocolos que usam um cache de só leitura são muitos simples. Os arquivos somente podem ser abertos ou para leitura ou para escrita. Se o arquivo é aberto só para leitura, uma cópia dele é armazenada no cache dos clientes que o utilizarem. Já os arquivos que sejam abertos para serem modificados não podem ser levados até o cliente; eles são modificados diretamente no servidor. Esta abordagem é muito simples de ser implementada e não tem problemas de inconsistência nos dados. No entanto, ela não permite que um arquivo seja aberto simultaneamente para leitura e escrita e as operações de escrita não aproveitam as vantagens do uso do cache. A proposta é adequada para sistemas que utilizam arquivos majoritariamente para leitura.

Os protocolos *stateful* são aqueles onde os servidores mantêm informações de cada um dos clientes que estão fazendo uso dos arquivos. A maior vantagem destes algoritmos é que geralmente eles mantêm uma consistência estrita entre as distintas cópias dos arquivos. Os maiores inconvenientes são as dificuldades de se obter escalabilidade e tolerância a falhas. Além disso, estes protocolos necessitam modificar os clientes para que recebam mensagens do servidor.

O protocolo *stateful* mais utilizado é o *callback*. É usado um mecanismo chamado de *callback promise* para garantir que a informação armazenada no cache do cliente esteja atualizada. Quando um cliente abre um arquivo, uma cópia é colocada no seu computador e junto com ele, um *callback promise*, isto é, uma promessa de que o servidor notificará ao

cliente se o arquivo mudar e, portanto, a cópia no seu cache ficar desatualizada. A principal vantagem deste protocolo é que mantém uma consistência estrita dos diferentes caches. No entanto, não é escalável nem tolerante a falhas.

Outro protocolo muito similar ao *callback* usa uma política de cache baseada em *leases*. Quando um cliente abre um arquivo, junto com os dados ele recebe um *lease*, que é uma espécie de *ticket* que permite usá-los por um período de tempo determinado. Durante esse período de tempo o cliente tem a certeza de que o arquivo no seu cache está atualizado e qualquer mudança nele será notificada pelo servidor. Uma vez que o período de tempo expire, o cliente tem que contatar o servidor para poder utilizá-lo novamente, atualizando-o caso necessário. Embora sejam muitos similares, em um *lease* a recuperação ante possíveis falhas é mais simples. No entanto, este esquema também não é escalável.

Por outro lado, nos protocolos *stateless* os servidores não mantêm informação nenhuma dos clientes que estão fazendo uso dos arquivos. Geralmente, estes protocolos são mais tolerantes a falhas e mais escaláveis que os protocolos *stateful*. No entanto, muitos deles não garantem uma consistência estrita entre as cópias.

Um dos protocolos deste tipo é o protocolo baseado em tempo. Este protocolo assume que um arquivo no cache vai estar atualizado por um tempo determinado. Quando o tempo expirar, a cópia do arquivo é considerada não válida e uma próxima utilização do arquivo requer uma comunicação com o servidor para atualizá-lo. A maior vantagem deste protocolo é que diminui as iterações entre os clientes e os servidores, mas pode gerar inconsistência no período de tempo que o arquivo é mantido no cliente, já que ele pode ser modificado por um outro cliente, e portanto, a cópia no cache pode ficar desatualizada.

Já nos protocolos de *polling*, os clientes periodicamente perguntam ao servidor se um determinado arquivo no seu cache está atualizado. São implementados diferentes algoritmos para determinar a freqüência das interações com o servidor. Geralmente é usado o tempo que os arquivos estão no cache para determinar essa freqüência. Uma vez transcorrido o tempo antes mencionado, o arquivo é considerado não válido e a próxima operação é realizada no servidor. Este mecanismo também não garante uma consistência estrita entre as diferentes cópias, mas diminui o tráfego na rede.

O último protocolo que será apresentado é baseado em *tokens*. Os *tokens* oferecidos aos clientes podem ser para leitura e escrita. Ter um *token* de leitura garante ao cliente que a cópia que ele tem do arquivo está atualizada. Um *token* de escrita permite aos clientes acessos de escrita aos arquivos. Cada *token* tem um tempo de validade. Transcorrido esse tempo, o cliente precisa se comunicar com o servidor para renovar o tempo. Caso eles não o fizerem, o protocolo não garante que a sua cópia esteja atualizada. Se um cliente abre um arquivo para leitura, ele recebe um *token* de leitura por um tempo determinado. Nesse tempo, o cliente tem a certeza de que o arquivo esta atualizado. Outros clientes podem abrir o arquivo para leitura sem problemas. Se algum cliente abre o arquivo para escrita, o servidor bloqueia o arquivo para posteriores operações e espera que todos os *tokens* dos clientes que estão usando o arquivo expirem. Uma vez que os tempos expirem, o novo cliente ganha o *token* de escrita e pode modificar o arquivo. Nenhum outro cliente pode usar o arquivo para leitura ou escrita até que o tempo do *token* de escrita expirar. Este mecanismo garante a consistência entre as cópias, mas não permite acesso concorrente ao

mesmo arquivo para escrita.

2.10.3 Cache no Módulo do Cliente no SAD_GSA

O SAD_GSA também utiliza cache nos Módulos dos Clientes. Quando uma determinada tarefa de uma aplicação utiliza um determinado arquivo, é criada uma cópia do arquivo no computador onde a tarefa está sendo executada com o objetivo de minimizar o tempo das operações sobre o arquivo e aumentar a eficiência do sistema. Antes de determinar qual técnica para manter a consistência dos caches será utilizada, definiremos a semântica do sistema de arquivos proposto, ou seja, as regras usadas pelo sistema para obter acesso aos arquivos.

Semântica do Sistema de Arquivos

Sempre que vários usuários usem um mesmo arquivo ou grupo de arquivos simultaneamente, é necessário definir as regras (ou semântica) para ler e escrever neles e evitar possíveis problemas e inconsistências no futuro.

A semântica mais desejada por qualquer sistema de arquivos que compartilhe informação é conhecida como Semântica Unix. Nesta semântica qualquer mudança feita em um arquivo por qualquer processo é vista imediatamente pelos demais processos que estão utilizando o arquivo. Para conseguir sempre devolver o valor mais recente, o sistema deve manter uma organização global dos pedidos.

A semântica utilizada no sistema de arquivo depende da forma como ele esteja implementado. Em sistemas de arquivos centralizados ou em sistemas de arquivos distribuídos que não usem nem replicação nem cache no cliente, manter uma semântica Unix é simples. No entanto, para sistemas distribuídos que usem de forma intensiva mecanismos de replicação e cache no cliente, manter esta semântica é uma tarefa muito complexa, às vezes impossível, ou impõe uma sobrecarga inaceitável no sistema. E se estes sistemas distribuídos estão baseados em redes lentas e heterogêneas como a Internet, este problema é ainda maior.

A semântica mais utilizada pela maioria dos sistemas de arquivos distribuídos é conhecida como Semântica de Sessão. Esta semântica diz que mudanças em um arquivo aberto somente são vistas pelo processo que abriu e modificou o arquivo. Assim que o arquivo é fechado, as mudanças são visíveis para os demais processos. Isto é, quando um arquivo é aberto por um processo, ele será copiado de volta no servidor somente quando for fechado. Neste momento, outros processos poderão ver as mudanças feitas nele. Usando uma semântica de sessão, os sistemas de arquivos minimizam as dificuldades para manter a consistência entre as diferentes réplicas.

Existem outras semânticas, como a Semântica de Arquivos Imutáveis e a Semântica de Transações. No entanto, as duas semânticas detalhadas acima são mais utilizadas por serem mais vantajosas.

O sistema de arquivos proposto não deve ser somente distribuído; ele precisa também ser adeqüado para plataformas maciçamente paralelas baseadas na Internet. Estas duas

características tornam mais difícil a manutenção de uma semântica Unix. Portanto, o SAD_GSA utiliza uma semântica de sessão.

Definição do Cache

Em seções anteriores foi definida a forma como são utilizados os arquivos no sistema de arquivos proposto. As aplicações usam *streams* de entrada para ler dados dos arquivos em forma seqüencial e *streams* de saída para escrever dados neles. No entanto, para determinar como o cache será definido e manipulado pelo sistema, além da forma como os arquivos são utilizados, também é importante definir como as aplicações vão interagir com eles. Esta forma de interação depende de como diferentes tarefas compartilham um mesmo arquivo ou grupo de arquivos.

De forma geral, podemos ter duas formas de interação possíveis no sistema, as quais são citadas a seguir.

Acesso de leitura e escrita a um mesmo arquivo: o sistema permite que um mesmo arquivo seja aberto para leitura e para escrita por várias tarefas simultaneamente.

Acesso de leitura ou escrita a um mesmo arquivo: o sistema permite que um mesmo arquivo seja aberto para leitura ou para escrita por várias tarefas simultaneamente, mas não permite que seja aberto para ambas operações ao mesmo tempo.

A abordagem mais geral é permitir que um mesmo arquivo possa ser utilizado simultaneamente para leitura e para escrita, isto é, permitir que, se uma tarefa abre um arquivo para leitura, uma outra tarefa também possa abrir simultaneamente o mesmo arquivo para escrita, ou vice-versa. Esta abordagem é a mais abrangente é permite modelar qualquer forma de interação entre uma aplicação e os arquivos que ela utiliza. No entanto, pelo fato de se estar usando uma semântica de sessão, esta abordagem introduz alguns pontos a considerar. A primeira questão diz respeito à geração de cópias de um mesmo arquivo. Se uma tarefa abre um arquivo, mesmo que seja só para leitura, o sistema tem que criar uma cópia do mesmo para a tarefa. Isto é devido ao fato de que uma outra tarefa pode abrir o arquivo e modificá-lo, e portanto, se a primeira tarefa não tem sua própria cópia do arquivo, a semântica fica indefinida. Por exemplo, em *streams* de entrada é muito comum e extremamente útil implementar uma operação chamada de *mark - reset* (esta operação é suportada pelo sistema de arquivos proposto, como é mostrado no Apêndice C). Se o stream suporta esta operação, quem estiver usando-o pode usar o *mark* para marcar uma determinada posição no *stream* de dados. Uma posterior chamada ao *reset* reposicionaria novamente o *stream* na posição marcada. Todas as leituras feitas a partir dessa posição têm que retornar os mesmos dados. A única forma de manter a semântica definida nesta abordagem é mantendo cópias separadas dos arquivos para cada tarefa. Além disso, o arquivo tem que ser replicado por completo no cache do Módulo do Cliente, já que se é feito por blocos, o sistema não tem como garantir que os blocos que ficaram no servidor não serão modificados.

Outra questão que surge nesta abordagem é onde manter estas cópias. O sistema pode gerar diferentes cópias e mantê-las no próprio servidor ou em um outro componente

específico para isso. Seja onde for, é preciso haver uma capacidade grande de disco para conseguir manter a semântica definida. Se estas cópias são mantidas no Módulo do Cliente, é necessário que os computadores onde eles são executados tenham caches grandes para armazenar todos os arquivos que utilizarem. Note que um cache pode até conter mais de uma cópia de um mesmo arquivo. Por exemplo, temos a tarefa 1 que abre o arquivo X para leitura. Logo após, a tarefa 2 abre-o para escrita, modificando-o. E por último, a tarefa 3 também abre o arquivo para leitura. Nesse caso, uma cópia do arquivo para cada tarefa é necessária. O que acontece se um determinado computador não tiver disco nem memória suficiente para armazenar as cópias completas dos arquivos? Será que as tarefas executadas podem usar diretamente os arquivos do servidor? A resposta é negativa. O sistema poderia criar uma cópia do arquivo e mantê-la no servidor ou em outro ponto, mas a tarefa não poderia usar o arquivo diretamente, já que a semântica ficaria indeterminada pelo fato de que uma outra tarefa poderia modificar o arquivo.

Todos estes problemas ficariam resolvidos ou minimizados se o sistema não permitisse acesso de leitura e escrita simultâneos a um mesmo arquivo. Se um arquivo é aberto para leitura ou para escrita, ele só pode ser usado para este tipo de operação. Para que ele possa ser usado para outro tipo de operação, ele deve ser explicitamente fechado. Com esta abordagem, se um arquivo é aberto para leitura, é possível usar somente uma cópia do mesmo por computador. Todas as tarefas sendo executadas nesse computador podem usar o mesmo arquivo simultaneamente, mantendo a semântica definida. Somente se o arquivo fosse aberto para escrita, o sistema teria que fazer uma cópia do mesmo para cada tarefa que quisesse modificá-lo. No entanto, é possível fazer uma otimização no caso de uso dos arquivos como *stream* de saída. Cada tarefa que abrir um arquivo para escrita não precisa pegar os dados do arquivo do servidor e copiá-los no Módulo do Cliente. O sistema somente precisa manter no cache os dados que cada tarefa quiser escrever; só quando o arquivo for fechado é que os dados fornecidos pela tarefa são enviados ao servidor. Esta otimização também é possível na abordagem de permitir acesso simultâneos aos arquivos, embora para determinados casos não funcione (caso em que uma mesma tarefa abre o arquivo para leitura e escrita).

Não compartilhar arquivos para leitura e escrita simultaneamente também permite que os arquivos possam ser lidos nos caches dos Módulos dos Clientes por blocos. O sistema pode, uma vez que um arquivo é aberto para leitura, copiar somente alguns blocos do mesmo e não o arquivo completo (para um arquivo aberto para escrita, como ressaltado no parágrafo anterior, não é necessário copiar nenhum dado no cache do Módulo do Cliente), já que ele tem certeza de que os blocos que ficaram no servidor não serão modificados. Os blocos poderiam ser trazidos do servidor sob demanda, evitando copias desnecessárias de dados nos caches dos Módulos dos Clientes, possibilitando a redução do tamanho dos mesmos. Mesmo se no Módulo do Cliente não tiver disco nem memória para armazenar os arquivos, ele ainda pode usá-los diretamente do servidor, sem alterar a semântica adotada.

No entanto, usando esta abordagem limitamos as aplicações a um modelo específico de uso dos arquivos, tirando flexibilidade no uso do sistema de arquivos como um todo.

A Tabela 2.3 mostra a comparação das duas abordagens.

No SAD_GSA as aplicações somente podem abrir um arquivo para leitura ou para es-

	Leitura e escrita simultâneas	Leitura e escrita exclusivas
Vantagens	- Mais abrangente - Permite modelar qualquer interação da aplicação com o sistema de arquivos - Maior flexibilidade	- Só uma cópia de arquivo por computador - Os arquivos podem ser lidos por blocos - Os arquivos podem ser usados diretamente do servidor
Desvantagens	- Uma cópia do arquivo para cada tarefa - Necessita replicação dos arquivos completos no cache - Precisa de grandes capacidades no disco e/ou caches grandes - Os arquivos não podem ser usados diretamente	- Limitação nos modelos de interação da aplicação com o sistema de arquivos - Pouca flexibilidade

Tabela 2.3: Comparação das formas de utilização dos arquivos

crita (ver seção 5.2). A maior desvantagem desta abordagem é não permitir às aplicações compartilhar arquivos simultaneamente para leitura e escrita, o que pode limitar a forma de interagir destas aplicações com os arquivos que elas utilizam. No entanto, pelo fato destas aplicações serem aplicações paralelas, geralmente são usados outros modelos para compartilhar arquivos. A Figura 2.12 mostra o modelo que geralmente é usado por aplicações paralelas, e como estas aplicações poderiam usar os arquivos no sistema de arquivos proposto sem a necessidade de compartilhá-los simultaneamente para leitura e escrita.

De forma geral, uma aplicação paralela está formada por uma tarefa principal e um conjunto de tarefas filhas. A tarefa principal encarrega-se de dividir o trabalho em parcelas menores e entregá-las às tarefas filhas para elas processarem a informação. Ou seja, a primeira função da tarefa principal é fazer as inicializações necessárias para a aplicação, preparando o ambiente de execução. Uma das possíveis inicializações pode ser a relacionada com os arquivos que a aplicação vai utilizar. Nesta etapa serão criados os arquivos necessários, provavelmente fazendo algumas operações iniciais neles. Logo após este processo de inicialização, a tarefa principal cria um conjunto de tarefas filhas, que por sua vez também podem criar novas tarefas. Cada tarefa filha recebe a sua parcela de trabalho com os dados necessários para a sua execução, processa-os e retorna os resultados à tarefa principal. Estas tarefas, por exemplo, podem receber um conjunto de arquivos onde se encontram os dados necessários para o processamento. Os resultados podem ser armazenados em um ou vários arquivos, que podem ser criados por elas mesmas ou pela tarefa principal no processo de inicialização. É função também da tarefa principal receber todos os resultados obtidos pelas suas tarefas filhas e processá-los, obtendo o resultado final da aplicação.

O modelo apresentado permite definir uma forma genérica para que uma aplicação paralela use os recursos oferecidos pelo SAD_GSA sem a necessidade de compartilhar arquivos

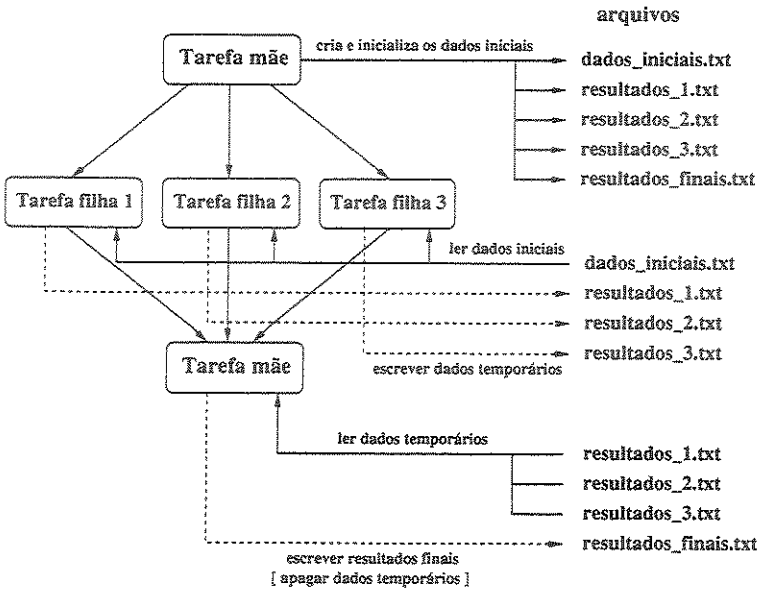

Figura 2.12: Modelo de uso dos arquivos pelas aplicações paralelas

para leitura e escrita simultaneamente. O modelo permite que qualquer arquivo possa ser lido ou modificado por diferentes tarefas; só não permite que ele seja utilizado para leitura e escrita ao mesmo tempo. Este modelo pode ser implementado facilmente por qualquer aplicação para obter acesso ao sistema de arquivos proposto.

Além disso, usando esta abordagem de não compartilhar arquivos simultaneamente para leitura e escrita não seria necessário implementar uma política de manutenção da coerência entre as diferentes réplicas dos arquivos espalhadas nos diferentes caches dos Módulos dos Clientes. O sistema somente tem que garantir que seja cumprida a semântica definida, ou seja, que uma vez que um arquivo é fechado, todas as mudanças feitas nele sejam visíveis em futuros acessos. Para obter este resultado, o sistema de arquivos proposto faz uso de escritas síncronas. Cada vez que o arquivo é fechado, todas as mudanças feitas no cache do Módulo do Cliente são enviadas ao servidor imediatamente. Assim, a operação de fechar o arquivo não retorna até que os dados sejam atualizados no servidor. Desta forma fica garantido que se uma outra tarefa abrir o arquivo logo após ele ser fechado, ela sempre obtém os dados atualizados. Para evitar que uma aplicação tenha que ficar fechando e abrindo o arquivo para que os dados sejam atualizados no servidor, o sistema implementa a operação *flush*. Esta operação também envia todas as mudanças feitas em um arquivo para o servidor sem a necessidade de fechar o arquivo.

2.11 Características do Sistema de Arquivos Proposto

A proposta apresentada satisfaz as principais características que um sistema de arquivos para um MPVC deve ter. As mesmas são discutidas abaixo.

Portabilidade: como o sistema visa explorar o poder computacional existente na Internet, especialmente através dos navegadores, é usada a linguagem Java para atenuar as dificuldades de se trabalhar com computadores heterogêneos. O próximo capítulo é dedicado à implementação e apresenta maiores detalhes deste aspecto.

Disponibilidade e tolerância a falhas: o sistema de arquivos oferece mecanismos de replicação de servidores para aumentar a disponibilidade e a tolerância a falhas do serviço oferecido, possibilitando que, caso um ou mais servidores de arquivos caiam ou percam sua conexão com os demais componentes do sistema, tenha-se disponível um ponto alternativo onde obter acesso aos dados. Foram definidos três níveis de replicação. O primeiro nível é formado pelos Cache nos Módulos dos Clientes, o segundo é formado pelos servidores que formam os Grupos de Servidores para Aplicações e o terceiro nível é formado pelo Servidor de Arquivos Central. Além disso, o sistema usa servidores do tipo *stateless*, mais tolerantes a falhas, já que não armazenam informações dos Módulos dos Clientes que estão fazendo uso de seus arquivos. Em caso de falha o processo de reiniciá-los é muito simples.

Consistência: o sistema usa, em cada Grupo de Servidores de Arquivos, um protocolo de atualização para modificar um arquivo, mantendo atualizadas as informações mantidas neles. Assim, qualquer um dos servidores em um grupo pode atender qualquer requisição feita pelas aplicações a ele associadas. Se houver problemas de coerência entre as diferentes cópias espalhadas no grupo, é iniciado um Processo de Reconciliação encarregado de corrigí-los.

Escalabilidade: o sistema de arquivos proposto é escalável devido à capacidade que seus grupos têm de se adaptar às necessidades das aplicações que estão sendo executadas e à quantidade de computadores do MPVC. A quantidade de servidores dentro de um grupo pode variar dinamicamente, aumentando caso os servidores de um determinado grupo não possam atender de forma eficiente todos os pedidos. Também é implementado um mecanismo de balanceamento de carga dos servidores totalmente distribuído. Além disso, o fato de se usar servidores *stateless* torna o sistema de arquivos proposto mais escalável.

Reconfiguração automática: o sistema de arquivos proposto também pode se reconfigurar automaticamente. O mecanismo de replicação implementado é totalmente transparente, isto é, caso um servidor fique fora do ar, o sistema se reconfigura automaticamente e os Módulos dos Clientes não ficam sabendo da falha. É implementado um mecanismo para detectar estas falhas e corrigi-las.

Eficiência: além da replicação de servidores e do mecanismo de balanceamento de carga implementado, que aumentam a eficiência do sistema, o conceito de cache é usado no Módulo do Cliente de forma intensiva. Usando cache no Módulo do Cliente diminui-se o tempo das operações de escrita e leitura, o tráfego na rede e a carga dos servidores. O sistema não permite que um mesmo arquivo seja utilizado para leitura e escrita simultaneamente. Embora esta abordagem não seja a mais flexível, não é necessária

a implementação de um protocolo de manutenção da coerência entre as diferentes réplicas mantidas nos caches dos Módulos dos Clientes e é fornecido um modelo que abrange de forma geral a utilização dos arquivos por aplicações paralelas. O sistema garante que a semântica utilizada (semântica de sessão) seja cumprida, usando escritas síncronas para enviar as mudanças feitas em um arquivo para o servidor uma vez que ele seja fechado.

Espaço de nomes: o sistema tem um espaço de nomes global e uniforme, oferecendo uma notação simples para se referir aos arquivos. Esta notação é transparente e independente de localização. O sistema define uma estrutura de diretórios que deve ser respeitada pelas aplicações que usam seus serviços. De forma geral, o sistema cria um diretório para cada usuário do sistema. Estes usuários devem criar um diretório para cada aplicação que eles criarem, armazenando nele os arquivos usados por essa aplicação. Esta hierarquia permite uma melhor organização da informação armazenada.

Segurança: o sistema de arquivos proposto usa o mecanismo conhecido como modo de bits de Unix para restringir o acesso aos arquivos. Os acessos são divididos em três categorias: acesso do usuário que criou o arquivo, acesso dos usuários que pertencem ao mesmo grupo do proprietário do arquivo e acesso dos demais usuários. Para cada categoria definem-se dois tipos de acesso: acesso de leitura e acesso de escrita. Para cada operação realizada o sistema analisa se o usuário tem a permissão requerida. Caso ele não tenha, a operação não é completada.

Instalação: o sistema de arquivos não necessita de instalação de software nem de manutenção por parte dos proprietários dos computadores que formarão parte do sistema. Estes proprietários simplesmente necessitam se conectar a um servidor Web e descarregar automaticamente uma pequena plataforma básica de software. Nessa plataforma estará incluído o componente necessário do sistema de arquivos para a comunicação com os servidores disponíveis e o uso dos serviços que eles oferecem.

2.12 Sumário

Nenhum dos sistemas de arquivos atuais satisfazem as principais exigências necessárias para poder ser usado de forma eficiente em um computador paralelo virtual baseado na Internet. Um sistema de arquivos adequado para sistemas deste tipo deve ser portável, permitindo seu uso em arquiteturas heterogêneas. A disponibilidade dos seus componentes deve ser alta para as aplicações, aumentando sua tolerância ante possíveis falhas que possam acontecer e oferecendo mecanismos de reconfiguração automática totalmente transparentes. Também deve ser escalável, evitando que surjam pontos de gargalo no sistema quando o número de tarefas aumente. Além disso, o sistema deve minimizar sempre que possível as comunicações pela rede mediante o uso de cache de forma intensiva nos clientes, aumentando assim a eficiência do mesmo. No entanto, a manutenção da consistência entre todas as réplicas espalhadas no sistema deve estar garantida.

Além destas propriedades principais, um sistema adequado para ser usado na Internet deve oferecer um espaço de nomes global e uniforme para facilitar a denominação dos arquivos, garantir a segurança dos dados e não requerer a instalação e manutenção de softwares específicos por parte dos usuários destes computadores.

Foram apresentadas três propostas preliminares e uma comparação inicial das mesmas foi feita para se fazer a melhor escolha do sistema de arquivos desejado. A primeira proposta foi SAC, cuja principal vantagem é a sua simplicidade e a manutenção de uma coerência estrita dos dados. No entanto, a sua disponibilidade é baixa, não é tolerante a falhas e não é escalável.

Além desta, foram apresentadas duas outras propostas com abordagens distribuídas. A primeira proposta é o SAD_GC. Esta abordagem é escalável, a disponibilidade da informação é maior, tornando-a mais tolerante a falhas. No entanto, devido ao grande número de servidores onde um mesmo arquivo (ou grupo de arquivos) pode ser armazenado, a manutenção da consistência das diferentes réplicas requer um esforço muito grande.

A última proposta foi o SAD_GSA, que visou limitar o número de servidores para cada arquivo, principal desvantagem da proposta anterior. Esta abordagem é escalável, devido à possibilidade dos seus grupos se adaptarem às necessidades das aplicações que estão sendo executadas e à quantidade de computadores do MPVC. Além disso, a disponibilidade da informação é maior, fazendo desta proposta a mais tolerante a falhas. No entanto, ela é a mais complexa de todas, e em princípio não prevê aplicações que compartilhem arquivos durante a execução. Entretanto, ela parece ser a mais adequada para um sistema de arquivos para sistemas de processamento maciçamente paralelo baseado na Internet. Esta afirmação se baseia na alta disponibilidade, tolerância a falhas e escalabilidade desta abordagem. Além disso, esta proposta atenua o principal problema da segunda proposta, ou seja, a manutenção da consistência entre as diferentes réplicas dos arquivos espalhadas no sistema.

Os principais componentes desta proposta são citados a seguir. O primeiro componente é o Servidor de Arquivos Central, que armazena uma cópia de todos os arquivos usados no computador virtual. Além disso, temos os Servidores de Arquivos para Aplicações, que replicam as informações armazenadas no Servidor de Arquivos Central e atendem os pedidos das tarefas das aplicações executadas no MPVC. Estes servidores formam os Grupos de Servidores para Aplicações, cada um dos quais atende uma determinada aplicação ou grupo de aplicações. O Resolvedor de Endereços é outro dos componentes principais do sistema, encarregado de manter os endereços de todos os servidores disponíveis, assim como as informação sobre suas características e carga de trabalho. Por último, o Módulo do Cliente é o componente descarregado nos computadores participantes no MPVC que permite a interação entre as aplicações e os servidores de arquivos.

Esta proposta satisfaz as principais características esperadas para um sistema de arquivos para plataformas de processamento paralelo baseadas na Internet. Ela é portável, tolerante a falhas e escalável. Além disso, tem a possibilidade de se reconfigurar de forma automática ante possíveis falhas, mantendo a eficiência do sistema. Não são necessários esforços de instalação ou manutenção por parte dos proprietários dos computadores que participam do processamento.

O próximo capítulo apresenta uma implementação do SAD_GSA usando JoiN como plataforma de teste. JoiN é um sistema para o processamento maciçamente paralelo baseado na Internet que está sendo desenvolvido no Departamento de Engenharia de Computação e Automação Industrial da Faculdade de Engenharia Elétrica e de Computação da UNICAMP. Esta plataforma é baseada em troca de mensagens e implementa todos os mecanismos necessários para a construção de um computador virtual.

Capítulo 3

Implementação do Sistema de Arquivos

Para testar e comprovar a eficácia das idéias propostas neste trabalho foi feita uma implementação do sistema de arquivos sobre a plataforma JoiN, um sistema para o processamento maciçamente paralelo na Internet que está sendo desenvolvido no Departamento de Engenharia de Computação e Automação Industrial da Faculdade de Engenharia Elétrica e Computação da UNICAMP [YH98b, YH98a].

3.1 JoiN, Sistema para o Processamento Maciçamente Paralelo

JoiN é um sistema que implementa um computador paralelo virtual usando os computadores conectados à Internet. O modelo básico de JoiN é similar ao modelo usado como referência neste trabalho e, portanto, baseia-se na ampla disponibilidade de navegadores para se ter acesso à Internet, suas facilidades de executar *applets Java* e na proliferação de Java como uma linguagem de programação independente de plataforma.

3.1.1 Objetivos de JoiN

- Usar de forma transparente a ampla diversidade de computadores heterogêneos conectados pela Internet.
- Permitir que qualquer computador conectado à Internet, mesmo os computadores pessoais, possa participar do computador paralelo virtual.
- Oferecer uma interface simples de utilização dos recursos.
- Ser escalável, permitindo a formação de um computador paralelo virtual com um grande número de computadores individuais.
- Ser tolerante a falhas, permitindo a reconfiguração automática quando um computador sair ou entrar no sistema.

- Não exigir esforços de instalação e manutenção dos proprietários que colocam seus computadores no sistema.
- Permitir a comunicação livre e segura entre os diferentes computadores que formam o MPVC.
- Recompensar os proprietários que disponibilizam seus computadores para serem usados no processamento paralelo e garantir a segurança dos dados nos mesmos.

Para tornar mais claros os objetivos e a natureza da plataforma usada como referência para o MPVC, é necessário ressaltar que a mesma não tem o propósito de ser usada para aplicações que necessitem de muita comunicação entre suas tarefas, devido à baixa velocidade média de transmissão na Internet. Também não é objetivo desta plataforma ser adequada para aplicações com poucas tarefas, já que seu ponto forte é a utilização do poder de cômputo de uma grande quantidade de computadores conectados pela Internet, poder que seria desperdiçado por aplicações deste tipo. Seu funcionamento se baseia na quantidade de computadores que pode agrupar e não na qualidade dos mesmos nem quão rápido eles podem se comunicar.

3.1.2 Arquitetura Básica

De forma geral, a arquitetura de JoiN é muito similar à mostrada na Figura 1.1. No entanto, ele apresenta algumas pequenas modificações para permitir uma interação mais estreita entre os diferentes componentes do computador virtual. Estas mudanças são mostradas na Figura 3.1.

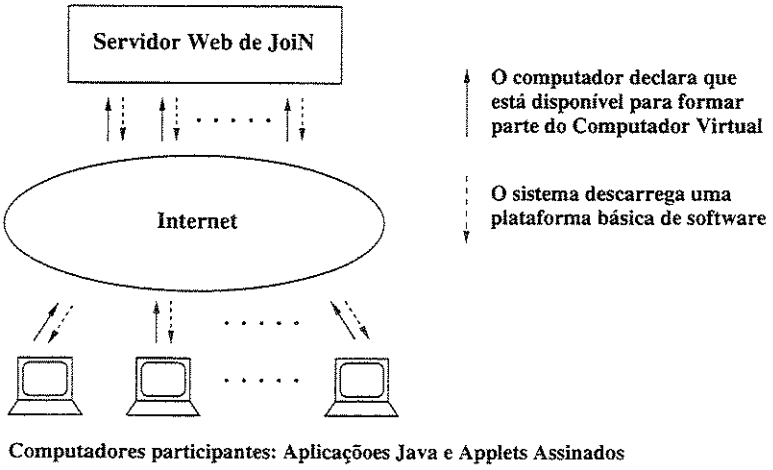

Figura 3.1: Arquitetura básica do computador virtual em JoiN

Ele é um sistema totalmente implementado em Java, garantido assim sua portabilidade. Como na arquitetura usada como referência, em JoiN tem-se também a existência de um Servidor Web que serve como ponto de entrada do sistema para os computadores da

Internet. Uma pessoa que quiser participar do sistema somente precisa de um navegador para estabelecer uma conexão com este servidor e declarar a sua disposição de colocar o seu computador para formar parte do computador paralelo virtual. Uma vez estabelecida a comunicação, uma plataforma básica de software é descarregada no computador em forma de *applet*, a qual permitirá sua integração ao sistema. Além desta plataforma, também é descarregado o endereço de um outro componente de JOIN, o Servidor para Aplicações, encarregado da distribuição das aplicações que serão executadas. O novo computador que está entrando no sistema comunica-se com este servidor para informar que está pronto para tomar parte no processamento.

Além desta forma de se unir ao JOIN usando navegadores, existe outra, destinada a usuários mais avançados. Assume-se que estes usuários podem instalar uma aplicação no seu computador que permita a interação direta com o sistema sem necessidade de a utilização de navegadores para WWW. A plataforma básica que é descarregada em forma de *applet* nos computadores participantes está também disponível em forma de aplicação Java. Desta forma, usuários que coloquem seus computadores à disposição do sistema com freqüência poderão melhorar os resultados de sua participação instalando esta aplicação em seu(s) computador(es).

Na Figura 1.1 mostra-se de forma geral este processo de integração de um novo computador ao JOIN.

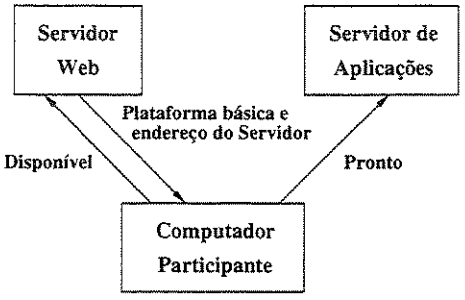

Figura 3.2: Novo computador entrando no JOIN

Para conseguir administrar os vários computadores que fazem parte do computador paralelo virtual, JOIN divide-os em grupos. Cada grupo tem um ou mais Coordenadores, como é mostrado na Figura 3.3. Eles são responsáveis pela organização de todos os computadores de seu grupo e pela comunicação com outros Coordenadores e com qualquer um dos computadores de outros grupos. Os Coordenadores conhecem o endereço de todos os computadores de seu grupo. Os outros computadores do grupo, chamados de Trabalhadores, conhecem o endereço de seus Coordenadores e são os encarregados de executar as aplicações paralelas submetidas ao sistema. Um computador não pode pertencer a mais de um grupo. Os grupos são conectados logicamente seguindo uma topologia de hipercubo binário, que recebe o nome de Hipercubo Dinâmico Virtual [HH99].

JOIN oferece uma interface gráfica que pode ser executada em qualquer computador para permitir aos seus clientes interagirem com o sistema. Esta interface permite a instalação, remoção e submissão de aplicações, e a obtenção de informações atualizadas sobre o

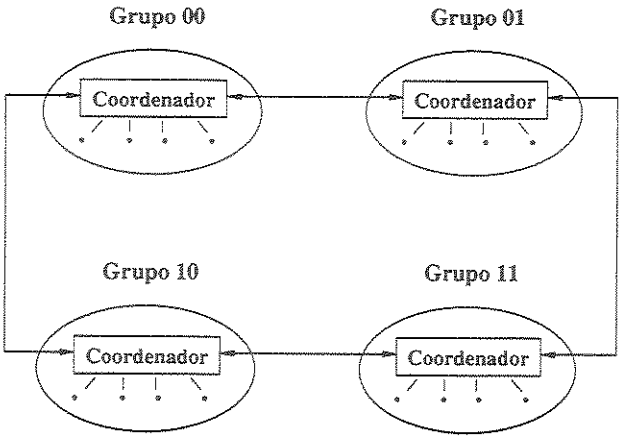

Figura 3.3: Grupos em JoiN

comportamento do sistema, dentre outras funções.

Tanto os componentes do sistema como as aplicações estão formados por tarefas, que são divididas em duas categorias principais: tarefas de sistema e tarefas de aplicações. Com esta abordagem, obtém-se maior simplicidade e uniformidade na implementação e na manipulação do sistema, já que são usados os mesmos mecanismos para criá-las, executá-las, terminá-las, assim como para prover comunicação entre as mesmas. Cada tarefa em JoiN herda da classe *Thread* de Java, ou seja, tem sua própria linha de execução.

A plataforma básica no JoiN é baseada em troca de mensagens. Esta plataforma fornece as facilidades mínimas para a comunicação entre tarefas e para a execução de aplicações paralelas no computador virtual. Neste paradigma, as tarefas comunicam-se entre si mediante o envio e recepção de mensagens. Para isto, cada tarefa tem um identificador, chamado de TID (*Task ID*), único para cada tarefa. As tarefas do sistema tem TID negativo e as tarefas das aplicações, TID positivo. Uma tarefa de sistema pode criar diretamente tantas tarefas quanto quiser. Uma aplicação paralela em JoiN é formada por uma tarefa inicial, que também pode criar novas tarefas (filhas) durante sua execução. A tarefa mãe conhece o TID de todas suas tarefas filhas e cada tarefa filha conhece o TID da tarefa que a criou.

Na Figura 3.4 tem-se mais detalhes de como é realizada esta comunicação.

3.2 Kernel e Serviços em JoiN

Para aumentar a flexibilidade e a robustez do sistema, JoiN está dividido em duas partes principais, o *kernel* e os serviços. O *kernel* suporta as funcionalidades mínimas do sistema, como o gerenciamento de mensagens, a resolução de TID, as comunicações básicas entre tarefas, a administração de tarefas, dentre outras funções. Os serviços são componentes que não são imprescindíveis para o funcionamento do sistema e cujo objetivo é permitir a extensão da funcionalidade do sistema sem alterar o *kernel* básico. Estes serviços podem ser substituídos por outros, permitindo testar várias soluções para o mesmo problema,

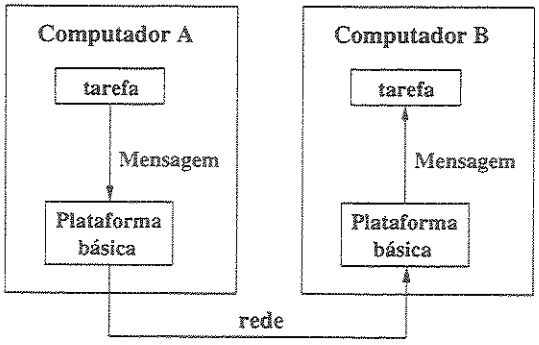

Figura 3.4: Plataforma baseada em troca de mensagens.

assim como adicionar e remover funcionalidade do sistema sem alterar suas características principais e sem exigir grandes esforços de implementação.

3.2.1 Implementação dos Serviços

Cada serviço é implementado como um conjunto de componentes que são executados em computadores diferentes. Quem quiser implementar um serviço tem que lidar com cinco tipos diferentes de componentes, cada um com suas características particulares. No entanto, não é necessário implementar todos os componentes. Este número depende da complexidade e objetivos do serviço que se deseja implementar. A seguir são mostrados os componentes que formam um serviço.

Componente do Servidor: componente executado quando é iniciado o Servidor Web de JOIN.

Componente do Coordenador: componente executado toda vez que um Coordenador é iniciado.

Componente dos Trabalhadores: componente executado toda vez que um novo trabalhador é incorporado ao sistema.

Componente Gráfico: componente associado à interface do JOIN, e cujo objetivo é permitir que o serviço ofereça uma interface gráfica ao usuário.

Componente das Aplicações: componente executado no computador onde uma aplicação é submetida.

A forma como os componentes de um serviço podem interagir entre si está definida pelo sistema. Cada componente conhece o endereço do Componente do Coordenador, que por sua vez conhece o endereço do Componente do Servidor, como mostra a Figura 3.5.

Todos estes componentes dos serviços são um tipo particular de tarefa de sistema em JOIN, chamados de Componentes de Serviço (*Service Components*). Um Componente de Serviço pode criar novos Componentes de Serviço.

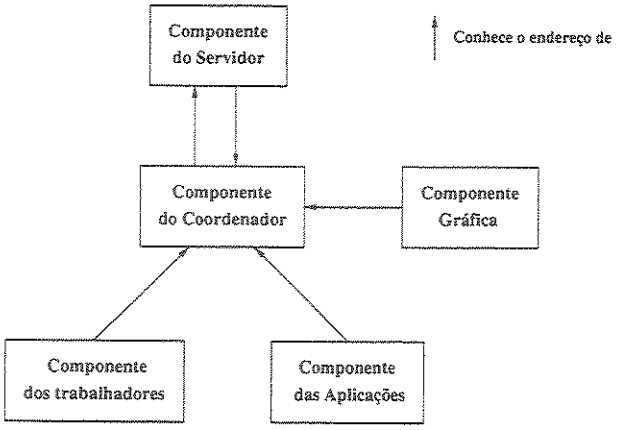

Figura 3.5: Componentes que formam um serviço em JoiN e a interação entre eles

3.3 O SAD_GSA como um Serviço em JoiN

Um dos primeiros serviços implementados para o JoiN foi o sistema de arquivos. Este serviço usa as especificações expostas neste trabalho e também é totalmente implementado em Java, o que o torna portável.

A implementação do sistema de arquivos fez uso dos cinco componentes que integram um serviço em JoiN. Na Tabela 3.1 mostra-se a relação entre os diferentes componentes do sistema de arquivos apresentado no capítulo anterior e os componentes que formam um serviço em JoiN. Maiores informações sobre os componentes do SAD_GSA, suas características e funções podem ser obtidas no Capítulo 2.

Componentes de Serviços	Componente do Sistema de Arquivos
Componente do Servidor	Servidor de Arquivos Central
Componente do Coordenador	Resolvedor de Endereços
Componente dos Trabalhadores	Módulo do Cliente
Componente das Aplicações	Replicador de Diretórios
Componente Gráfico	Servidor de Arquivos de Aplicações

Tabela 3.1: Componentes de Serviços de JoiN e componentes do SAD_GSA

3.3.1 Componente do Servidor - Servidor de Arquivos Central

Os Componentes dos Servidores de todos os serviços instalados são executados quando o Servidor de JoiN é iniciado. Este servidor é um dos computadores mais importantes no sistema e imprescindível para a construção e funcionamento do computador virtual. Também deve ser um dos pontos mais visíveis e estáveis do sistema.

O Servidor de Arquivos Central é o componente do sistema de arquivos que armazena todos os arquivos usados no computador virtual, garantindo assim a disponibilidade de pelo

menos uma cópia atualizada de todos os arquivos para as aplicações que serão executadas. Sendo assim, é importante implementar o Servidor de Arquivos Central como o Componente do Servidor do serviço de arquivos de JOIN.

3.3.2 Componente do Coordenador - Resolvedor de Endereços

Os Componentes do Coordenador dos serviços instalados são executados em cada Coordenador do JOIN. Os Coordenadores são os responsáveis pela organização dos computadores no MPVC e pelas comunicações entre eles. Todos os demais componentes que implementam um serviço conhecem o endereço deste componente. Assim, ele atua como ponto de interações do serviço.

O Resolvedor de Endereços é o componente do sistema de arquivos encarregado de armazenar o endereço e as informações de todos os servidores de arquivos disponíveis no sistema. Este resolvedor é o primeiro ponto de contato do sistema de arquivos para fornecer estas informações aos Módulos dos Clientes descarregados nos computadores do MPVC.

Portanto, implementar o Resolvedor de Endereços como o Componente do Coordenador do serviço de arquivos de JOIN possibilita que qualquer outro componente do serviço possa se comunicar com ele, em particular os componentes que implementam os Servidores de Arquivos e os Módulos dos Clientes. Sendo assim, o Resolvedor de Endereço seria replicado da mesma forma como são replicados os Coordenadores em JOIN. O impacto desta replicação no sistema é muito baixo, já que o número de resolvedores é limitado e o volume de informação que estes componentes armazenam é pequeno. Note que os Resolvedores de Endereços no precisam conhecer todos os arquivos existentes no sistema, mas apenas os endereços e características dos servidores de arquivos onde eles estão armazenados.

3.3.3 Componente Gráfico - Servidores de Arquivos para Aplicações

O Componente Gráfico de um serviço é executado sempre que a Interface Gráfica de JOIN é iniciada. Esta interface serve para que os usuários de JOIN obtenham informações sobre o sistema, como o número de computadores que formam o MPVC, o número de tarefas no MPVC e por aplicação, o estado destas tarefas, dentre outras informações. Além disso, usando esta interface eles podem instalar e remover aplicações, submetê-las ao sistema, etc.

O sistema de arquivos cria Grupos de Servidores para Aplicações que atenderão diretamente os pedidos das aplicações que são executadas no computador virtual. Na seção 2.4.3 vimos como o sistema de arquivos cria um Grupos de Servidores para Aplicações para cada usuário que estiver interagindo com o sistema, e assim, todas as aplicações desse usuário executadas em JOIN usam os Servidores de Arquivos para Aplicações desse grupo.

Desta forma, a implementação destes Servidores de Arquivos para Aplicações como o Componente Gráfico de um serviço em JOIN garante que quando um usuário execute a Interface Gráfica para interagir com o sistema (provavelmente para submeter uma aplicação), um servidor do Grupo de Servidores para Aplicações seja executado nesse computador. À medida que aumente a carga deste servidor, novos Servidores de Arquivos para Aplicações poderão ser criados e incorporados ao grupo.

3.3.4 Componente das Aplicações - Replicador de Diretórios

Uma vez que um usuário execute a Interface Gráfica de JOIN, ele pode instalar aplicações e submetê-las ao sistema. Toda vez que uma aplicação é submetida, os Componentes das Aplicações dos serviços instalados em JOIN são executados.

O Replicador de Diretórios é um componente do sistema de arquivos que replica diretórios do Servidor de Arquivos Central para os Servidores de Arquivos para Aplicações. Estes diretórios contêm os arquivos usados pelas aplicações. Uma maneira eficiente de fazer esta replicação é sob demanda, isto é, somente replicar os dados estritamente necessários para a aplicação.

Portanto, implementando o Replicador de Diretórios como o Componente das Aplicações do serviço de arquivos de JOIN, podemos obter este resultado. Toda vez que uma aplicação é submetida, este replicador é executado no mesmo computador onde a Interface Gráfica foi iniciada, replicando somente o diretório que contém os arquivos usados pela mesma. Na seção 2.5.2 foi explicada a estrutura de diretório proposta para o sistema. Para cada aplicação implementada é criado um diretório onde são armazenados seus arquivos. O replicador somente precisa replicar este diretório, evitando criar cópias desnecessárias de arquivos.

3.3.5 Componente dos Trabalhadores - Módulos dos Clientes

Os Componentes dos Trabalhadores são os componentes dos serviços em JOIN que são executados em cada novo computador que entra no sistema e que será usado como Trabalhador, isto é, o computador onde as aplicações serão executadas. Estes componentes conhecem o endereço do Componente do Serviço que será executado no Coordenador (Componente do Coordenador).

Os Módulos dos Clientes são os componentes do sistema de arquivos descarregados nos computadores participantes no MPVC que permitem a interação entre as aplicações e os servidores de arquivos. As informações dos servidores disponíveis são obtidas no Resolvedor de Endereços e atualizadas freqüentemente com a ajuda dos Servidores de Arquivos para Aplicações.

Sendo assim, implementando os Módulos dos Clientes do sistema de arquivos como Componentes dos Trabalhadores do serviço, obtêm-se os resultados desejados. Desta forma, em todos os computadores que executarão tarefas das aplicações será instalado o componente do sistema de arquivos que possibilitará a interação com os servidores apropriados.

3.4 Metodologia

Neste trabalho foram usados os conceitos de orientação a objetos, tentando explorar ao máximo as vantagens que está metodologia oferece, tais como encapsulamento, polimorfismo e herança.

O SAD_GSA é uma aplicação 100% Java. Foram usados os recursos oferecidos pelo JDK 1.1.7, versão mais estável no momento da implementação. A interface oferecida pelo SAD_GSA é simples e procurou-se fazer com que a mesma fosse o mais próxima possível

da interface nativa de Java. Sendo assim, a forma como as aplicações usam os recursos oferecidos pelo sistema de arquivo proposto é muito similar àquela como Java manipula os arquivos.

A comunicação entre os diferentes componentes do sistema está baseada somente nas classes oferecidas por JOIN. Não foram usados outros recursos disponíveis, como por exemplo, RMI ou CORBA.

3.5 Detalhes de Implementação

A seguir são mostrados detalhes de como foram implementados os diferentes componentes e os principais mecanismos no sistema de arquivos.

Esta seção é dividida em cinco partes. As primeiras duas apresentam as classes que formam a base do sistema de arquivos. As restantes apresentam detalhes da implementação dos diferentes componentes do sistema de arquivos (Servidor de Arquivos Central, Servidores de Arquivos para Aplicações, Resolvedor de Endereços e Módulos dos Clientes). Estes componentes usam as classes básicas explicadas nas duas primeiras seções para implementar os diferentes componentes de um serviço na construção do sistema de arquivos.

3.5.1 Arquivos Regulares e Diretórios

Na seção 2.5 foram mostrados os tipos de arquivos usados no sistema. Foram definidos os Arquivos Regulares, encarregados de armazenar os dados, e Diretórios, encarregados da organização do sistema de Arquivos. Também foram definidos os atributos de cada tipo de arquivo em particular.

Para a manipulação destes arquivos e seus atributos foi implementada toda uma hierarquia de classes. Tem-se duas classes principais associadas a cada tipo de arquivos, uma para manipular os atributos do arquivo e outra para manipular o arquivo.

A classe implementada para manipular os atributos dos Arquivos Regulares (classe *RegularFileAttributes*) permite obter e modificar o dono do arquivo, o tamanho em bytes, a data em que o arquivo foi criado, acessado e modificado pela última vez, as permissões de acesso dos diferentes usuários (leitura e escrita) e o estado do arquivo (fechado, aberto para leitura ou aberto para escrita). A classe implementada para manipular um Arquivo Regular (classe *RegularFile*) permite a criação de novos Arquivos Regulares e a abertura de arquivos já existentes, além de leitura e escrita de dados.

Similar à classe implementada para manipular os atributos dos Arquivos Regulares, a classe para manipular os atributos dos Diretórios (classe *DirectoryAttributes*) permite obter e modificar o dono do diretório, a quantidade de arquivos que ele contém, quando o diretório foi criado, acessado e modificado pela última vez, as permissões de acesso dos diferentes usuários e o nome de todos os arquivos e diretórios contidos neles. Para manipular um Diretório foi implementada uma classe (classe *Directory*) que permite criar novos Diretórios ou manipular Diretórios já existentes, além de oferecer recursos para criar e apagar Diretórios e Arquivos Regulares no diretório gerenciado e listar o seu conteúdo.

Este conjunto de classes é usado pelos servidores de arquivos para manipular toda a estrutura de arquivos gerada pelo sistema e que é oferecida às aplicações. No entanto, existe outro grupo de classes que possibilitam a manipulação dos arquivos que são criados e mantidos no cache dos Módulos dos Clientes. Como foi explicado na seção 2.10.3, o sistema faz uso intensivo de cache nos Módulos dos Clientes, tornando possível a criação de arquivos locais que serão usados na resolução das operações de entrada/saída das aplicações, diminuindo as comunicações através da rede e aumentando a eficiência do sistema. Este cache é implementado e gerenciado usando-se fundamentalmente duas classes principais para a manipulação destes arquivos. Uma delas permite a criação e manipulação de arquivos que são abertos para leitura (classe *ReadCacheFile*) e a outra permite a criação e manipulação de arquivos abertos para escrita (classe *WriteCacheFile*). Mais a frente veremos como este cache é implementado e a forma como estas classes são usadas.

A Figura 3.6 mostra a hierarquia de classes implementada para a manipulação dos arquivos e diretórios. Todas estas classes formam o núcleo do sistema de arquivo.

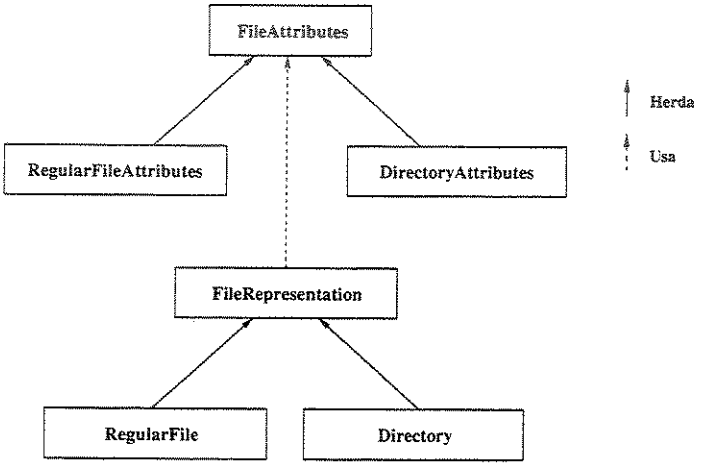

Figura 3.6: Hierarquia de classes para a manipulação de Arquivos Regular e Diretórios

3.5.2 Comandos e Utilitários

Comandos

O sistema de arquivos oferece, além da interface oferecida às aplicações, vários comandos para interagir diretamente com o sistema. Estes comandos são aplicações Java que permitem aos usuários do sistema fazer diferentes operações, como criar novos arquivos ou diretórios, importar e exportar arquivos, criar novos usuários, dentre outras. Todas estas operações são realizadas diretamente no Servidor de Arquivos Central. A seguir são resumidos os principais comandos implementados.

CreateFileSystem: cria o sistema de diretórios inicial. São criados os diretórios *system* e *users*, assim como os arquivos de configuração do sistema. Somente os usuários

pertencentes ao grupo ADMIN podem criar esta estrutura inicial de diretórios (seção 2.5.1).

NewUser: criar um novo usuário no sistema. Para cada novo usuário criado é definido um nome, uma senha e um grupo ao qual pertence. Além de criar o usuário, o Diretório do Usuário (*Home Directory*, definido na seção 2.5.2) também é criado. Somente os usuários pertencentes ao grupo ADMIN podem criar novos usuários.

NewDirectory: cria um novo Diretório no sistema de arquivos. O usuário que executa este comando precisa ter acesso de escrita no diretório onde o novo diretório será criado.

DeleteDirectory: apaga um Diretório do sistema de arquivos. O usuário que executa este comando precisa ter acesso de escrita no diretório ao qual pertence o diretório que será apagado. Se o diretório ainda contiver arquivos, a operação não será executada.

NewFile: cria um novo Arquivo Regular no sistema de arquivos. O usuário que executa este comando precisa ter acesso de escrita no diretório onde o arquivo é criado.

DeleteFile: apaga um Arquivo Regular do sistema de arquivos. O usuário que executa este comando precisa ter acesso de escrita no diretório ao qual o arquivo pertence.

CopyFile: copia um Arquivo Regular de um diretório do sistema para outro. O usuário que executa este comando precisa ter acesso de leitura no diretório ao qual o arquivo pertence e acesso de escrita no diretório para onde o arquivo será copiado.

ImportFile: importa um arquivo de um outro sistema de arquivos (por exemplo Unix ou Windows) para o novo sistema de arquivos. O usuário que execute este comando precisa ter acesso de escrita no diretório para onde o arquivo será copiado.

ExportFile: exporta um arquivo do novo sistema de arquivo para um outro sistema de arquivos (por exemplo Unix ou Windows). O usuário que execute este comando precisa ter acesso de leitura no diretório ao qual o arquivo pertence.

Todos estes comandos são implementados como aplicações Java e permitem aos usuários do sistema preparar as condições iniciais antes de executar uma aplicação.

Utilitários

Além destes comandos, o sistema de arquivos implementa uma série de Utilitários, que podem ser classificados em três grandes grupos, mostrados a seguir.

Utilitários de redes: estas classes manipulam as informações do sistema que são compartilhadas pelos diferentes componentes (servidores de arquivos, Resolvedor de Endereços, Módulos dos Clientes, tarefas das aplicações) e enviadas pela rede em forma de mensagens. Dentre estas informações estão as informações dos servidores, as informações dos pedidos de entrada/saída feitos pelas aplicações e as informações das

respostas enviadas pelos servidores. Todas estas classes implementam a interface *Serializable* de Java, o que permite maior facilidade de envio das mesmas pela rede.

Utilitários de compressão e descompressão: estas classes permitem a compressão e descompressão tanto de informações isoladas como de arquivos e diretórios do sistema de arquivos. Estas classes são usadas para a compressão e descompressão das informações que são enviadas pela rede. No Capítulo 4 são mostrados os testes realizados para determinar o tamanho de mensagem até o qual não se usa compressão.

Utilitários gerais do sistema de arquivos: estas classes implementam funcionalidades usadas pelos diferentes componentes do sistema de arquivos na implementação dos mecanismos por ele oferecidos. Dentre estas funcionalidades estão a criação e manipulação das tabelas de arquivos abertos, usadas pelos servidores de arquivos e pelos Módulos dos Clientes na implementação do cache. Estas tabelas podem ser de tamanho fixo ou variável. Também está presente a implementação dos relógios lógicos usados para sincronizar os diferentes servidores de arquivos em um Grupo de Servidores para Aplicações no mecanismo de replicação (ver seção 2.7.3), dentre outras funcionalidades.

3.5.3 Servidores de Arquivos

Tanto o Servidor de Arquivos Central como os Servidores de Arquivos para Aplicações foram implementados usando vários Componentes de Serviços de JOIN. Esta abordagem permite que sejam usadas várias linhas de execução para se manipular os arquivos (servidores *multithreads*), aumentando assim a eficiência dos mesmos e diminuindo o tempo de resposta aos pedidos. Cada servidor está formado por um componente principal, chamados de Servidor de Arquivos (classes *MainFileServer* e *AppFileServer*), e vários componentes auxiliares, chamados de Gerentes de Arquivos (*MainFileRequestManager* e *AppFileRequestManager*).

Na seção 2.6 foi definido o tipo de servidor que é usado no sistema de arquivos. Os servidores *stateless* se mostraram mais adequados para o sistema proposto, devido a sua maior escalabilidade e tolerância a falhas. No entanto, eles são menos eficientes que os servidores do tipo *stateful*. Para melhorar seu desempenho, cada servidor de arquivos cria uma tabela de arquivos abertos. Cada arquivo referenciado pelas aplicações é aberto e colocado nessa tabela para ser acessado mais rapidamente em futuras requisições, evitando ter que procurá-lo no disco. Para cada arquivo aberto é criado um Gerente de Arquivos, encarregado de manipular todas as requisições feitas ao mesmo. Cada entrada na tabela armazena o nome do arquivo que está sendo referenciado, o TID do Gerente de Arquivos que o manipula (tanto o Servidor de Arquivos como os gerentes são Componentes de Serviços e, como outras tarefas de sistema em JOIN, têm associado um TID único) e um campo numérico que indica a última referência feita ao arquivo.

O tamanho desta tabela é fixo, o qual é definido como um parâmetro de configuração do sistema de arquivos. É usada uma política LRU (*Least Recently Used*) para atualizá-la. Fixando o número de arquivos abertos evita-se que a tabela aumente muito de tamanho, o que tornaria difícil sua manipulação. Além disso, o número de gerentes criados também é controlado, impedindo que a manipulação dos mesmos diminua a eficiência do sistema.

Quando o servidor de arquivos é iniciado, ele executa um conjunto de operações de inicialização, tais como carregar os parâmetros do sistema, montar o sistema de arquivos, criar a tabela de arquivos abertos e se registrar no Resolvedor de Endereços, dentre outras funções. Logo após este processo inicial, o servidor de arquivos fica aguardando os pedidos de entrada e saída das aplicações, que interagem diretamente com este servidor. Para cada requisição feita a um arquivo, o servidor executa as operações mostradas a seguir.

- Procura o arquivo referenciado na tabela.
- Se ele for achado, o pedido é enviado ao Gerente de Arquivos associado ao arquivo para ser processado.
- Se o arquivo não for achado na tabela, o próximo passo do servidor depende do estado da tabela.
- Se a tabela estiver cheia, o servidor precisa fechar um dos arquivos abertos (usando a política LRU, que utiliza a última referência feita ao arquivo) e colocar na posição desocupada na tabela o novo arquivo referenciado. Neste caso, não é preciso criar um novo Gerente de Arquivos, mas simplesmente informar ao gerente associado ao arquivo fechado o nome do novo arquivo que ele vai manipular.
- Se a tabela ainda não estiver cheia, o servidor de arquivos cria um novo Gerente de Arquivos, usando como parâmetro de criação o nome do arquivo referenciado.
- Uma vez determinado qual gerente foi associado ao arquivo referenciado, o pedido é enviado a ele para ser processado.
- O Gerente do Arquivo recebe o pedido e o processa.
- O resultado da operação é enviada diretamente ao Módulo do Cliente pelo gerente.

As possíveis operações a serem realizadas em um arquivo podem ser resumidas em três tipos principais.

Operações sobre atributos: neste grupo tem-se operações para determinar, dado o nome de um arquivo, se ele representa um Arquivo Regular ou um Diretório, e operações para obter ou modificar os atributos de um determinado arquivo, tais como o dono do arquivo, o tamanho, as data de criação, último acesso e última modificação, assim como saber se um determinado usuário pode ou não ler ou modificar o arquivo.

Operações sobre Diretórios: estas são operações realizadas somente sobre Diretórios. Elas agrupam operações para determinar se um determinado arquivo existe, para criar e apagar Arquivos Regulares e Diretórios, assim como listar o conteúdo destes últimos.

Operações sobre Arquivos Regulares: estas são operações realizadas somente sobre Arquivos Regulares. Neste grupo tem-se operações para abrir um arquivo para leitura ou escrita, ler ou escrever dados nele e fechar o arquivo.

3.5.4 Grupos de Servidores para Aplicações

O sistema de arquivos forma Grupos de Servidores para Aplicações que atenderão diretamente os pedidos das aplicações a eles associadas. Como visto na seção 3.3, para cada usuário que execute uma aplicação em JOIN será criado um Grupo de Servidores para Aplicações que atenderá todos os pedidos de entrada/saída dessa aplicação e de outras aplicações que esse usuário possa vir a executar.

Todos os arquivos usados por uma aplicação são armazenados em cada um dos servidores do grupo. Sendo assim, é necessária a implementação de um mecanismo para a manutenção da coerência entre as cópias espalhadas no grupo. Este mecanismo de replicação foi explicado em detalhes na seção 2.7.3, onde foi vista a necessidade de uma certa sincronização entre os servidores que formam um grupo, a qual permita determinar, entre dois eventos, qual ocorreu primeiro. Foi usado o algoritmo de Lamport para obter esta sincronização dos relógios dos servidores do grupo. Este algoritmo faz uso de relógios lógicos, ao invés dos relógios físicos dos computadores. A seguir será explicado como estes relógios lógicos foram implementados. A classe implementada (classe *Timer*) é um dos utilitários oferecidos pelo sistema (seção 3.5.2).

Relógio Lógico

Um dos componentes básicos de um computador é o seu relógio interno. Geralmente este relógio é um contador implementado com um cristal de quartzo, que oscila a uma freqüência bem conhecida incrementando o contador continuamente a intervalos regulares. Os sistemas operacionais utilizam o relógio interno dos computadores para implementar o que chamaremos de relógios físicos, usados pelas diferentes aplicações para obter o tempo real. Estes relógios não podem ser modificados pelas aplicações, mas somente acessados para obter a hora local.

Devido à dificuldade de sincronizar com precisão os relógios físicos de vários computadores que formam um sistema distribuído, geralmente estes sistemas usam relógios lógicos. Na implementação de um relógio lógico, é necessário levar em consideração dois pontos fundamentais.

Atualização constante: os relógios lógicos devem, da mesma forma como os relógios físicos, se atualizar periodicamente, isto é, acompanhar o transcurso do tempo.

Incrementos discretos: relógios lógicos, diferentemente dos relógios físicos, podem ser modificados em valores discretos e sua hora pode ser alterada por uma aplicação. No entanto, esta modificação só é possível para adiantar a hora do relógio, nunca para atrasá-la.

O sistema de arquivos implementa um relógio lógico que é usado pelos Servidores de Arquivos para Aplicações em um grupo para se sincronizarem. A Figura 3.7 mostra como este relógio lógico é implementado e usado pelos servidores.

Como foi explicado na seção 2.7.3, os Servidores de Arquivos para Aplicações comunicam-se com o Servidor de Arquivos Central para replicar os arquivos que serão usados pelas

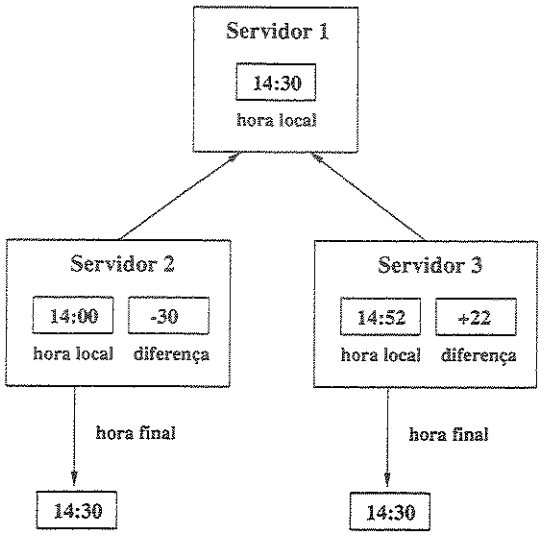

Figura 3.7: Implementação de um relógio lógico

aplicações associadas ao Grupo de Servidores para Aplicações ao qual pertencem. Nessa comunicação, além dos arquivos a replicar, o Servidor de Arquivos Central envia a sua hora local. Assim que um Servidor de Arquivos de Aplicações recebe esta hora, cria o seu relógio lógico, que armazena a diferença entre sua hora local e a hora recebida do outro servidor. Quando alguma tarefa solicita a hora ao servidor, ele retorna a hora local do computador (obtida usando o método *getCurrentTimeMillis* da classe *System* de Java), corrigida segundo a diferença calculada.

Tomando como exemplo a figura anterior, a hora local do Servidor de Arquivos Central (que chamaremos de S_1) é 14:30. As horas locais dos Servidores de Arquivos para Aplicações (que chamaremos de S_2 e S_3) no grupo, quando recebem a mensagem de S_1, são 14:00 e 14:52, respectivamente. Com a hora que vem de S_1 e suas horas locais, S_2 e S_3 criam seus relógios lógicos, armazenando a diferença entre as duas horas, neste caso, -30 para S_2 e +22 para S_3. Se uma tarefa sendo executada em S_2 ou S_3, como por exemplo o Gerente de Arquivos apresentados na seção 3.5.3, pede a hora para o seu servidor respectivo, cada um retorna sua hora local corrigida pela diferença calculada. Ou seja, para S_2 a hora seria $t_2 = (14{:}00 - (-30)) = 14{:}30$, e para S_3 seria $t_3 = (14{:}52 - (+22)) = 14{:}30$, a mesma hora de S_1.

No entanto, como os relógios lógicos usam os relógios físicos dos seus respectivos computadores, as horas dos servidores do grupo provavelmente são atualizadas de forma diferente. É aplicado então o Algoritmo de Lamport para sincronizar os relógios de maneira que cada servidor possa, por exemplo, determinar entre um pedido local e um pedido que veio de um outro servidor do grupo, qual ocorreu primeiro (seção 2.7.3). Este algoritmo altera a hora dos servidores para mantê-los sincronizados, e o relógio lógico implementado oferece a funcionalidade necessária para isto mediante o uso de duas funções, uma para incrementar a hora do relógio local em um determinado valor e outra para alterá-la para uma nova hora (este novo valor não pode ser menor que a hora atual do relógio lógico, ou seja, não se pode

atrasá-lo).

A funcionalidade da classe que implementa o relógio lógico no sistema de arquivos pode ser resumida como a seguir.

Construtor: recebe a hora padrão e calcula a diferença com a hora local. Esta diferença é usada em futuros pedidos das aplicações da hora do relógio.

Pedido de hora: devolve a hora às aplicações. A hora local é corrigida com a diferença calculada.

Incremento de hora: recebe um valor que será somado à hora do relógio, adiantando-o.

Alteração de hora: recebe o novo valor para o qual a hora vai ser modificada. Este valor sempre é maior que a hora atual do relógio lógico.

3.5.5 Módulo do Cliente

Na seção 3.1.2 foram apresentadas as duas formas possíveis de participar de JOIN. Um proprietário de um computador interessado em participar no computador virtual pode usar um navegador para estabelecer uma conexão com o servidor Web do JOIN, descarregando a plataforma básica que possibilitará sua integração ao sistema. A outra forma de participar do processamento paralelo é que o proprietário instale uma aplicação no seu computador que permita uma interação direta com o sistema sem necessidade da máquina virtual Java dos navegadores.

A primeira das abordagens usa *applets Java* para se comunicar com os restantes integrantes do computador, enquanto a segunda usa aplicações Java. Existem várias diferenças entre o uso de *applets* ou aplicações Java, mas a mais importante no caso de JOIN são as restrições de acesso. Uma aplicação Java tem total acesso aos recursos do computador onde é executada. Já os *applets Java* têm restrições de acesso impostas pelos navegadores [Sun97c]. Geralmente um *applet* não pode realizar várias tarefas permitidas às aplicações. A seguir são mostradas as mais importantes.

- Abrir conexões com outros computadores que não seja aquele do qual foi descarregado.
- Ler ou escrever no sistema de arquivos local do computador.

No entanto, atualmente existem várias propostas para relaxar as restrições de segurança inicialmente impostas aos *applets Java* pelos navegadores [Net98a]. Um *applet* pode ser assinado usando certificados digitais. Quando um usuário descarrega um *applet* assinado, o seu navegador lhe informará sobre essa situação, indicando quem é o assinante do mesmo. Dependendo da confiança que o usuário tenha na pessoa ou instituição que assinou o *applet*, ele poderá permitir ações proibidas para *applets* comuns. Muitos dos principais navegadores atuais já oferecem este tipo de recursos, como por exemplo Netscape e Internet Explorer (a partir das versões 4.x) [Net98b].

JOIN permite a comunicação livre entre *applets* mediante a utilização dos mecanismos de assinaturas digitais para permitir aos *applets* descarregados nos computadores ações que

normalmente seriam proibidas para eles. Um *applet* assinado digitalmente pode solicitar ao proprietário do computador, por exemplo, permissão para abrir e aceitar conexões com computadores diferentes do servidor WWW, ou para escrever em arquivos no seu disco local.

Sendo assim, os computadores que participam do MPVC podem ou não usar os mecanismos de cache oferecido pelo sistema de arquivos, dependendo da forma como o dono do computador estiver participando e das permissões que ele resolva dar aos *applets* descarregados (caso seja esta a forma de participação). Se os computadores estiverem usando *applets Java*, a interação será direta com os servidores. No entanto, sempre que possível, o componente do sistema de arquivos descarregado nos computadores participantes usará o disco local destes computadores para criar um cache com os arquivos usados pelas aplicações executadas nele. Como visto no início deste capítulo, este componente é o Módulo do Cliente (classe *FileClient*), que corresponde ao Componente dos Trabalhadores do serviço de arquivos de JOIN.

O Módulo do Cliente têm três funções principais no sistema de arquivos, as quais são mostradas a seguir.

- Criar e gerenciar uma lista com os servidores de arquivos disponíveis no sistema.
- Implementar a Interface do Sistema de Arquivos.
- Criar e gerenciar o cache de arquivos.

O restante desta seção apresenta como estas três funções são implementadas.

Lista de Servidores de Arquivos

A Figura 3.8 mostra como as informações dos servidores de arquivos disponíveis são mantidas pelo Módulo do Cliente.

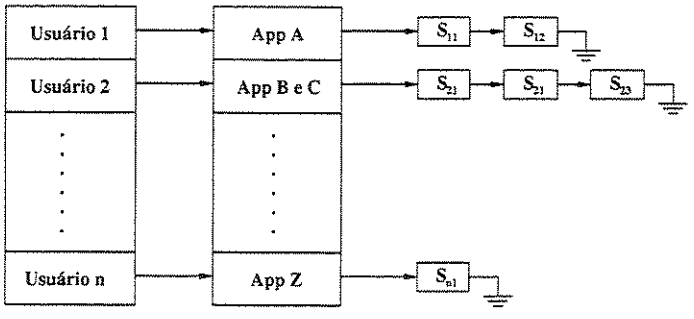

Figura 3.8: Lista de servidores de arquivos mantida no Módulo do Cliente

Cada Módulo do Cliente cria uma tabela para armazenar todas as informações dos servidores de arquivos disponíveis no sistema. Quando um novo computador entra no sistema e descarrega a plataforma básica, a primeira coisa que o Módulo do Cliente criado nesse computador faz é se comunicar com o Resolvedor de Endereços e obter os endereços e

características de todos os servidores registrados até esse momento. A seção 3.3.3 mostrou a forma de submeter aplicações no JOIN e como os Grupos de Servidores para Aplicações são associados a um determinado usuário. Portanto, com as informações obtidas no resolvedor, o Módulo do Cliente cria uma tabela, cujos elementos são da forma (usuário, aplicações, lista de servidores, Servidor *Default*). A lista de servidores é atualizada regularmente, sempre que o Módulo do Cliente é informado de que algum dos servidores mudou seu estado. Para cada usuário na tabela, o Módulo do Cliente calcula, usando a lista de servidores associada, qual será o seu Servidor *Default*. Toda requisição feita pelas aplicações desse usuário serão enviadas a este servidor. O cálculo deste Servidor *Default* é feito sempre que a lista de servidores é modificada. Desta forma, o sistema não precisa estar calculando o servidor mais adequado para enviar um determinado pedido no momento que ele chegar; isto somente é feito quando a lista de servidores muda, aumentando a eficiência do sistema. A forma como o Servidor *Default* é calculado foi explicada na seção 2.8.1.

A seguir veremos como o Módulo do Cliente comunica-se com o servidor de arquivos adequado para cada requisição recebida.

- Uma aplicação solicita uma operação no sistema de arquivos.
- O Módulo do Cliente recebe a solicitação e determina qual usuário fez o pedido (como todos os servidores são *stateless*, o nome do usuário está incluído no pedido).
- O Módulo do Cliente procura o usuário na tabela.
- Se o usuário for achado, o Módulo do Cliente envia o pedido ao Servidor *Default* desse usuário, que será encarregado de resolvê-lo.
- Se ele não for achado, o Módulo do Cliente não tem armazenada informação nenhuma dos servidores associados a este usuário. Nesse caso, o Módulo do Cliente se comunica novamente com o Resolvedor de Endereços para atualizar sua tabela, ou seja, somente quando não houver servidores disponíveis é que esta comunicação com o resolvedor é realizada. Esta possibilidade existe já que um servidor de arquivos (ou mesmo um grupo) pode ser criado após a criação do Módulo do Cliente. Sendo assim, o Módulo do Cliente não conhece seu endereço.
- Se após esta comunicação o Módulo do Cliente não obtiver o endereço de um servidor, então não se tem servidores disponíveis no grupo. Neste caso é usado o Servidor de Arquivos Central para atender os pedidos das aplicações desse usuário.

Interface do Sistema de Arquivos

Além de gerenciar os servidores de arquivos disponíveis, o Módulo do Cliente implementa o Componente dos Trabalhadores do serviço de arquivos, ou seja, implementa a Interface do Sistema de Arquivos, que define a forma como as aplicações podem interagir com este.

Para garantir portabilidade e total compatibilidade com JOIN, toda a implementação do Sistema de Arquivos é realizada usando Java. Esta linguagem oferece uma hierarquia de classes muito bem definida e com uma funcionalidade muito grande para tratar arquivos

e *streams* (Figura 3.9). Por exemplo, um programador pode utilizar um *stream* para ler ou escrever dados, sem se preocupar se os dados são lidos de um arquivo local, de um *socket* ou de um outro dispositivo. Além disso, também oferece a possibilidade de serializar objetos, isto é, enviar e receber objetos pela rede sem necessidade de se preocupar com a forma como estes objetos serão manipulados. A seguir são mostradas as classes principais que formam esta hierarquia.

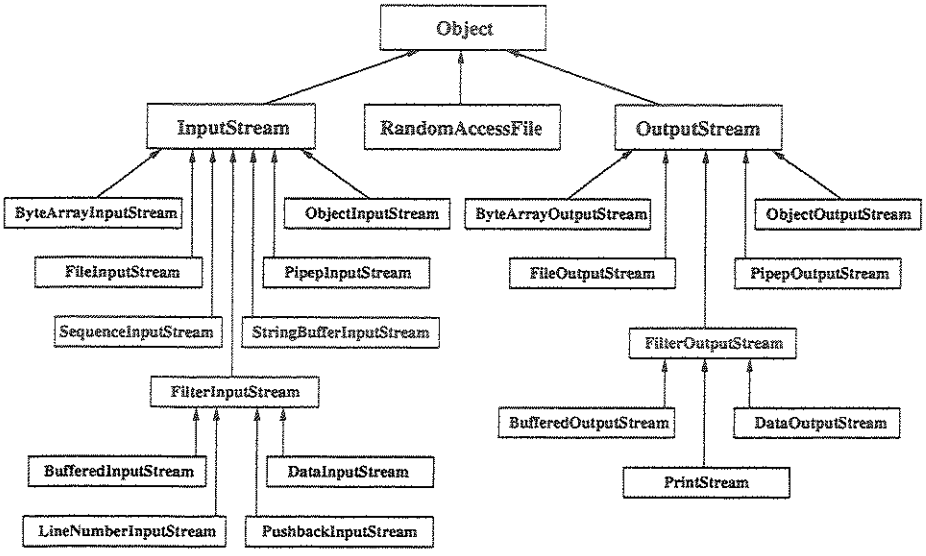

Figura 3.9: Principais classes para representar os streams em Java

No topo da hierarquia estão as classes *File*, *InputStream* e *OutputStream*, que definem a forma de gerenciar os arquivos no sistema de arquivos local e as interfaces dos *streams* de entrada e de saída, respectivamente. Os programadores em Java já estão acostumados à utilização das potencialidades que todas as classes desta hierarquia oferecem. Por esta razão e com o objetivo de facilitar a utilização por parte dos programadores, as classes implementadas no sistema de arquivos foram incorporadas a esta hierarquia, para desta forma explorar a sua funcionalidade. A Figura 3.10 mostra a incorporação das classes implementadas.

O acesso aos arquivos somente pode ser realizado usando-se as três classes definidas na interface do sistema (Apêndice C). A seguir são mostradas suas funções principais.

Classe *RemoteFile*: implementa a forma de manipular um arquivo remoto e seus atributos. Esta classe oferece às aplicações a funcionalidade necessária para criar e apagar Arquivos Regulares, criar, apagar e listar Diretórios, ter acesso e modificar os atributos dos arquivos, dentre outras funções. Além disso, é esta classe que implementa o espaço de nomes do sistema de arquivos (ver seção 2.5.2).

Classe *RemoteFileInputStream*: implementa um *stream* de entrada. A funcionalidade desta classe inclui abrir arquivos para leitura, fechá-los e ler dados deles, dentre outras. Esta classe implementa o mecanismo conhecido como *mark-reset*, que aumenta a

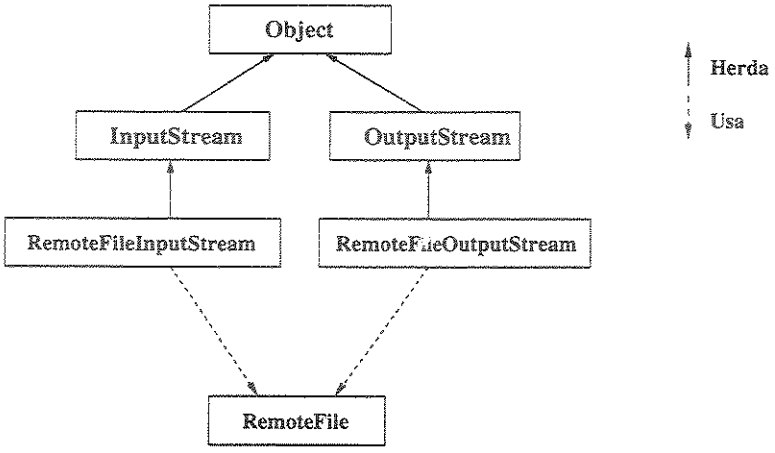

Figura 3.10: Principais classes para representar os *streams* no sistema de arquivo implementado

funcionalidade dos *streams* de entrada. Herda da classe *InputStream* de Java as funcionalidades mínimas para manipulação de um *stream* de entrada.

Classe *RemoteFileOutputStream*: implementa um *stream* de saída. Esta classe oferece funções para abrir arquivos para escrita, fechá-los, escrever dados neles, dentre outras. Ela implementa também a função *flush*, útil para salvar os dados de volta no servidor sem ter que fechar a arquivo. Herda da classe *OutputStream* de Java as funcionalidades mínimas para manipulação de *stream* de saída.

A seguir são apresentados maiores detalhes de como os diferentes tipos de arquivos são usados, controlados diretamente por estas classes, para a implementação do cache nos Módulos dos Clientes.

Cache nos Módulos dos Clientes

Na seção anterior foram explicadas as três classes principais que o sistema oferece às aplicações para manipular os arquivos. Os arquivos manipulados com a classe *RemoteFile*, que geralmente é usada para criar, apagar, obter ou modificar os atributos dos arquivos do sistema, não são replicados nos computadores colaboradores; eles são mantidos nos servidores e acessados diretamente pelos Módulos dos Clientes. Isto é devido a que as operações que esta classe implementa são operações simples, que não manipulam os dados armazenados nos arquivos e que são feitas poucas vezes (por exemplo, um arquivo é criado ou apagado só um vez), e mantê-los no cache não traria benefícios significativos. Além disso, esta abordagem permite que Arquivos Regulares e Diretórios criados por uma tarefa (por exemplo, a tarefa inicial de uma aplicação) sejam vistos rapidamente pelas outras tarefas e possam também ser utilizados por ela.

Já os manipulados com as classes *RemoteFileInputStream* e *RemoteFileOutputStream*, que permitem usar estes arquivos como *streams* para ler e escrever dados, são replicados no cache dos Módulos dos Clientes. Manter os dados dos arquivos nos cache dos Módulos dos

Clientes para serem usados pelas aplicações diminuiria o tempo de resposta das operações de escrita e leitura, o tráfego na rede e a carga dos servidores.

Portanto, o cache implementado somente replica os Arquivos Regulares que são abertos para leitura e/ou escrita; ele não replica Diretórios nem Arquivos Regulares que não sejam usados para estes fins.

A terceira função do Módulo do Cliente é o gerenciamento deste cache. Para isso, ele é auxiliado por outros Componentes de Serviços, que chamaremos de Gerentes de Arquivos do Cache (classe *ClientFileRequestManager*). Semelhante aos servidores, o cache dos Módulos dos Clientes é implementado usando-se várias linhas de controle, uma para cada arquivo mantido nele. Estes arquivos são mantidos em uma tabela criada pelo Módulo do Cliente. Cada arquivo que é aberto para escrita ou para leitura por uma tarefa sendo executada no computador trabalhador é colocado nessa tabela. Para cada um destes arquivos é criado um Gerenciador de Arquivos do Cache, que será encarregado de manipulá-lo. Da mesma forma que é feita nos servidores (explicados na seção 3.5.3), cada entrada na tabela armazena o nome do arquivo que está sendo usado e o TID do gerenciador (que é um Componente de Serviço, e portanto uma tarefa de sistema com seu TID próprio). No entanto, o tamanho desta tabela é variável e depende do número de tarefas que estão sendo executadas no computador e os arquivos que elas estão usando. Cada tarefa recebe o TID do Gerenciador de Arquivos do Cache que está manipulando o arquivo por ele referenciado. Desta forma, as tarefas podem interagir diretamente com estes gerenciadores sem a intervenção do Módulo do Cliente.

Para melhor entender a forma como o Módulo do Cliente gerencia o cache, a seguir serão explicadas com mais detalhes as operações feitas por este componente e como ele manipula os diferentes tipos de arquivos. Dependendo da classe que a aplicação esteja usando para interagir com o sistema de arquivos, o Módulo do Cliente atua de forma diferente.

- Se a operação foi feita através da classe *RemoteFile*, ela é diretamente gerenciada pelo Módulo do Cliente e enviada ao Servidor *Default* da aplicação. Para este tipo de operação não é usado o cache.

- Se a operação foi feita através da classe *RemoteFileInputStream*, ela só pode ser para abrir um novo arquivo para leitura. A primeira coisa que o Módulo do Cliente faz é procurar o arquivo na tabela de arquivos. Se o arquivo é encontrado lá, o Módulo do Cliente simplesmente retorna o TID do gerenciador que está manipulando o arquivo e incrementa o número de tarefas que estão usando o arquivo. Se o arquivo não está na tabela (é a primeira tarefa nesse computador que usa o arquivo), o Módulo do Cliente comunica-se com o Servidor *Default* da aplicação que realizou o pedido para saber se o mesmo existe. Se não existir, o Módulo do Cliente retorna uma mensagem de erro à aplicação. Se o arquivo existe, o Módulo do Cliente cria um novo arquivo no cache, cria um gerenciador para manipulá-lo (segundo foi explicado anteriormente) e envia para a tarefa o TID do gerenciador. As demais operações feitas através desta classe são enviadas diretamente pela tarefa para o gerenciador do arquivo.

- Se a operação foi realizada usando a classe *RemoteFileOutputStream*, ela também só pode ser para abrir um novo arquivo para escrita. Como a semântica usada no

sistema de arquivos é uma semântica de sessão, cada tarefa cria seu próprio arquivo, e por isso, não é necessário procurar na tabela. O Módulo do Cliente comunica-se com o Servidor *Default* da aplicação para obter o tamanho do arquivo. Logo após esta comunicação, o Módulo do Cliente cria um novo arquivo no cache, executa um novo Gerenciador de Arquivos do Cache para manipulá-lo e envia seu TID para a tarefa que fez a operação. Todas as outras operações realizadas pela tarefa nesse arquivo através dessa classe são enviadas diretamente ao gerenciador.

Resumindo, o Módulo do Cliente somente manipula as operações realizadas usando a classe *RemoteFile*. As requisições feitas através das classes que manipulam os *streams* são gerenciadas diretamente por um Gerenciador de Arquivos do Cache. Esta abordagem permite descentralizar a gerenciamento de cache, aumentando a sua eficiência e evitando que algum componente seja um possível ponto de gargalo.

Os gerenciadores também tratam de forma diferente os arquivos dependendo de como eles sejam usados. Os arquivos abertos para leitura são manipulados por classes diferentes das classes que manipulam arquivos abertos para escrita. As próximas seções explicam a forma de gerenciamento de cada tipo de *stream* no sistema. Vale lembrar que o sistema adotou a abordagem de não permitir que um mesmo arquivo seja aberto para leitura e escrita simultaneamente. Maiores detalhes se encontram nas seções 2.5.1 e 2.10.3.

Stream de Leitura

Os arquivos abertos para leitura são manipulados usando a classe *RemoteFileInputStream*. Para cada arquivo aberto é criado um gerenciador que o manipula. Este gerenciador atenderá diretamente todos os pedidos da tarefa que o criou e das tarefas nesse computador que o referenciem. Assim, somente uma cópia do arquivo é mantida no computador.

Além disso, o arquivo não é copiado completo no computador. Ele é copiado por blocos, usando-se uma política sob demanda para trazer novos blocos de dados dos servidores. O tamanho do bloco é definido pelo sistema (no próximo capítulo veremos como ele deve ser escolhido). Estes blocos podem ser removidos do disco local do computador para liberar espaço.

Os arquivos abertos para leitura são manipulados pelo gerenciador com ajuda da classe *ReadCacheFile*, pertence ao *kernel* do sistema de arquivos. Esta classe permite manter algumas informações do arquivo, como por exemplo o número de tarefas que o estão usando. A seguir são mostrados de forma resumida os passos realizados por um Gerenciador de Arquivos do Cache para manipular um *stream* de leitura.

- Quando o arquivo é aberto e antes que a tarefa faça alguma operação de leitura no arquivo, o gerenciador traz dos servidores o primeiro bloco de dados, copiando-o no cache.

- Quando a tarefa da aplicação realiza uma operação de leitura no arquivo, o gerenciador verifica se os dados requisitados já estão no cache. Se já se encontram no cache, pula-se o próximo passo.

- Se os dados necessários não estiverem no cache, o gerenciador comunica-se com os servidores para obtê-los do arquivo.
- O gerenciador resolve o pedido localmente e envia a resposta à tarefa.
- O arquivo é fechado pelo gerenciador.

O gerenciador também implementa um mecanismo que visa predizer futuras operações das aplicações. Como os arquivos são lidos do servidor por blocos, ou seja, eles não se encontram armazenados completamente no cache do computador, o Módulo do Cliente tenta fazer com que, em cada leitura feita por uma aplicação, os dados referenciados sejam encontrados localmente. Para evitar que esta comunicação com o servidor seja realizada no momento da operação, toda vez que uma leitura é recebida, processada e a resposta enviada à aplicação, o Módulo do Cliente compara a posição do último byte referenciado com a quantidade de bytes do arquivo disponíveis localmente no computador. Se esta posição está se aproximando ao número de bytes disponíveis localmente significa que provavelmente em futuras operações de leituras será necessária uma comunicação com o servidor para obter um outro bloco de dados. Por isso, segundo um determinado critério (por exemplo que a quantidade de byte referenciados pelas aplicações ultrapasse 75% da quantidade de bytes disponíveis localmente no computador), o Módulo do Cliente comunica-se com o servidor para obter novos dados. Esta operação é feita logo após da operação de leitura feita pela aplicação com o objetivo de diminuir o impacto no tempo de resposta da mesma.

Coloquemos um exemplo concreto para entender melhor o mecanismo implementado. Vamos supor que o tamanho dos blocos lidos do servidor para o cache é de 10 KBytes e que está sendo executada uma aplicação que faz leituras sucessivas de 1 KBytes em um arquivo X. Quando o arquivo é aberto pela aplicação, é criada uma cópia do mesmo no cache local (que chamaremos de X_1) e são lidos do servidor os primeiros 10 KBytes de dados do arquivo original. A cada operação de leitura feita pela aplicação (leituras de 1 KBytes de dados, realizadas diretamente sobre X_1), o Módulo do Cliente faz a comparação antes mencionada. Se a quantidade de bytes já lidos representa mais que 75% da quantidade de bytes disponíveis localmente no computador, ou seja, se já foram lidos mais de 7,5 KBytes de X_1, é solicitada automaticamente a leitura de um novo bloco.

Embora o mecanismo usado seja muito simples, em determinados casos, como por exemplo leituras sucessivas em um arquivo, pode ser obtido um aumento na eficiência destas operações, já que em cada leitura realizada os dados provavelmente sejam encontrados no cache local, sendo desnecessária uma comunicação com o servidor no momento da operação.

A operação de fechar um arquivo usado para leitura é muito simples. Basta decrementar o número de usuários que têm o arquivo aberto. Se o arquivo já não está sendo usado por outras tarefas, o Módulo do Cliente pode liberar os recursos usados localmente pelo arquivo. Nesse caso ele comunica-se com o servidor para que ele também decremente o número de cópias espalhadas no sistema. Manter a informação de quantos Módulos dos Clientes fizeram cópias do arquivo para leitura é importante, já que somente quando todas as tarefas fecharem o arquivo (aberto para leitura), ele poderá ser usado para escrita. Neste ponto, surge uma pergunta: e se algum dos Módulos dos Clientes falhar e não conseguir fechar o arquivo? Nesse caso, o contador ficaria desatualizado e o arquivo não poderia ser usado

para escrita. Para resolver esse problema, o sistema define um *timeout* de não utilização dos arquivos como um dos parâmetros de configuração do SAD_GSA. Se um arquivo aberto para leitura não é acessado por nenhuma tarefa no período de tempo definido, ele é fechado, evitando que ele fique nesse estado inconsistente de forma indefinida.

Stream de Escrita

No lado oposto estão os arquivos abertos para escrita. Eles são manipulados pela classe *RemoteFileOutputStream*. Da mesma forma que para *streams* de leitura, será criado um Gerenciador de Arquivos do Cache para sua manipulação, atendendo diretamente aos pedidos de escrita da tarefa que o criou. Como foi mostrado na seção 2.10.3, é utilizada uma semântica de sessão no sistema de arquivos. Se uma tarefa abre um arquivo para escrita, somente quando ele for fechado as mudanças nele serão visíveis pelas demais tarefas. É por isso que o Módulo do Cliente cria um arquivo para cada tarefa diferente.

No caso dos arquivos abertos para escrita, não é necessário se comunicar com os servidores no processamento do arquivo nem trazer dados do arquivo para o cache local. O Módulo do Cliente mantém as diferentes cópias dos arquivos de cada tarefa e só quando o arquivo é fechado é que os dados fornecidos pela tarefa são enviados ao servidor.

A classe *WriteCacheFile* é usada no gerenciamento dos arquivos abertos para escrita. Esta classe também pertence ao *kernel* do sistema de arquivos. Os passos que um gerenciador executa para manipular um *stream* de escrita são mostrados a seguir de forma resumida. Estes passos são mais simples que no caso dos *stream* de entrada.

- Quando o arquivo é aberto, o gerenciador comunica-se com um dos servidores para determinar se ele existe, e abrí-lo. Se não existe, cria-se o arquivo.
- Todas as operações de escrita são realizadas no arquivo local. O gerenciador resolve o pedido localmente e envia a resposta à tarefa.
- Se houver uma operação do tipo *flush*, o gerenciador envia todos os dados do arquivo para o servidor para serem armazenados em um lugar seguro.
- O gerenciador fecha o arquivo.

O operação de fechar um arquivo usado para escrita é mais complexa que nos *streams* de leitura. O gerenciador fecha o arquivo local e libera todos os recursos usados por esse arquivo. Para garantir a semântica definida, todas as mudanças no arquivo são enviadas para os servidores e assim, se uma nova tarefa abre o arquivo, pode ver as mudanças realizadas. Além disso, o número de arquivos abertos para escrita mantido nos servidores tem que ser decrementado. Esta informação é importante, já que somente quando todos os arquivos fecharem o arquivo usado para escrita, ele poderá ser usado para leitura. O *timeout* de não utilização dos arquivo explicado na seção anterior também é usado para arquivos abertos para escrita.

3.6 Sumário

Para concretizar as idéias propostas neste trabalho, foi usado JOIN como plataforma de desenvolvimento do SAD_GSA. JOIN é um sistema que implementa um computador paralelo virtual baseado na Internet que tira proveito de Java como uma linguagem de programação independente de plataforma. Diferente de outras propostas com o mesmo objetivo, JOIN tenta atacar os principais problemas encontrados em sistemas deste tipo, como por exemplo a escalabilidade e a tolerância a falhas.

Com o objetivo de aumentar a flexibilidade e facilitar a implementação e manutenção do sistema, JOIN foi implementado usando o conceito dos *microkernel*. Este *microkernel* está dividido em duas partes principais, o *kernel*, que oferece as funcionalidades mínimas do sistema, e os serviços, que permitem a extensão das funcionalidades do sistema. O SAD_GSA foi implementado como um serviço em JOIN.

Além de implementar os diferentes Componentes de Serviços necessários, SAD_GSA criou um conjunto de classes que formam a base do sistema de arquivos. A implementação pode ser dividida em vários grupos, mostrados a seguir.

No primeiro deles, agrupam-se as classes relacionadas com o gerenciamento de arquivos e diretórios. Estas classes permitem a criação e manipulação de arquivos e diretórios, assim como a manipulação dos seus atributos. Um ponto importante para se ressaltar é a implementação de especializações da classe para a manipulação de tipos de arquivos específicos, como por exemplo os arquivos mantidos no cache dos Módulos dos Clientes.

O segundo grupo está formado por um conjunto de classes muito vinculadas às classes citadas acima. Estas classes são comandos externos oferecidos pelo sistema de arquivos para a realização de operações como criação do Sistema de Diretórios do SAD_GSA, criação de usuários, criação, cópia e remoção de arquivos e diretórios, importação e exportação de arquivos, assim como vários utilitários úteis para o sistema.

O terceiro grupo de classes está relacionado com a implementação dos servidores de arquivos. Cada servidor de arquivo do SAD_GSA usa várias linhas de execução (servidores *multithreads*), com o objetivo de aumentar a eficiência dos mesmos. Também neste grupo encontra-se a classe que implementa os Grupos de Servidores para Aplicações.

O último grupo está formados pelas classes que implementam os Módulos dos Clientes. Estas classes estão divididas em dois subgrupos. O primeiro agrupa as classes para a manipulação dos arquivos e a comunicação com o Resolvedor de Endereços. O outro subgrupo está formado pela Interface do Sistema de Arquivos. No Apêndice B tem-se um diagrama com todas as classes implementadas, assim como a relação entre elas. Já as classes que formam a Interface do Sistema de Arquivos são mostradas em detalhes no Apêndice C.

Uma vez implementado o sistema de arquivos, é preciso testá-lo e analisar seu desempenho. O capítulo seguinte faz uma série de testes com o sistema de arquivos e implementa algumas aplicações (cálculo de números primos e multiplicação de matrizes) usando os serviços que ele oferece. Estes testes permitem tirar conclusões sobre a eficiência, escalabilidade e tolerância a falhas do SAD_GSA.

Capítulo 4

Testes e Resultados

Este capítulo apresenta os testes realizados com o SAD_GSA e os resultados obtidos nos mesmos. Assim, foi possível determinar com maior exatidão a influência do sistema de arquivos no desempenho geral de um sistema paralelo como o JoiN.

4.1 Condições dos Testes

Todos os testes foram realizados usando os recursos disponíveis nos laboratórios da Faculdade de Engenharia Elétrica e de Computação citados a seguir.

Laboratório do Departamento de Engenharia de Computação e Automação Industrial (LCA): rede local que combina os padrões *Ethernet* (10 Mbits), *FastEthernet* (100 Mbits) e ATM (155 Mbits). Os computadores são estações Sparc da SUN executando o sistema operacional Solaris e PCs executando os sistemas operacionais Linux e Windows NT.

Laboratório do Departamento de Sistemas de Energia Elétrica (DSEE): rede local que combina os padrões *Ethernet* e *FastEthernet*. Os computadores são estações Sparc da SUN executando o sistema operacional Solaris.

Laboratório do Setor de Informática (SIFEEC): rede local padrão *Ethernet*. Os computadores são estações Sparc da SUN executando o sistema operacional Solaris, e PCs executando os sistemas operacionais Linux e Windows NT.

A descrição dos computadores usados nos testes (nome, tipo de computador, sistema operacional, função no sistema de arquivos, quantidade de memória, CPU e velocidade do enlace) em cada laboratório é mostrada nas Tabelas 4.1, 4.2 e 4.3.

No total, foram usados 35 computadores, 13 do LCA, 11 do DSEE e 11 do SIFEEC. Os testes foram realizados em horários de pouca carga de trabalho para evitar que fatores externos influenciassem nos resultados. Além disso, os computadores mais rápidos em cada laboratório foram escolhidos para executar os principais componentes do sistema de arquivos, tais como o Servidor de Arquivos Central, o Resolvedor de Endereços e os Servidores de Arquivos para Aplicações.

Computador	Tipo	SO	Função	Memória	CPU	Enlace
rocas	WS	Unix	Serv. Central	256 MBytes	360 Mhz	100 Mbits
dunas	WS	Unix	Resolvedor	256 MBytes	360 Mhz	100 Mbits
jureia	WS	Unix	Serv. Aplic.	256 MBytes	360 Mhz	100 Mbits
grumari	WS	Unix	Mód. Cliente	256 MBytes	187 Mhz	100 Mbits
copacabana	WS	Unix	Mód. Cliente	256 MBytes	187 Mhz	100 Mbits
buzios	WS	Unix	Mód. Cliente	256 MBytes	187 Mhz	100 Mbits
bombas	WS	Unix	Mód. Cliente	256 MBytes	143 Mhz	100 Mbits
botafogo	WS	Unix	Mód. Cliente	190 MBytes	200 Mhz	155 Mbits
calhau	WS	Unix	Mód. Cliente	128 MBytes	270 Mhz	100 Mbits
enseada	WS	Unix	Mód. Cliente	64 MBytes	167 Mhz	155 Mbits
itanhaem	WS	Unix	Mód. Cliente	64 MBytes	167 Mhz	100 Mbits
laguna	PC	Linux	Mód. Cliente	384 MBytes	833 Mhz	100 Mbits
atlantida	PC	NT	Mód. Cliente	64 MBytes	600 Mhz	100 Mbits

Tabela 4.1: Descrição dos computadores usados do LCA

Computador	Tipo	SO	Função	Memória	CPU	Enlace
baiacu	WS	Unix	Serv. Aplic.	512 MBytes	300 Mhz	10 Mbits
jau	WS	Unix	Mód. Cliente	132 MBytes	167 Mhz	10 Mbits
tambacu	WS	Unix	Mód. Cliente	132 MBytes	167 Mhz	10 Mbits
atum	WS	Unix	Mód. Cliente	132 MBytes	167 Mhz	10 Mbits
tubarao	WS	Unix	Mód. Cliente	88 MBytes	167 Mhz	100 Mbits
piava	WS	Unix	Mód. Cliente	88 MBytes	167 Mhz	100 Mbits
cascudo	WS	Unix	Mód. Cliente	88 MBytes	167 Mhz	100 Mbits
lambari	WS	Unix	Mód. Cliente	88 MBytes	167 Mhz	100 Mbits
tuvira	WS	Unix	Mód. Cliente	88 MBytes	167 Mhz	100 Mbits
carpa	WS	Unix	Mód. Cliente	88 MBytes	167 Mhz	100 Mbits
dourado	WS	Unix	Mód. Cliente	88 MBytes	167 Mhz	100 Mbits

Tabela 4.2: Descrição dos computadores usados do DSEE

Computador	Tipo	SO	Função	Memória	CPU	Enlace
isolda	WS	Unix	Serv. Aplic.	128 MBytes	110 Mhz	10 Mbits
balin	WS	Unix	Mód. Cliente	64 MBytes	110 Mhz	10 Mbits
maugantius	WS	Unix	Mód. Cliente	64 MBytes	110 Mhz	10 Mbits
ambrosius	WS	Unix	Mód. Cliente	32 MBytes	110 Mhz	10 Mbits
arthur	WS	Unix	Mód. Cliente	32 MBytes	110 Mhz	10 Mbits
viviane	WS	Unix	Mód. Cliente	32 MBytes	110 Mhz	10 Mbits
lancelot	WS	Unix	Mód. Cliente	32 MBytes	110 Mhz	10 Mbits
cynwyl	WS	Unix	Mód. Cliente	32 MBytes	110 Mhz	10 Mbits
polaris	PC	Linux	Mód. Cliente	32 MBytes	400 Mhz	10 Mbits
storm	PC	Linux	Mód. Cliente	32 MBytes	400 Mhz	10 Mbits
tolkien	PC	NT	Mód. Cliente	32 MBytes	400 Mhz	10 Mbits

Tabela 4.3: Descrição dos computadores usados do SIFEEC

4.2 Classificação dos Testes

Para melhor organizar os testes e obter resultados que permitissem avaliar o sistema de arquivos implementado, os mesmos foram organizados como mostrado a seguir.

Testes de configuração: testes para determinar os parâmetros ótimos do sistema e assim aumentar a sua eficiência.

- Testes para determinar o tamanho ideal dos blocos de informação que são enviados pela rede. Uma boa escolha auxilia na obtenção de melhores tempos de resposta às aplicações.
- Testes para determinar o impacto no sistema causado pelo uso de mecanismos de compressão da informação que é transmitida pela rede.

Testes iniciais: testes para determinar se o sistema de arquivos funciona corretamente e satisfaz os principais requisitos desejados, tais como o uso do cache, a escalabilidade e a tolerância a falhas. Além disso, eles permitem testar os principais mecanismos implementados, como a replicação de servidores, o balanceamento de carga e a reconfiguração automática, dentre outros aspectos.

- Testes para determinar o impacto do uso de cache na eficiência do sistema de arquivos.
- Testes de escalabilidade usando acesso intensivo aos recursos, para saber como o sistema reage ao aumento na carga de trabalho e determinar o compromisso entre o número de réplicas necessárias para atender uma aplicação e a eficiência com que os pedidos são satisfeitos.

- Testes de robustez, para determinar como o sistema reage ante possíveis falhas dos seus componentes e assim determinar o grau de confiabilidade e eficiência do mecanismo de detecção e recuperação de falhas implementado.

Testes com aplicações: testes realizados com aplicações reais que usem de forma intensiva os recursos oferecidos pelo sistema de arquivos para determinar a sobrecarga imposta a uma aplicação pelo uso do sistema de arquivos. Esta sobrecarga sempre existe (os acessos de E/S ainda são as operações mais lentas), mas deve ser razoável para garantir bons resultados.

Para os dois primeiros grupos de testes, foram implementadas duas aplicações muito simples, cujo principal objetivo foi ter acesso de forma intensiva aos recursos do sistema de arquivos. Ambas as aplicações criam um determinado número de tarefas que interagem de forma intensiva com o sistema de arquivos. A primeira aplicação tem como objetivo testar as operações de leitura. Cada uma das tarefas criadas lê de um arquivo uma determinada quantidade de bytes em intervalos aleatórios, até ler o arquivo completo. O objetivo da segunda aplicação foi testar as operações de escrita. Para isso, cada uma das tarefas da aplicação escrevem em um arquivo uma determinada quantidade de bytes até alcançar um certo tamanho. O número de tarefas criadas, a quantidade de bytes lidas ou escritas nos arquivos, assim como o tamanho dos arquivos usados são parâmetros das aplicações. Desta forma podem ser obtidos diferentes resultados dependendo das condições dos testes realizados. Para o terceiro grupo de testes foram implementadas duas aplicações de maior complexidade: busca de números primos grandes e multiplicação de matrizes. Na seção 4.5 tem-se uma explicação mais detalhadas destas aplicações.

Um aspecto importante a ser especificado antes de começar a explicação dos testes realizados é a forma como as tarefas criadas são distribuídas entre os colaboradores que formam o MPVC. Esta distribuição é uma das tarefas principais de JoiN, que implementa mecanismos para fazer esta distribuição da melhor forma possível. O mecanismo de escalonamento usado pelo JoiN quando o sistema de arquivos proposto foi implementado era muito simples. JoiN simplesmente dividia o número de tarefas criadas entre os colaboradores existentes, ou seja, cada colaborador executa o mesmo número de tarefas. Atualmente JoiN oferece mecanismos de escalonamento mais sofisticados e eficientes.

4.3 Testes de Configuração

O objetivo principal deste primeiro grupo de testes foi determinar valores ótimos para alguns parâmetros do sistema, tentando aumentar a sua eficiência.

4.3.1 Tamanho dos Blocos

Os primeiros testes realizados tiveram como objetivo determinar qual seria o tamanho ideal dos blocos enviados pela rede. Blocos pequenos são ineficientes, mas blocos muito grandes podem gastar tempo com informações desnecessárias.

Para estes testes somente foi usada a aplicação que realiza operações de leitura, sem fazer uso do cache nos Módulos dos Clientes para forçar o tráfego de informações pela rede. A aplicação foi configurada para criar tarefas que variam o tamanho dos blocos lidos do arquivo de 1 byte até 1 Mbytes. O tamanho total do arquivo usado foi de 2 MBytes. Desta forma, pôde ser determinado com qual tamanho de bloco obtêm-se melhores resultados, isto é, para qual tamanho do bloco o arquivo é lido por completo no menor tempo. O Grupo de Servidores para Aplicações usado foi composto por somente um Servidor de Arquivos para Aplicações (jureia do LCA) e o número de computadores do MPVC foi de 15 (5 computadores de cada laboratório), considerando somente aqueles usados como Módulos dos Clientes do sistema de arquivos.

A Tabela 4.4 mostra os resultados dos testes, separando o tempo que uma operação de leitura leva para ser processada pelo servidor e o tempo que leva sua transmissão pela rede.

Tamanho dos blocos	**Tempos médios**						
	t_{p_s} (ms)	t_c (ms)				t_{l_b} (ms)	t_{l_a} (seg)
		LCA	DSEE	SIFEEC	Média		
1 MBytes	247,91	613,40	1400,34	1696,93	1236,89	1484,80	3,27
512 KBytes	21,63	411,21	801,18	941,05	717,81	739,44	3,26
256 KBytes	16,51	206,65	381,62	457,23	348,50	365,01	3,21
128 KBytes	7,50	134,05	233,17	269,27	212,16	219,66	3,86
64 KBytes	4,29	59,72	119,90	140,99	106,87	111,16	3,92
32 KBytes	3,30	35,29	58,64	70,36	54,76	58,06	4,09
16 KBytes	2,01	20,56	40,82	47,68	36,35	38,36	5,40
14 KBytes	1,78	17,88	35,45	41,78	31,70	33,48	5,41
12 KBytes	1,57	15,30	37,56	44,34	32,40	33,97	6,34
10 KBytes	1,46	13,86	34,32	41,02	29,73	31,19	7,01
8 KBytes	1,20	56,49	78,20	92,49	75,73	76,93	21,66
6 KBytes	1,26	57,03	79,97	93,78	76,93	78,19	27,86
4 KBytes	1,22	53,67	77,15	91,25	74,02	75,24	42,37
3 KBytes	1,16	53,65	75,77	87,31	72,24	73,40	55,15
2 KBytes	1,16	49,42	66,36	77,82	64,53	65,69	74,01
1 KBytes	1,07	46,14	63,76	76,08	61,99	63,06	166,70
512 bytes	1,05	7,63	24,06	28,57	20,09	21,13	92,21
256 bytes	1,02	7,57	23,25	27,86	19,56	20,58	185,54
128 bytes	1,00	6,91	24,17	28,19	19,76	20,76	374,14
64 bytes	1,01	7,16	23,40	27,31	19,29	20,30	731,35
32 bytes	0,98	6,85	23,25	27,58	19,23	20,21	1456,26
16 bytes	0,99	7,05	22,60	26,59	18,75	19,74	*
8 bytes	0,94	7,07	23,69	27,63	19,46	20,40	*
1 byte	0,96	7,06	22,91	27,20	19,06	20,02	*

Tabela 4.4: Resultados dos testes de tamanho dos blocos

Os resultados foram obtidos como mostrado a seguir. A aplicação de leitura, para cada operação realizada, armazenava em um arquivo (que chamaremos de arquivo cliente) o tempo que levava o sistema para processá-la (t_{p_c}). O servidor usado também armazenava em outro arquivo (que chamaremos de arquivo servidor) o tempo que ele levava para processar o pedido de um bloco (t_{p_s}). Sendo assim, o tempo de processamento de uma operação de leitura no servidor é o tempo armazenado no arquivo servidor, e o tempo de comunicação (t_c) é o tempo armazenado no arquivo cliente menos o tempo armazenado no arquivo servidor (Equação 4.1). O tempo total da operação de leitura de um bloco no Módulo do Cliente (t_{l_b}) é o valor armazenado no arquivo cliente que, conseqüentemente, é a soma do tempo de processamento de uma operação de leitura no servidor e o tempo de comunicação (Equação 4.2). Ou seja, o tempo de processamento e o tempo total foram medidos, e o tempo de comunicação foi calculado. Também foi medido o tempo total que levou o sistema para processar o arquivo todo (t_{l_a}).

$$t_c = t_{p_c} - t_{p_s} \tag{4.1}$$

$$t_{l_b} = t_{p_c} = t_{p_s} + t_c \tag{4.2}$$

A Figura 4.1 mostra graficamente os resultados obtidos nos testes para blocos de tamanho acima de 8 KBytes.

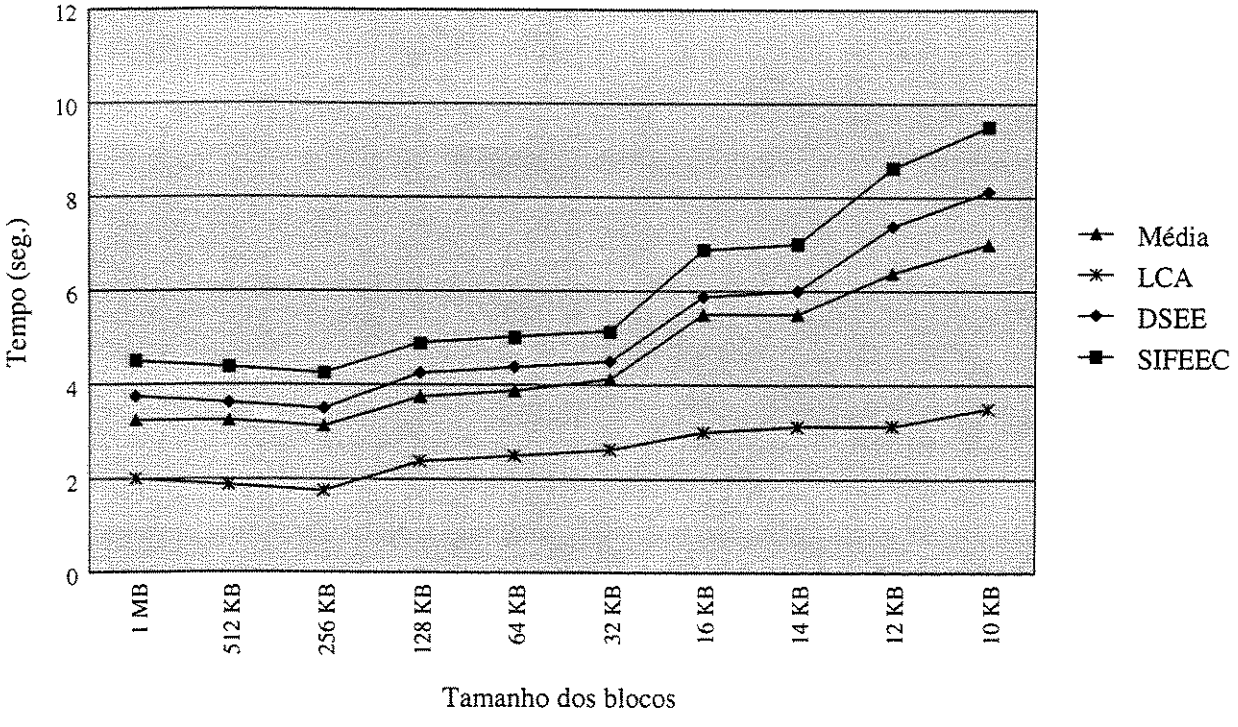

Figura 4.1: Tempo de leitura de arquivo em blocos acima de 8 KBytes

Para cada tamanho de bloco os testes foram repetidos 25 vezes, calculando-se os valores médios dos tempos de processamento, comunicação e total. Todos os tempos armazenados

nos arquivos foram somados e divididos pelo número de operações de leitura realizadas para cada tamanho de bloco. Os tempos de comunicação foram divididos por laboratórios e o tempo médio de comunicação foi calculado somando os resultados obtidos em cada laboratório e dividindo entre o número de laboratórios (3).

Os valores marcados com (*) não foram calculados devido à demora excessiva dos mesmos (o processamento do arquivo completo demoraria horas).

Os resultados mostram que para blocos de tamanhos 256 KBytes obteve-se o menor tempo. No entanto, blocos desse tamanho podem trazer informações desnecessárias. Assim, o sistema de arquivos adota tamanhos de blocos que variam entre 256 KBytes e 64 KBytes (valor configurável do sistema), faixa que ainda apresenta resultados satisfatórios (isto é, tempos até 25% acima do tempo mínimo).

4.3.2 Uso de Compressão da Informação Transmitida pela Rede

Os últimos testes deste grupo tiveram como objetivo determinar o impacto do uso de mecanismos de compressão da informação transmitida pela rede no desempenho do sistema, determinando se é mais eficiente comprimir a informação antes de enviá-la.

Para isto, foram usadas as mesmas condições do teste anterior. Desta forma, pôde-se obter o compromisso entre o tempo que demora a compressão e o tempo que demora o envio pela rede. Para a compressão, foram usadas as facilidades oferecidas pelo pacote *java.utils.zip*, que fornece classes para ler e escrever arquivos no formato ZIP. As taxas de compressão dos blocos dos arquivos usados nos testes esteve entre 25% e 50% do tamanho inicial.

Para obter os resultados destes testes, foram realizadas duas medidas. A primeira delas, que chamaremos tempo de *setup*, é o tempo que leva a compressão do bloco que será transmitido pela rede. O segunda medida, que chamaremos de tempo de transmissão, é o tempo gasto no envio do bloco pela rede. O tempo total para enviar o bloco pode ser obtido somando ambos tempos. Os testes também foram repetidos 25 vezes para cada tamanho de bloco.

A Tabela 4.5 mostra os resultados dos testes realizados, incluindo os tempos gastos para simplesmente enviar o bloco pela rede (dados obtidos a partir da Tabela 4.4, somando o tempo de processamento no servidor com o tempo para o envio da informação pela rede), e o tempo necessário para a compressão e envio das informações pela rede. Os resultados, como nos testes anteriores, também foram divididos por laboratórios.

A Tabela 4.6 mostra a variação no tempo médio entre as duas abordagens, mostrando a partir de que tamanho de bloco seria mais eficiente aplicar o mecanismo de compressão. Foi calculada a variação entre o tempo que levou o sistema para enviar um bloco sem usar o mecanismo de compressão ($t_{l_{bunzip}}$) e o tempo que leva usando-o ($t_{l_{bzip}}$) (Equação 4.3). Valores negativos indicam que são obtidos melhores resultados se o bloco é comprimido antes de ser enviado pela rede. Valores positivos indicam que não seria necessário, para esse tamanho de bloco em questão, se usar o mecanismo de compressão.

$$t_v = \left(\frac{t_{l_{bzip}} - t_{l_{bunzip}}}{t_{l_{bunzip}}} \right) * 100\% \tag{4.3}$$

Tam. dos blocos	Sem compressão (ms)			Com compressão (ms)					
	LCA	DSEE	SIFEEC	LCA		DSEE		SIFEEC	
				Setup	Envio	Setup	Envio	Setup	Envio
1 MB	861,31	1648,25	1944,84	486,79	453,81	511,13	855,72	541,47	997,81
512 KB	432,84	822,81	962,68	240,42	232,09	252,44	418,04	264,46	497,43
256 KB	223,16	398,13	473,74	130,86	148,63	137,41	239,11	143,95	290,61
128 KB	141,55	240,67	276,77	97,52	67,85	102,97	121,16	107,26	152,53
64 KB	64,01	124,19	145,28	60,98	40,13	64,03	57,71	67,08	77,34
32 KB	38,59	61,94	73,66	33,93	23,47	36,63	44,97	37,32	52,18
16 KB	22,57	42,83	49,69	19,38	20,69	20,35	39,11	21,82	45,74
14 KB	19,66	37,23	43,56	10,42	17,82	10,94	40,69	11,46	48,21
12 KB	16,87	39,13	45,91	6,91	15,93	7,23	37,93	7,61	44,65
10 KB	15,32	35,78	42,48	5,89	19.42	6,18	45.37	6,48	58,37
8 KB	57,69	79,40	93,69	5,61	60,04	5,89	84,48	6,17	99,72
6 KB	58,29	81,23	95,04	5,38	57,08	5,65	83,07	5,92	97,09
4 KB	54,89	78,37	92,47	4,91	56,49	5,16	80,78	5,41	92,89
3 KB	54,81	76,93	88,47	4,83	52,71	5,07	74,22	5,13	86,43
2 KB	50,58	67,52	78,98	4,56	48,49	4,79	67,42	5,01	81,01
1 KB	47,21	64,83	77,15	4,12	46,51	4,32	65,82	5,53	78,83

Tabela 4.5: Resultados dos testes de compressão da informação enviada pela rede

A Figura 4.2 apresenta os dados obtidos de forma gráfica, representando somente os blocos de tamanho até 10 Kbytes. Os blocos menores de 10 Kbytes não foram considerados já que não aportavam nenhum elemento novo a ser considerado.

Os resultados obtidos mostram que para os computadores do LCA (laboratório onde encontra-se o servidor usado nos testes) o processo de compressão não é necessário, já que os resultados sem o uso deste recurso são sempre melhores. Este resultado é devido a que, como neste laboratório tem-se uma rede local rápida, os tempos de transmissão dos blocos pela rede são inferiores e influenciam em menor escala no resultado final. No entanto, para os computadores fora do LCA não acontece a mesma coisa. Para transmitir blocos de informações de um tamanho em torno de 64 KBytes ou maiores obtém-se melhoras visíveis usando compressão. Isto é devido ao maior atraso da rede que comunica estes laboratórios. Este atraso crescerá ainda mais para redes mais lentas e/ou distantes. Por isso, como o sistema de arquivos proposto será usado no processamento maciçamente paralelo na Internet (rede distribuída geograficamente e com uma latência alta nas comunicações), sempre será usado usa o mecanismo de compressão nas mensagens com tamanho acima de 64 KBytes.

Tam. dos blocos	Tempos médios (ms)								
	LCA			DSEE			SIFEEC		
	$t_{l_{bunzip}}$	$t_{l_{bzip}}$	$t_v(\%)$	$t_{l_{bunzip}}$	$t_{l_{bzip}}$	$t_v(\%)$	$t_{l_{bunzip}}$	$t_{l_{bzip}}$	$t_v(\%)$
1 MB	861,31	940,60	+9,21	1648,25	1366,85	-17,07	1944,84	1539,28	-20,85
512 KB	432,84	472,51	+9,17	822,81	670,48	-18,51	962,68	761,89	-20,86
256 KB	223,16	279,49	+25,24	398,13	376,52	-5,43	473,74	434,56	-8,27
128 KB	141,55	165,37	+16,83	240,67	224,13	-6,87	276,77	259,79	-6,14
64 KB	64,01	101,11	+57,96	124,19	121,74	-1,97	145,28	144,42	-0,59
32 KB	38,59	57,40	+48,74	61,94	81,60	+31,74	73,66	89,50	+21,50
16 KB	22,57	40,07	+77,54	42,83	59,46	+38,83	49,69	67,56	+35,93
14 KB	19,66	28,24	+43,64	37,23	51,63	+38,68	43,56	59,67	+36,98
12 KB	16,87	22,84	+35,39	39,13	45,16	+15,41	45,91	52,26	+13,83
10 KB	15,32	25,31	+65,21	35,78	51,55	+30,59	42,48	64,85	+52,66
8 KB	57,69	65,65	+13,80	79,40	90,37	+13,82	93,69	105,89	+13,02
6 KB	58,29	62,46	+7,15	81,23	88,72	+9,22	95,04	103,01	+8,39
4 KB	54,89	61,40	+11,86	78,37	85,94	+9,66	92,47	98,30	+6,30
3 KB	54,81	57,54	+4,98	76,93	79,29	+3,07	88,47	88,06	+3,49
2 KB	50,58	53,05	+4,88	67,52	72,21	+6,95	78,98	86,02	+8,91
1 KB	47,21	50,63	+7,24	64,83	70,14	+8,19	77,15	84,36	+9,35

Tabela 4.6: Comparação entre tempos de processamento por blocos com e sem compressão

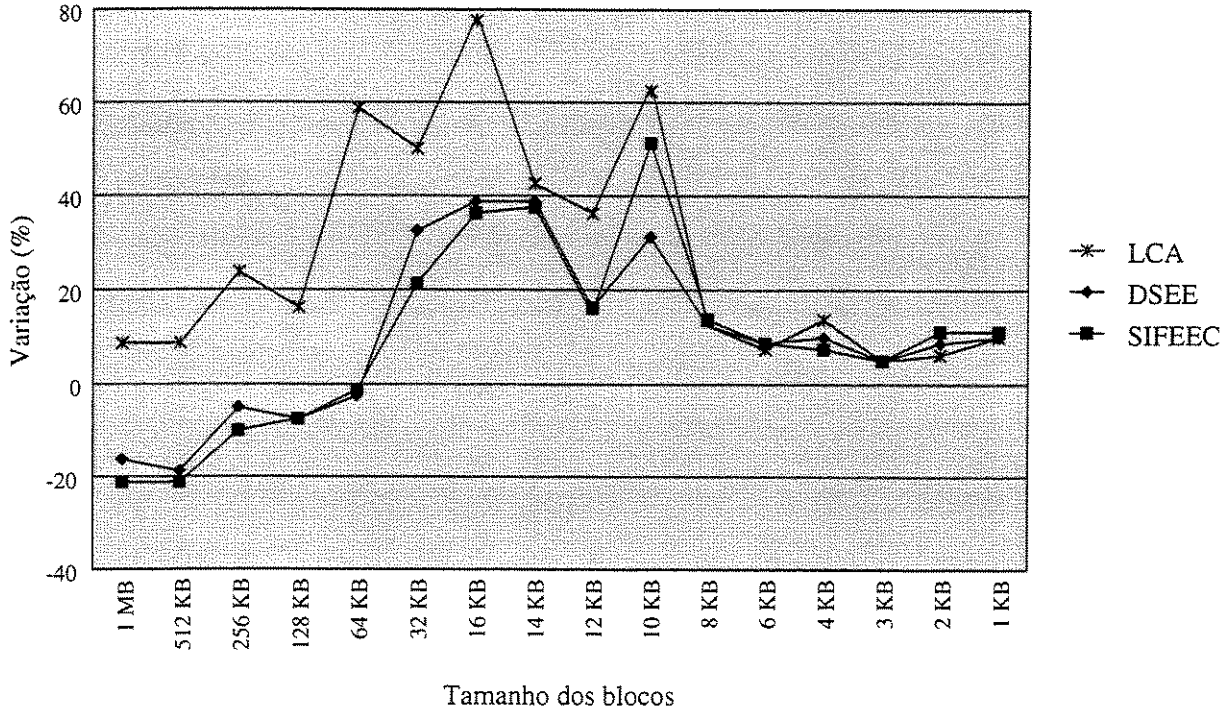

Figura 4.2: Diferença no tempo de leitura de arquivo quando é usada compressão

4.4 Testes Iniciais

Devido ao fato de que o sistema de arquivos distribuído foi projetado para satisfazer as necessidades de uma plataforma para o processamento maciçamente paralelo na Internet, as finalidades deste segundo grupo de testes foram analisar o impacto do uso de cache na eficiência do sistema, analisar como o sistema reage quando a carga de trabalho aumenta, comprovar se o sistema reage ante possíveis falhas e quanto ele demora para se reconfigurar. Os testes foram divididos em 3 grupos. Em todos os testes foram usados os principais mecanismos oferecidos pelo sistema de arquivos, tais como a replicação de servidores, o balanceamento de carga e a reconfiguração automática.

4.4.1 Cache no Módulo do Cliente

Os primeiros testes deste grupo foram realizados com o objetivo de determinar o impacto final do uso de cache no desempenho e eficiência do sistema de arquivos implementado. Para obter estes resultados, foram usadas as aplicações citadas na seção 4.2, configuradas para criarem 1, 10, 50, 100, 500 e 1000 tarefas. A quantidade de bytes lidos ou escritos em cada operação foi de 10 KBytes, e usou-se arquivos de 2 MBytes (total de 205 operações). Foi usado um Grupo de Servidores para Aplicações composto por 2 Servidores de Arquivos para Aplicações (jureia do LCA e baiacu do DSEE) e um MPVC composto por 15 computadores (5 de cada laboratório), considerando somente aqueles usados como Módulos dos Clientes do sistema de arquivos. Os testes foram divididos em duas etapas. Primeiramente, eles foram realizados sem usar cache nos Módulos dos Clientes. Posteriormente, eles foram repetidos nas mesmas condições, mas usando cache de forma intensiva. Este processo foi realizado 25 vezes. A Tabela 4.7 mostra de forma resumida as condições dos testes realizados.

Uso de Cache	Servidores	Colaboradores	Números de Tarefas
Não	2	15	1 a 1000
Sim	2	15	1 a 1000

Tabela 4.7: Condições dos testes de uso de cache nos Módulos dos Clientes

Os resultados obtidos são mostrados nas Tabelas 4.8, 4.9, 4.10 e 4.11. No caso dos testes sem o uso do cache nos Módulos dos Clientes (Tabelas 4.8 e 4.9), os dados incluem o tempo médio que cada operação de leitura e escrita levou para ser completada (em milisegundos), e o tempo médio total que cada tarefa levou para concluir o processamento do arquivo (em segundos). Na segunda etapa dos testes, também foi realizada uma comparação dos resultados obtidos nas duas etapas, ressaltando quantas vezes o processamento ficou mais rápido usando o cache nos Módulos dos Clientes (Tabelas 4.10 e 4.11). Para isso, foi dividido o tempo total do processamento de arquivo sem o uso de cache nos Módulos dos Clientes pelo tempo que levou usando este recurso. Para facilitar a compreensão destes resultados, as aplicações foram chamadas de *readX* e *writeX*, onde X é o número de tarefas que elas criaram.

Os resultados são mostrados de forma gráfica na Figura 4.3.

Aplicação	Tempo médio por operação (ms)	Tempo total (seg)
read1	31,16	6,63
read10	35,01	7,35
read50	39,56	8,31
read100	45,89	9,64
read500	54,61	11,47
read1000	66,62	13,91

Tabela 4.8: Resultados dos testes de leitura sem uso de cache nos Módulos dos Clientes (2 servidores e 15 colaboradores)

Aplicação	Tempo médio por operação (ms)	Tempo total (seg)
write1	36,27	7,62
write10	40,26	8,45
write50	45,49	9,55
write100	52,77	11,08
write500	62,81	13,19
write1000	76,62	16,09

Tabela 4.9: Resultados dos testes de escrita sem uso de cache nos Módulos dos Clientes (2 servidores e 15 colaboradores)

Aplicação	Tempo médio (ms)	Tempo total (seg)	Vezes mais rápido
read1	1,45	3,31	2,00
read10	1,60	3,34	2,20
read50	1,81	3,38	2,46
read100	2,09	3,44	2,80
read500	2,49	3,53	3,25
read1000	3,04	3,64	3,82

Tabela 4.10: Resultados dos testes de leitura com uso de cache nos Módulos dos Clientes (2 servidores e 15 colaboradores)

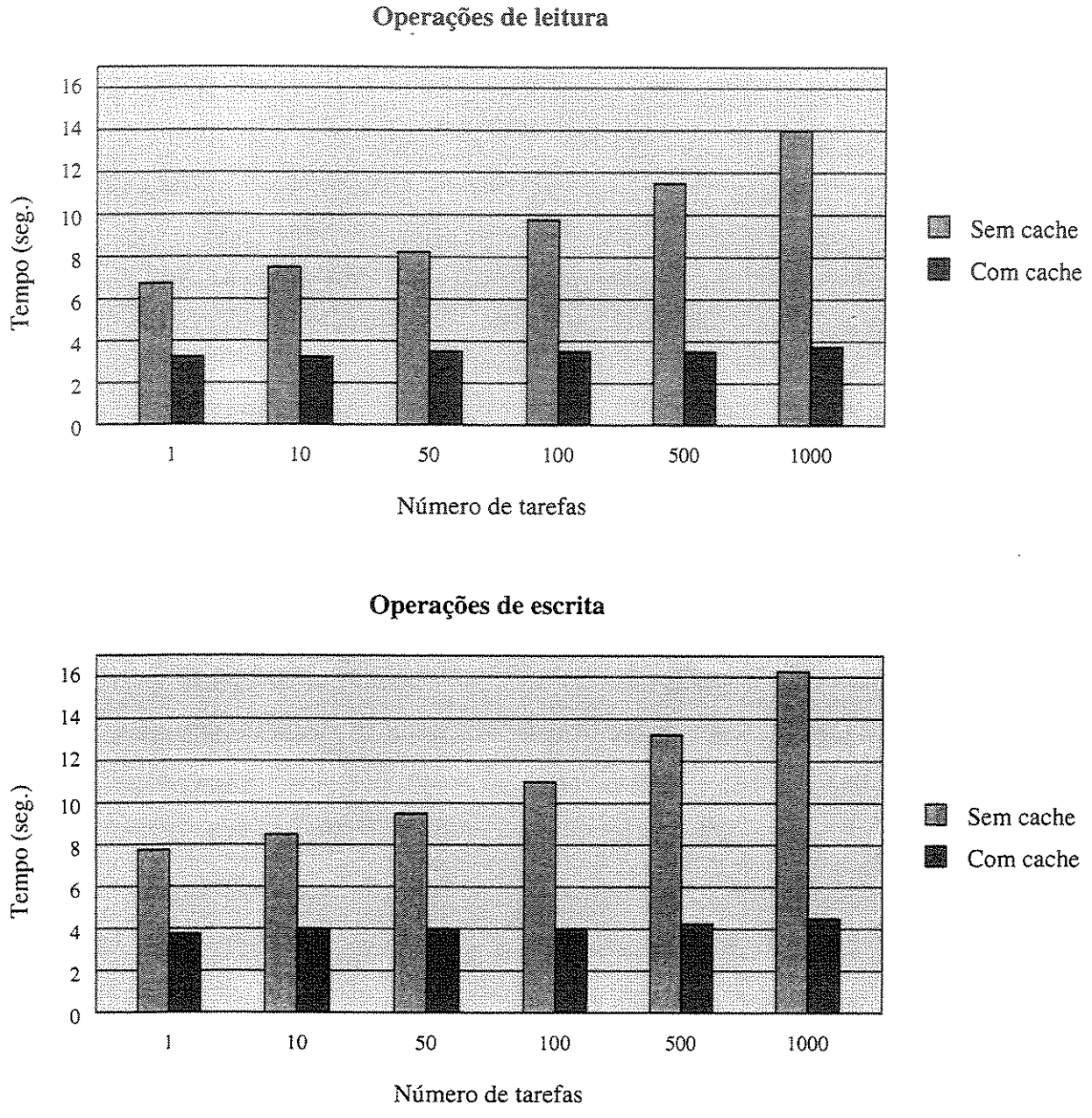

Figura 4.3: Influência do cache nos tempos de leitura e escrita de 15 trabalhadores em 2 servidores

Aplicação	Tempo médio (ms)	Tempo total (seg)	Vezes mais rápido
write1	1,69	3,85	1,98
write10	1,88	3,89	2,17
write50	2,13	3,95	2,42
write100	2,47	4,02	2,76
write500	2,93	4,12	3,20
write1000	3,58	4,25	3,79

Tabela 4.11: Resultados dos testes de escrita com uso de cache nos Módulos dos Clientes (2 servidores e 15 colaboradores)

Os resultados expostos mostram como a eficiência do sistema de arquivos distribuídos aumentou de 2 até 4 vezes com o uso de cache nos Módulos dos Clientes. Este ganho será maior quanto mais lenta e distribuída geograficamente seja a rede à qual pertencem os computadores do MPVC, devido a que o tempo de comunicação impacta de forma decisiva no tempo total de processamento (os dados da Tabela 4.5 mostram isso).

Um dado importante a ser mostrado é que se comparamos os tempos médios gastos no processamento de cada operação de leitura ou escrita sem o uso de cache e com o uso dele, vemos aumentos da velocidade em mais de 20 vezes (variam entre 20 e 25 vezes da Tabela 4.8 para a 4.10 e da Tabela 4.9 para a 4.11). No entanto, as diferenças nos tempos que levaram o processamento do arquivo todo são mais discretas (não chegam a 4 vezes). Esta diferença é devido a que, mesmo usando cache nos Módulos dos Clientes, é necessário ler (ou escrever) o arquivo a partir de (ou para) um servidor (na seção 2.10.3 encontramos uma explicação mais detalhada de como esta operação é realizada). Segundo os dados das Tabelas 4.4 e 4.5, o envio de um arquivo de 2 MBytes, mesmo usando os recursos de compressão de informação, leva em torno de 3 segundos para ser concluído. Portanto, o tempo médio por operação se refere apenas às leituras ou escritas nos caches locais.

Outro aspecto importante a ressaltar nestes testes é que à medida que aumenta o número de tarefas, aumentam os ganhos obtidos com o uso intensivo de cache nos Módulos dos Clientes. Este resultado foi o esperado. No caso dos testes sem o uso de cache, o aumento do número de tarefas faz com que aumente a carga dos servidores (nos testes de escalabilidade veremos como este impacto, mesmo que relativamente pequeno, existe), já que o número de pedidos a serem atendidos pelos mesmos aumenta. No entanto, nos testes com o uso de cache, o impacto do aumento do número de tarefas na carga dos servidores é menor. Como cada Módulo do Cliente mantém cópia dos arquivos usados pelas tarefas no seu computador, ele pode resolver os pedidos das mesmas localmente, comunicando-se com o servidor somente quando necessário (para obter novos blocos de dados do arquivo).

4.4.2 Escalabilidade

Para entender como o sistema reage ao aumento de carga, também foram usadas as aplicações acima citadas, variando o número de tarefas criadas e a quantidade de servidores usados. Os testes foram divididos em 3 etapas, usando-se Grupos de Servidores para Apli-

cações com 1 (jureia do LCA), 2 (jureia do LCA e baiacu do DSEE) ou 3 (jureia do LCA, baiacu do DSEE e isolda do SIFEEC) Servidores de Arquivos para Aplicações. Para verificar como o sistema se comporta quando a carga de trabalho aumenta, aumentou-se o número de tarefas criadas pelas aplicações e o números de computadores que formam o MPVC.

As aplicações foram configuradas para criar 1, 10, 50, 100, 500 e 1000 tarefas. A quantidade de bytes lidos ou escritos em cada operação foi de 10 KBytes e usou-se arquivos de 2 MBytes. Os testes usaram MPVCs com 15 e 30 computadores (5 e 10 computadores de cada laboratório, respectivamente), considerando somente aqueles usados como Módulos dos Clientes do sistema de arquivos. Os testes se basearam na configuração com uso de cache. Estes testes foram também repetidos 25 vezes para cada configuração usada. A Tabela 4.12 mostra de forma resumida como os testes foram realizados.

Servidores	Colaboradores	Números de Tarefas
1	15	1 a 1000
1	30	1 a 1000
2	15	1 a 1000
2	30	1 a 1000
3	15	1 a 1000
3	30	1 a 1000

Tabela 4.12: Condições dos testes de escalabilidade com uso de cache

A seguir são mostradas as Tabelas 4.13 a 4.28 com os resultados obtidos nos testes, que incluem tempo médio (em milisegundos) que cada operação de leitura e escrita levou para ser processada e do tempo médio total (em segundos) que cada tarefa levou para concluir o processamento do arquivo. Os testes também foram divididos tendo em conta o número de servidores e colaboradores usados no MPVC.

Aplicação	Tempo médio (ms)	Tempo total (s)
read1	1,47	3,31
read10	1,63	3,34
read50	1,84	3,39
read100	2,14	3,45
read500	2,55	3,54
read1000	3,11	3,65

Tabela 4.13: Resultado dos testes de escalabilidade (leitura) para MPVC formado por 1 servidor e 15 colaboradores

Os dados mostrados nas Tabelas 4.21 e 4.22, e nas Tabelas 4.10 e 4.11, respectivamente, mesmo sendo realizados com os mesmos parâmetros, não são os mesmos. A razão para as pequenas discrepâncias é que cada grupo de testes foi realizado em dias diferentes e sob distintas condições de carga na rede.

Aplicação	Tempo médio (ms)	Tempo total (s)
write1	1,68	3,84
write10	1,86	3,88
write50	2,12	3,96
write100	2,47	4,03
write500	2,95	4,11
write1000	3,60	4,24

Tabela 4.14: Resultado dos testes de escalabilidade (escrita) para MPVC formado por 1 servidor e 15 colaboradores

Aplicação	Tempo médio (ms)	Tempo total (s)
read1	1,48	3,32
read10	1,62	3,33
read50	1,74	3,37
read100	1,87	3,40
read500	2,37	3,50
read1000	2,59	3,55

Tabela 4.15: Resultado dos testes de escalabilidade (leitura) para MPVC formado por 1 servidor e 30 colaboradores

Aplicação	Tempo médio (ms)	Tempo total (s)
write1	1,68	3,84
write10	1,84	3,89
write50	2,01	3,92
write100	2,14	3,95
write500	2,73	4,07
write1000	2,99	4,13

Tabela 4.16: Resultado dos testes de escalabilidade (escrita) para MPVC formado por 1 servidor e 30 colaboradores

Aplicação	Tempo médio (ms)	Tempo total (s)
read1	1,47	3,31
read10	1,63	3,34
read50	1,79	3,38
read100	2,01	3,42
read500	2,46	3,52
read1000	2,85	3,60

Tabela 4.17: Resultado dos testes de escalabilidade (leitura) para MPVC formado por 2 servidores e 15 colaboradores

Aplicação	Tempo médio (ms)	Tempo total (s)
write1	1,67	3,85
write10	1,85	3,89
write50	2,07	3,93
write100	2,31	3,99
write500	2,84	4,09
write1000	3,30	4,21

Tabela 4.18: Resultado dos testes de escalabilidade (escrita) para MPVC formado por 2 servidores e 15 colaboradores

Aplicação	Tempo médio (ms)	Tempo total (s)
write1	1,67	3,85
write10	1,85	3,89
write50	2,07	3,93
write100	2,31	3,99
write500	2,84	4,09
write1000	3,30	4,21

Tabela 4.19: Resultado dos testes de escalabilidade (escrita) para MPVC formado por 2 servidores e 15 colaboradores

Aplicação	Tempo médio (ms)	Tempo total (s)
write1	1,67	3,85
write10	1,85	3,89
write50	2,07	3,93
write100	2,31	3,99
write500	2,84	4,09
write1000	3,30	4,21

Tabela 4.20: Resultado dos testes de escalabilidade (escrita) para MPVC formado por 2 servidores e 15 colaboradores

Aplicação	Tempo médio (ms)	Tempo total (s)
write1	1,67	3,85
write10	1,85	3,89
write50	2,07	3,93
write100	2,31	3,99
write500	2,84	4,09
write1000	3,30	4,21

Tabela 4.21: Resultado dos testes de escalabilidade (escrita) para MPVC formado por 2 servidores e 15 colaboradores

Aplicação	Tempo médio (ms)	Tempo total (s)
write1	1,67	3,85
write10	1,85	3,89
write50	2,07	3,93
write100	2,31	3,99
write500	2,84	4,09
write1000	3,30	4,21

Tabela 4.22: Resultado dos testes de escalabilidade (escrita) para MPVC formado por 2 servidores e 15 colaboradores

Aplicação	Tempo médio (ms)	Tempo total (s)
read1	1,46	3,31
read10	1,57	3,33
read50	1,72	3,36
read100	1,81	3,38
read500	2,26	3,48
read1000	2,51	3,53

Tabela 4.23: Resultado dos testes de escalabilidade (leitura) para MPVC formado por 2 servidores e 30 colaboradores

Aplicação	Tempo médio (ms)	Tempo total (s)
write1	1,68	3,86
write10	1,77	3,87
write50	1,97	3,91
write100	2,09	3,94
write500	2,61	4,05
write1000	2,88	4,11

Tabela 4.24: Resultado dos testes de escalabilidade (escrita) para MPVC formado por 2 servidores e 30 colaboradores

Aplicação	Tempo médio (ms)	Tempo total (s)
read1	1,47	3,31
read10	1,60	3,34
read50	1,73	3,36
read100	1,84	3,39
read500	2,31	3,48
read1000	2,55	3,54

Tabela 4.25: Resultado dos testes de escalabilidade (leitura) para MPVC formado por 3 servidores e 15 colaboradores

Aplicação	Tempo médio (ms)	Tempo total (s)
write1	1,68	3,85
write10	1,85	3,89
write50	2,09	3,94
write100	2,42	4,01
write500	2,90	4,10
write1000	3,52	4,23

Tabela 4.26: Resultado dos testes de escalabilidade (escrita) para MPVC formado por 3 servidores e 15 colaboradores

Aplicação	Tempo médio (ms)	Tempo total (s)
read1	1,46	3,31
read10	1,54	3,33
read50	1,67	3,36
read100	1,74	3,37
read500	2,08	3,44
read1000	2,34	3,50

Tabela 4.27: Resultado dos testes de escalabilidade (leitura) para MPVC formado por 3 servidores e 30 colaboradores

Aplicação	Tempo médio (ms)	Tempo total (s)
write1	1,67	3,85
write10	1,78	3,88
write50	1,96	3,94
write100	2,12	3,96
write500	2,69	4,06
write1000	2,96	4,12

Tabela 4.28: Resultado dos testes de escalabilidade (escrita) para MPVC formado por 3 servidores e 30 colaboradores

Os resultados também são mostrados de forma gráfica nas Figuras 4.4 e 4.5.

A principal conclusão destes testes de escalabilidade é que com o aumento do número de tarefas, também aumenta o tempo que uma tarefa leva para concluir seu processamento, mas sem causar um impacto significativo no desempenho do sistema de arquivos implementado. A Tabela 4.29 mostra que o aumento de 1 para 1000 no número de tarefas ocasionou um impacto entre aproximadamente 5 e 11% no tempo total de processamento do arquivo.

Servidores	Colaboradores	Leitura (%)	Escrita (%)
1	15	+10,27 (Tabela 4.13)	+10,42 (Tabela 4.14)
2	15	+8,76 (Tabela 4.21)	+9,35 (Tabela 4.22)
3	15	+6,95 (Tabela 4.25)	+9,87 (Tabela 4.26)
1	30	+6,93 (Tabela 4.15)	+7,55 (Tabela 4.16)
2	30	+6,65 (Tabela 4.23)	+6,48 (Tabela 4.24)
3	30	+5,74 (Tabela 4.27)	+7,01 (Tabela 4.28)

Tabela 4.29: Impacto do aumento do número de tarefas de 1 para 1000 no tempo total de processamento do arquivo

Além disso, vemos que o aumento da quantidade de servidores usados e/ou do número de computadores que formam o MPVC também diminuiu o impacto que o aumento do número de tarefas ocasiona no desempenho do sistema. Com o aumento do número de computadores que formam o MPVC obtém-se melhorias discretas nos resultados nas operações de leitura e escrita, já que cada cache local passa a ser usado por um número menor de tarefas. A Tabela 4.30 mostra como o tempo final de processamento do arquivo diminuiu com o aumento do número de computadores no MPVC de 15 para 30, com aplicações que criaram 1000 tarefas.

Servidores	Colaboradores	Leitura (%)	Escrita (%)
1	15 -> 30	-2,74 (Tabelas 4.13 e 4.15)	-2,59 (Tabelas 4.14 e 4.16)
2	15 -> 30	-1,94 (Tabelas 4.21 e 4.23)	-2,38 (Tabelas 4.22 e 4.24)
3	15 -> 30	-1,13 (Tabelas 4.25 e 4.27)	-2,60 (Tabelas 4.26 e 4.28)

Tabela 4.30: Impacto do aumento do número de computadores no MPVC de 15 para 30 no tempo total de processamento do arquivo por 1000 tarefas

Por último, a Tabela 4.31 mostra que, com o aumento do número de servidores em um grupo, o tempo das operações de leitura diminuiu, o que também provocou uma melhoria discreta no tempo que cada tarefa levou para concluir o processamento do arquivo completo. Isto é devido ao mecanismo de balanceamento de carga entre os servidores em um grupo, explicado na seção 2.8.1. Já nas operações de escrita o comportamento foi diferente. O tempo das operações de escrita usando o grupo de 2 servidores diminuiu discretamente se comparado com os resultados obtidos usando 1 servidor só. No entanto, este tempo aumentou ligeiramente quando foram usados 3 servidores. Este atraso é devido ao mecanismo

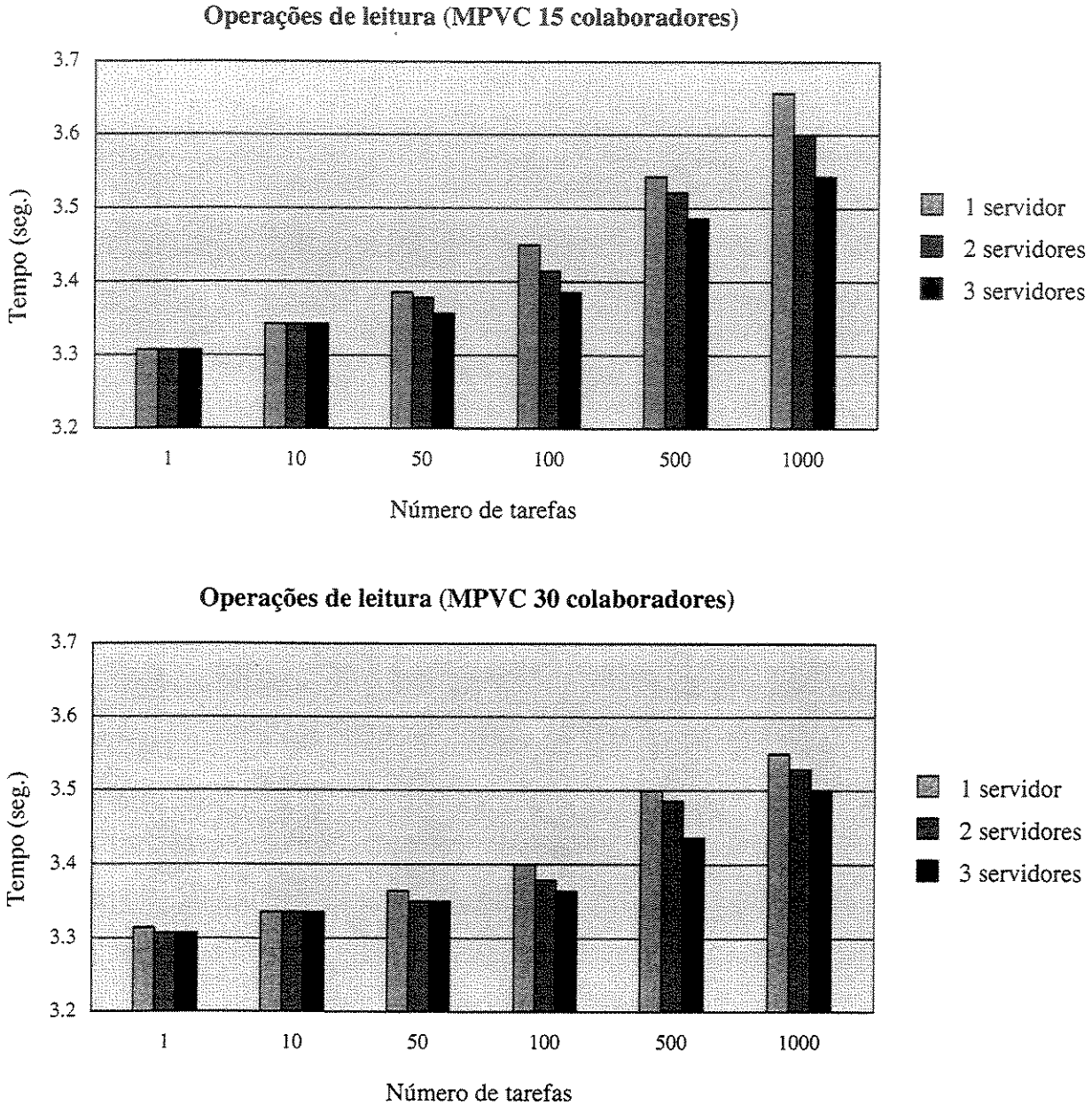

Figura 4.4: Testes de escalabilidade nas operações de leitura

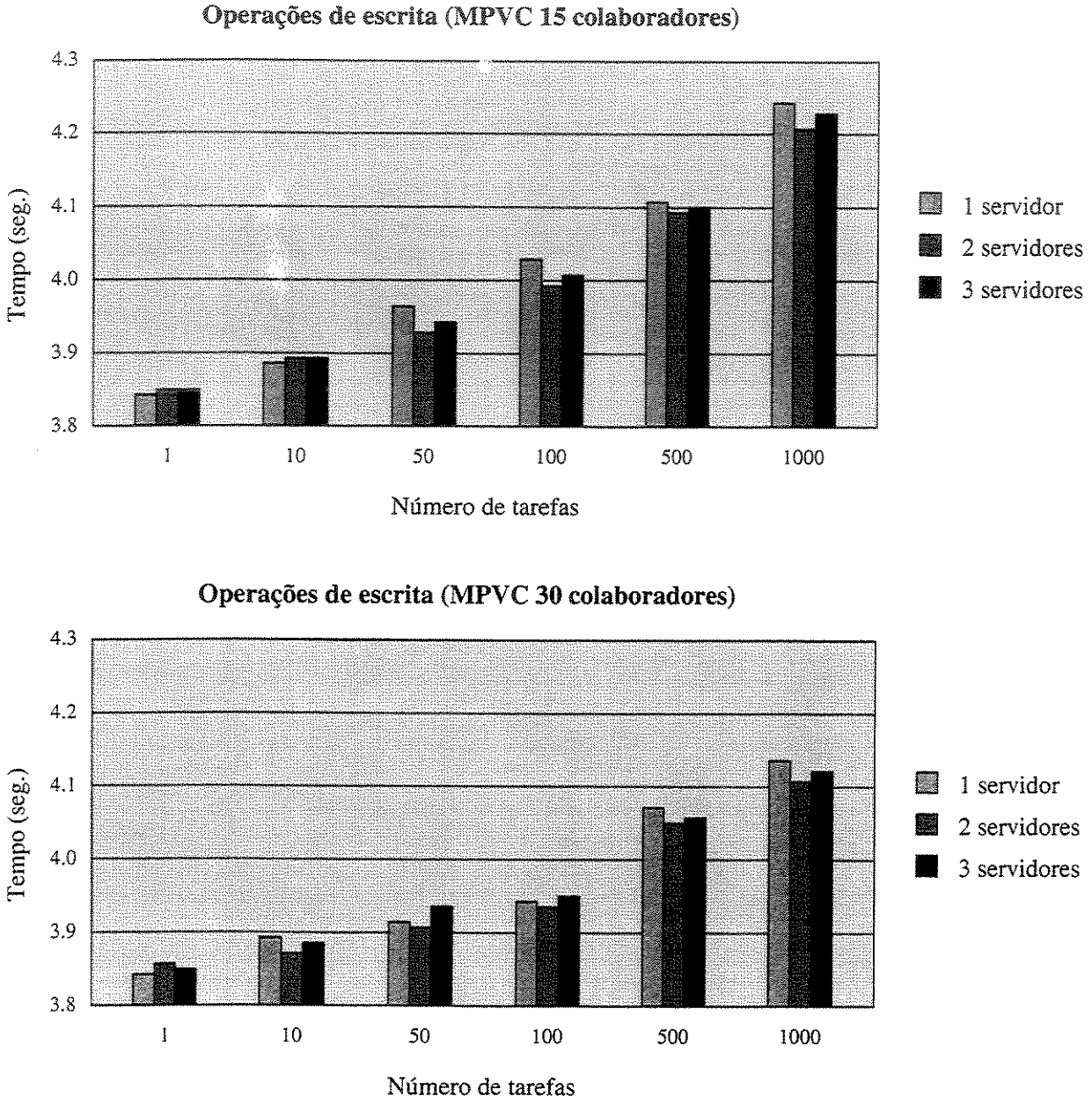

Figura 4.5: Testes de escalabilidade nas operações de escrita

de replicação implementado, que mantém as diferentes réplicas dos servidores atualizadas e necessita de mais comunicações entre seus componentes quando é realizada uma operação de escrita. Além disso, o mecanismo de conciliação foi disparado várias vezes (para grupos formados por 2 servidores foi disparado em média 1 vez e para grupos formados por 3 servidores foi disparado em média 2 vezes), aumentando também o tempo de processamento do arquivo. Ambos os mecanismos foram explicados na seção 2.7.3. Na Tabela 4.30 são comparadas as aplicações que criaram 1000 tarefas, calculando em quanto diminuiu o tempo que levou o processamento do arquivo com grupos de 2 e 3 servidores, se comparado com o tempo que levou com grupos de 1 e 2 servidores, respectivamente.

Colaboradores	Servidores	Leitura (%)	Escrita (%)
15	1 -> 2	-1,37 (Tabelas 4.13 e 4.21)	-0,71 (Tabelas 4.14 e 4.22)
15	2 -> 3	-1,67 (Tabelas 4.21 e 4.25)	+0,48 (Tabelas 4.22 e 4.26)
30	1 -> 2	-0,56 (Tabelas 4.15 e 4.23)	-0,48 (Tabelas 4.16 e 4.24)
30	2 -> 3	-0,85 (Tabelas 4.23 e 4.27)	-0,24 (Tabelas 4.24 e 4.28)

Tabela 4.31: Impacto do aumento do número de servidores em um grupo no tempo total de processamento do arquivo por 1000 tarefas

Por último, deve-se destacar que os testes realizados usando número de tarefas acima de 1000 não foram satisfatórios. Para requisitos muito altos de memória, a máquina virtual Java não conseguiu executar os diferentes componentes do sistema de arquivos (principalmente os Servidores de Arquivos para Aplicações), abortando sua execução (ver seção 5.2).

4.4.3 Tolerância a Falhas

Com o objetivo de determinar o grau de confiabilidade e eficiência do mecanismo de detecção e recuperação de falhas implementado, foram realizados testes para saber como o sistema reage a possíveis falhas. Estes testes foram realizados nas mesmas condições dos testes de escalabilidade (Tabela 4.12), usando-se somente a aplicação de leitura. As aplicações de escrita não foram usadas, já que os resultados obtidos seriam os mesmos. Foram provocadas, de forma intencional, falhas no sistema.

O primeiro teste foi aquele onde o Servidor de Arquivos Central foi desligado. Neste caso, os Grupos de Servidores para Aplicações simplesmente pararam de enviar dados para este servidor. Quando ele foi reiniciado, registrou-se novamente no Resolvedor de Endereços e todas as atualizações pendentes lhe foram enviadas. O sistema continuou trabalhando normalmente. Não houve variações de tempos significativas.

No segundo teste foi desligado um dos servidores de um Grupo de Servidores para Aplicações. O sistema reconfigurou-se automaticamente e todos as tarefas que estavam usando o servidor que caiu começaram a utilizar o outro servidor disponível no grupo ou o Servidor de Arquivos Central, para grupos com um servidor só. As Tabelas 4.32 a 4.37 mostram os resultados obtidos nos testes realizados. Elas incluem os tempos totais médios

(em segundos) que cada tarefa levou para concluir o processamento sem falhas e com a falha provocada (as medidas foram calculadas após 25 execuções dos aplicativos de teste). Também foi calculada a diferença entre os tempos. Os testes foram divididos tendo em conta o número de servidores e colaboradores usados no MPVC.

Nestes testes tem-se alguns resultados que não são os mesmos daqueles obtidos em testes anteriores realizados com os mesmos parâmetros. A explicação novamente é que cada grupo de testes foi realizado em dias diferentes, sob novas condições de carga na rede e nos computadores participantes.

Aplicação	Tempo médio (seg)		Diferença (seg)	Diferença (%)
	Sem falhas	Com 1 falha		
read1	3,31	4,17	+0,86	+25,98
read10	3,34	4,27	+0,93	+27,84
read50	3,39	4,71	+1,32	+38,93
read100	3,45	4,91	+1,46	+42,32
read500	3,54	5,27	+1,73	+48,87
read1000	3,65	5,76	+2,11	+57,81

Tabela 4.32: Resultado dos testes de tolerância a falhas para MPVC formado por 1 servidor e 15 colaboradores

Aplicação	Tempo médio (seg)		Diferença (seg)	Diferença (%)
	Sem falhas	Com 1 falha		
read1	3,32	4,17	+0,85	+25,60
read10	3,33	4,26	+0,93	+27,93
read50	3,37	4,49	+1,12	+33,23
read100	3,40	4,77	+1,37	+40,29
read500	3,51	5,16	+1,65	+47,01
read1000	3,56	5,47	+1,91	+53,65

Tabela 4.33: Resultado dos testes de tolerância a falhas para MPVC formado por 1 servidor e 30 colaboradores

Os resultados destes testes também são mostrados de forma gráfica nas Figuras 4.6, 4.7 e 4.8.

O sistema de arquivos distribuído implementado mostrou suas possibilidades de se recuperar automaticamente ante uma possível falha, embora o impacto na eficiência do sistema fosse relativamente alto. Em todas as condições dos testes todas as tarefas concluíram o processamento do arquivo e, segundo as tabelas mostradas, o tempo que levou cada tarefa para processá-lo aumentou entre 25 e 58%. Este impacto foi maior à medida que aumentou o número de tarefas. A principal causa deste impacto foi a forma como o mecanismo de reconfiguração automática foi implementado, explicado em detalhes na seção 2.9.

Aplicação	Tempo médio (seg)		Diferença (seg)	Diferença (%)
	Sem falhas	Com 1 falha		
read1	3,31	4,16	+0,85	+25,68
read10	3,34	4,21	+0,87	+26,05
read50	3,38	4,39	+1,01	+29,88
read100	3,42	4,61	+1,19	+34,79
read500	3,52	5,03	+1,51	+42,89
read1000	3,60	5,53	+1,93	+53,61

Tabela 4.34: Resultado dos testes de tolerância a falhas para MPVC formado por 2 servidores e 15 colaboradores

Aplicação	Tempo médio (seg)		Diferença (seg)	Diferença (%)
	Sem falhas	Com 1 falha		
read1	3,31	4,17	+0,86	+25,98
read10	3,33	4,22	+0,89	+26,73
read50	3,36	4,34	+0,98	+29,17
read100	3,38	4,50	+1,12	+33,14
read500	3,48	4,89	+1,41	+40,52
read1000	3,53	5,26	+1,73	+49,01

Tabela 4.35: Resultado dos testes de tolerância a falhas para MPVC formado por 2 servidores e 30 colaboradores

Aplicação	Tempo médio (seg)		Diferença (seg)	Diferença (%)
	Sem falhas	Com 1 falha		
read1	3,31	4,17	+0,86	+25,98
read10	3,34	4,21	+0,87	+26,05
read50	3,36	4,27	+0,91	+27,08
read100	3,39	4,39	+1,00	+29,51
read500	3,48	4,80	+1,32	+37,93
read1000	3,54	5,19	+1,65	+46,61

Tabela 4.36: Resultado dos testes de tolerância a falhas para MPVC formado por 3 servidores e 15 colaboradores

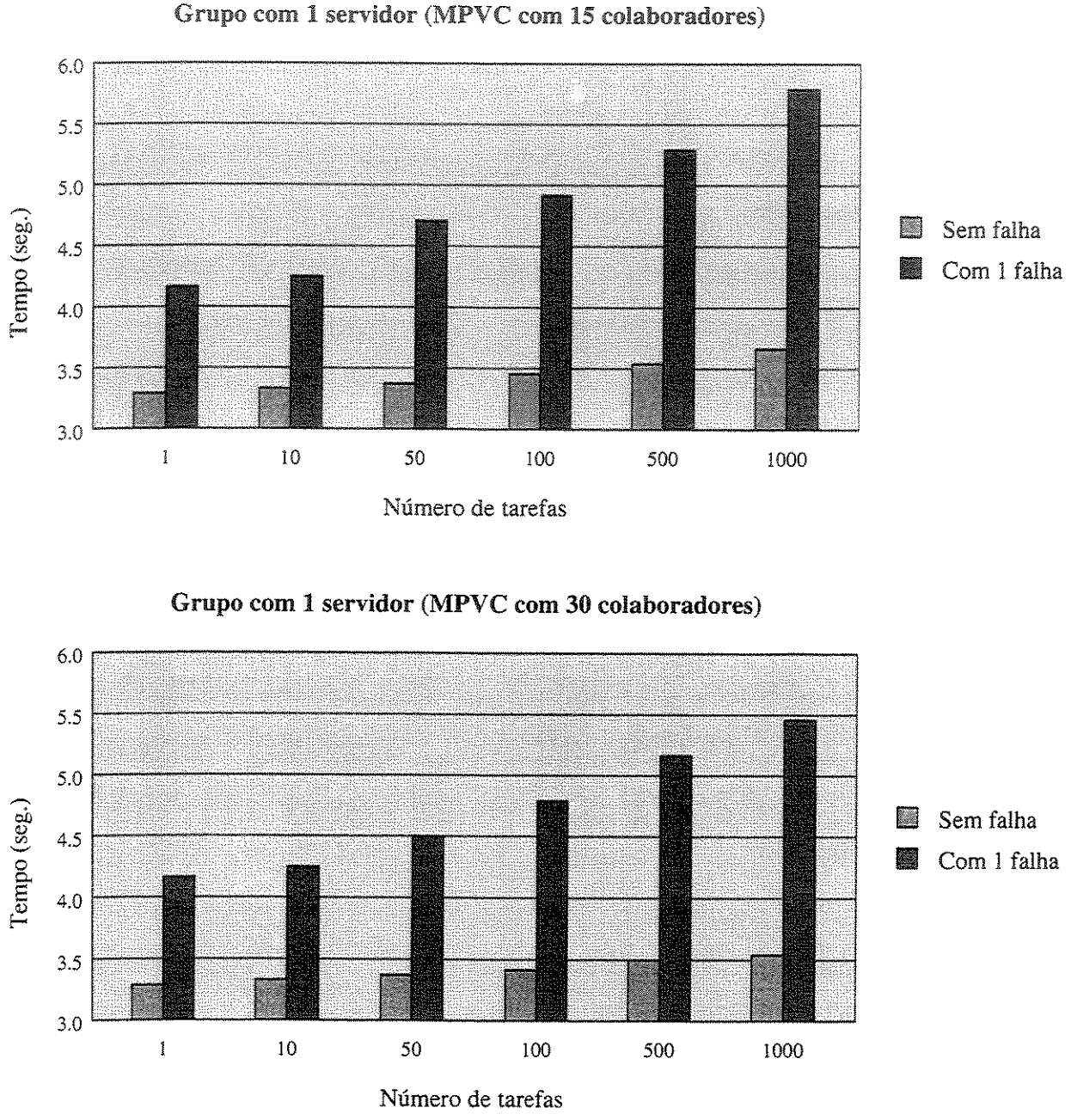

Figura 4.6: Impacto de 1 falha no tempo de leitura de arquivos (1 Servidor de Arquivos para Aplicações e o Servidor Central)

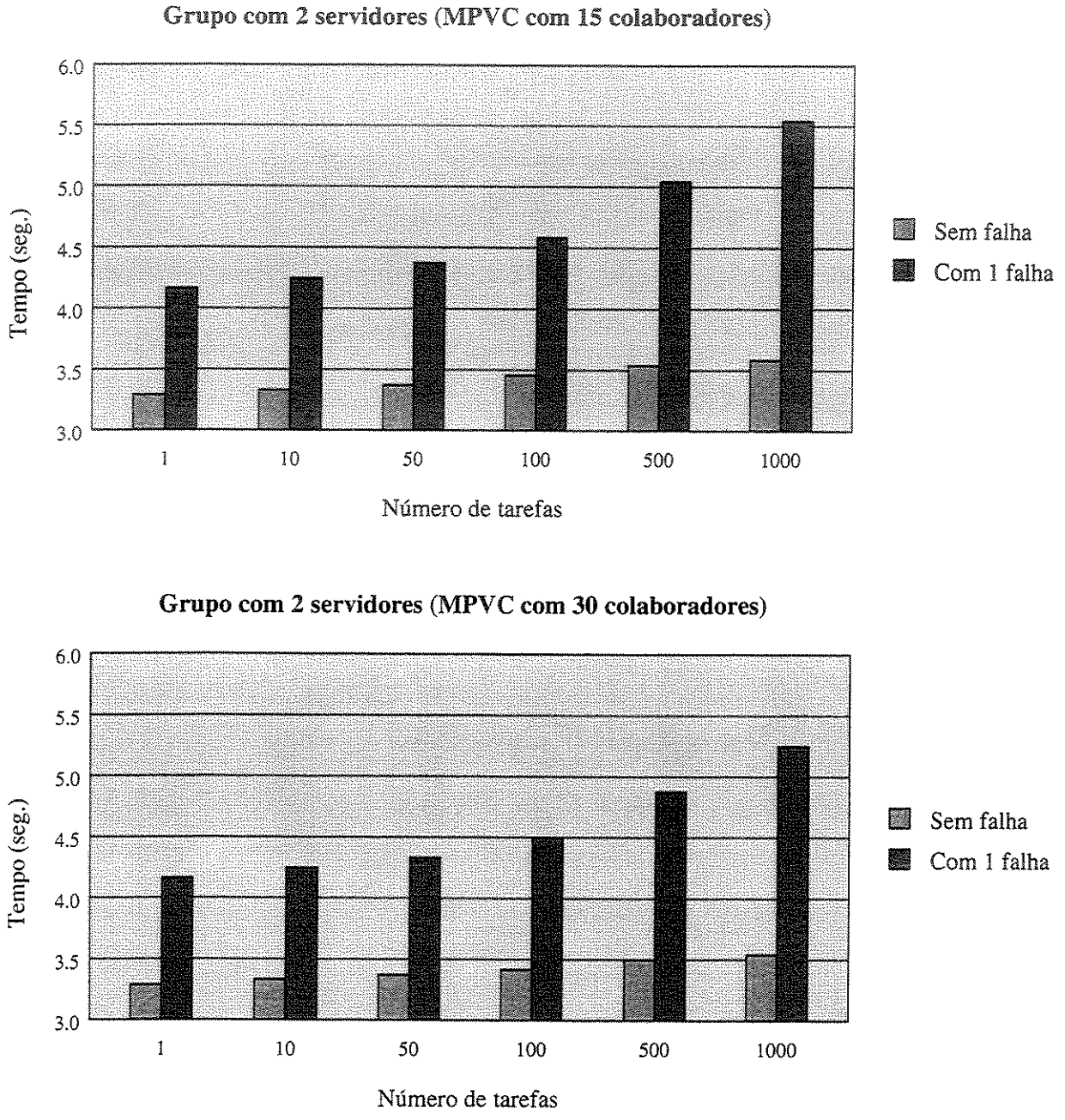

Figura 4.7: Impacto de 1 falha no tempo de leitura de arquivos (2 Servidores de Arquivos para Aplicações e o Servidor Central)

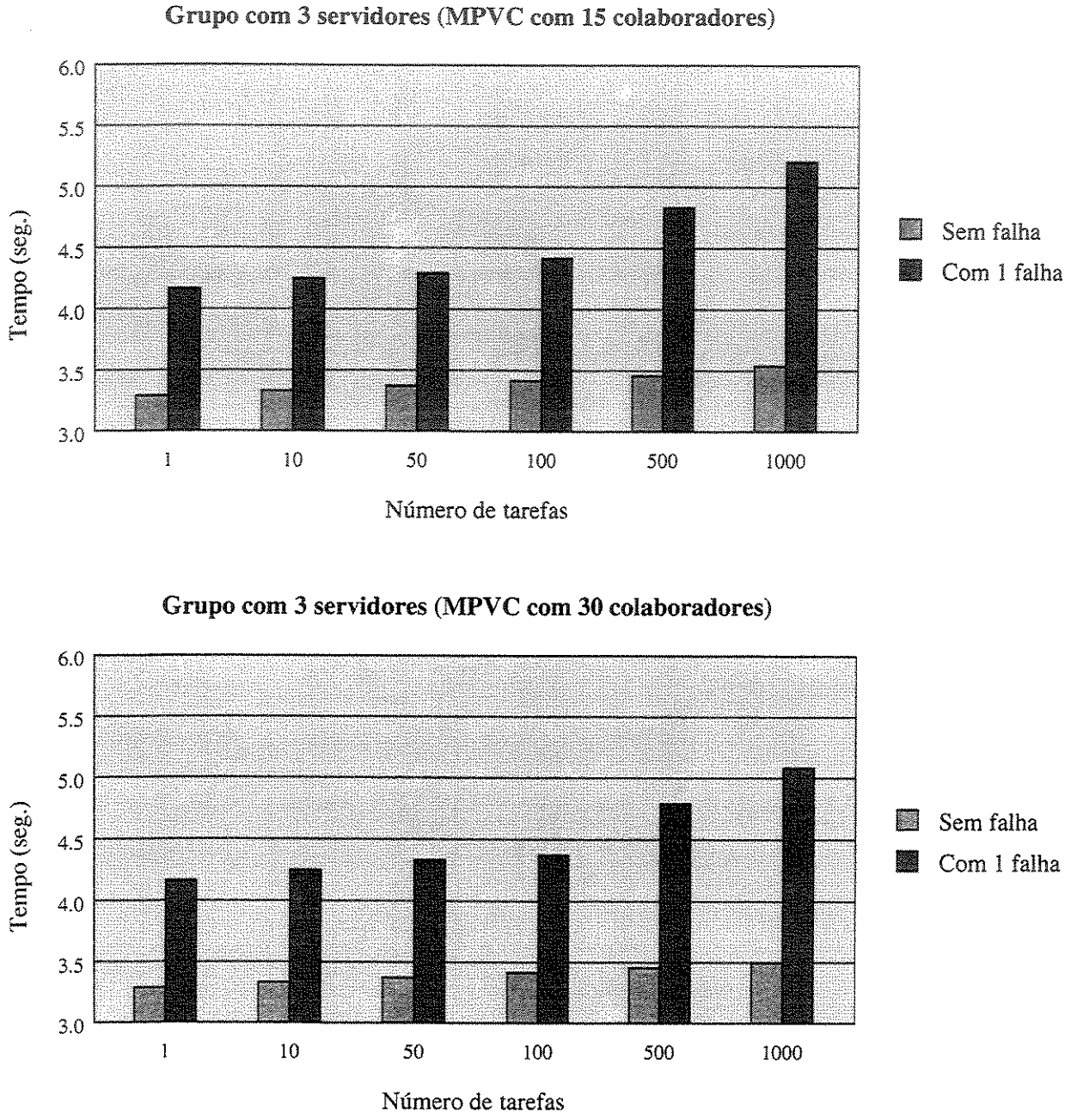

Figura 4.8: Impacto de 1 falha no tempo de leitura de arquivos (3 Servidores de Arquivos para Aplicações e o Servidor Central)

Aplicação	Tempo médio (seg)		Diferença (seg)	Diferença (%)
	Sem falhas	Com 1 falha		
read1	3,31	4,16	+0,85	+25,68
read10	3,33	4,20	+0,87	+26,13
read50	3,36	4,30	+0,94	+27,98
read100	3,37	4,36	+0,99	+29,38
read500	3,44	4,76	+1,32	+38,37
read1000	3,50	5,10	+1,60	+45,71

Tabela 4.37: Resultado dos testes de tolerância a falhas para MPVC formado por 3 servidores e 30 colaboradores

No entanto, analisando mais detalhadamente os resultados obtidos, podemos tirar outras conclusões. A primeira conclusão é que, agrupando os resultados por servidores (Tabelas 4.32, 4.34 e 4.36, e Tabelas 4.33, 4.35 e 4.37), vemos que com o aumento do número de servidores no grupo diminuiu o impacto final no tempo de processamento do arquivo, já que quando um servidor cai, as tarefas que estavam utilizando os seus serviços são repartidas entre um número maior de servidores, aumentando em menor escala a carga dos mesmos.

Já agrupando os resultados por computadores (Tabelas 4.32 e 4.33, Tabelas 4.34 e 4.35, e Tabelas 4.36 e 4.37), vemos que quando aumenta o número de computadores no MPVC o impacto final no processamento do arquivo diminui na maioria dos casos. Isto é devido a que cada Módulo do Cliente recebe um número menor de tarefas para serem executadas no seu computador. Desta forma, estes módulos ficam mais leve (recebem um número menor de pedidos das tarefas), possibilitando uma reconfiguração mais rápida dos mesmos.

Não foram realizados testes provocando duas falhas porque consideramos que o resultado final seria o mesmo, ou seja, o sistema se recuperaria da falha com sucesso, somente demoraria mais tempo para se reconfigurar.

4.5 Testes com Aplicações

Para obter resultados que mostrem que o sistema de arquivos distribuído pode ser usado para resolver problemas reais, foram implementadas duas aplicações em JoiN que utilizam o serviço de arquivos: uma aplicação de busca de números primos grandes e uma aplicação de multiplicação de matrizes.

A aplicação de busca de números primos grandes é uma aplicação simples formada por uma tarefa principal que recebe um intervalo, o divide em um determinado número de subintervalos e cria uma quantidade equivalente de tarefas-filha. Cada uma destas tarefas-filha recebe um subintervalo e o processa, achando todos os números primos contidos nele. Estes números são armazenados em um arquivo que cada uma das tarefas cria e assim que elas concluem o processamento, enviam o nome dos arquivos para a tarefa principal, encarregada de criar um único arquivo de saída, agrupando todos os resultados.

Nestes testes foram usados os parâmetros mostrados a seguir.

Intervalo: 1 até 30.000.000

Extensão de cada subintervalo: 250.000

Número de tarefas: 120

A aplicação de multiplicação de matrizes é uma aplicação um pouco mais complexa. Ela está formada por uma tarefa principal que recebe as duas matrizes a multiplicar e divide a primeira delas em um determinado número de submatrizes (por linhas). Logo após dividir a matriz, a tarefa principal cria o mesmo números de tarefas-filha, enviando-lhes uma das submatrizes criadas e a segunda matriz completa. Cada tarefa-filha multiplica as matrizes recebidas e cria uma nova matriz. Todas estas matrizes resultantes são enviadas à tarefa principal, que cria a matriz final juntando os resultando obtidos nas tarefas-filha.

As duas matrizes a multiplicar são criadas e armazenadas em arquivos antes de executar a aplicação. Estas matrizes podem ser armazenadas em disco por linhas ou por colunas, para tornar mais simples sua manipulação. Além disso, também foram implementadas funções para ler e escrever uma linha ou uma coluna de uma matriz em um arquivo. Estas funções foram usadas para evitar carregar a matriz inteira na memória do computador.

A tarefa principal da aplicação recebe o nome dos arquivos onde as duas matrizes estão armazenadas. A primeira das matrizes deve ter sido armazenada por linhas e a segunda por colunas, e o número de colunas da primeira deve ser igual ao número de linhas da segunda (requisito da multiplicação de matrizes). Os nomes dos arquivos são enviados para as tarefas-filha, além dos números das linhas da primeira matriz que cada uma delas vai usar na multiplicação. Para calcular a matriz resultante, cada tarefa-filha não precisa ler a matriz completa em memória; somente necessita ler a linha da primeira matriz e a coluna da segunda com as quais esta trabalhando. Estas tarefas vão gerando as linhas da matriz resultante, que são armazenadas diretamente em disco (em um arquivo criado pela tarefa principal para armazenar a matriz resultante). Quando todas as tarefas-filha completam o processamento, a matriz resultante está armazenada completa no arquivo criado.

Como todas as tarefas vão escrever em um mesmo arquivo de resultado, parte do tempo seria gasto com resolução de conflitos (colisão de pedidos). No entanto, estas colisões não vão ocorrer devido a que cada tarefa vai criar linhas diferentes da matriz resultante, as quais são obtidas pelo produto de um subconjunto, sempre disjunto, das linhas na primeira matriz passadas a cada tarefa. Isto faz com que cada tarefa escreva no arquivo resultado da multiplicação linhas diferentes da matriz resultante, e conseqüentemente, não existiram colisões entre os pedidos. Além disso, a aplicação cria um *buffer* onde acumula o resultado de várias multiplicação, mandando-os de uma vez só para a tarefa principal. Desta forma diminui o tráfego na rede, tornando a multiplicação mais rápida.

Os parâmetros usados nestes testes são mostrados a seguir.

Tamanho das matrizes: 1024 x 1024

Número de tarefas: 128

Os testes realizados com estas aplicações tiveram como objetivo determinar se o uso do sistema de arquivos impõe uma sobrecarga muito grande ao desempenho das mesmas. Os acessos a disco são mais lentos que o uso da memória e é por isso que sempre tem-se uma sobrecarga na utilização de um sistema de arquivos. No entanto, esta sobrecarga deve ser razoável para tornar viável o uso do sistema de arquivos.

Para fazer os testes e determinar a sobrecarga imposta, foram comparados os resultados obtidos com uma versão destas aplicações que não usou arquivos (intervalos e matrizes estão embutidas no programa) e uma versão usando o serviço de arquivos SAD_GSA. Esta segunda versão usou todos os mecanismos implementados pelo SAD_GSA, como a replicação de servidores, balanceamento de carga, reconfiguração automática e uso de cache nos Módulos dos Clientes. Foi usado um Grupo de Servidores para Aplicações formado por 2 Servidores de Arquivos para Aplicações (jureia do LCA e baiacu do DSEE) e o número de computadores que formavam o MPVC foi de 15 computadores (5 computadores de cada laboratório, respectivamente), considerando somente aqueles usados como Módulos dos Clientes do sistema de arquivos. Estes testes foram repetidos 25 vezes. Os resultados (em segundos) são mostrados na Tabela 4.38.

Aplicação	Tempo médio (seg)		Sobrecarga (%)
	Sem SA	Com SAD_GSA	
Números primos	29,55	40,64	+37,53
Multiplicação de matrizes	186,47	267,99	+43,72

Tabela 4.38: Resultado dos testes de sobrecarga no uso do sistema de arquivos para MPVC com 15 colaboradores e 2 Servidores de Arquivos para Aplicações

Os resultados indicam que o sistema de arquivos impõe uma sobrecarga abaixo de 50% às aplicações que o usam, se comparado ao tempo que leva o processamento com uma versão similar que não use acesso a disco. Se consideramos as vantagens que um sistema de arquivos pode oferecer, esta sobrecarga não é muito grande e, portanto, viável para aplicações que usem o SAD_GSA.

4.6 Sumário

Para verificar se o sistema de arquivos implementado satisfaz de forma eficiente as necessidades de um Computador Maciçamente Paralelo Virtual formado por computadores conectados pela Internet (como o JOIN) e para determinar com maior exatidão a influência do sistema de arquivos na performance geral de computador virtual, foram realizados vários testes, divididos em 3 grupos.

Os testes iniciais foram feitos para determinar se o sistema de arquivos satisfaz os requisitos desejados, tais como eficiência, escalabilidade e tolerância a falhas. Os testes de configuração foram usados para determinar parâmetros adequados de operação do sistema. E por último, os testes realizados com aplicações mais complexas permitiram comprovar se o sistema está preparado para que seus recursos sejam usados de forma intensiva para resolver

problemas reais. Os resultados obtidos nestes testes foram satisfatórios. Os resultados mais importantes são resumidos a seguir.

- O tamanho recomendado para os blocos que são transmitidos pela rede variou de 64 KBytes a 256 KBytes.
- Em redes rápidas o uso de mecanismos de compressão não é necessário. Já para redes mais lentas, como é o caso da Internet, e para mensagens iguais ou maiores que 64 KBytes obteve-se melhores resultados usando estes mecanismos.
- O uso de cache nos Módulos dos Clientes aumentou significativamente o desempenho do sistema, minimizando o tempo de resposta às aplicações de 2 até quase 4 vezes.
- Com o aumento do número de tarefas criadas pelas aplicações também aumentaram os ganhos obtidos pelo uso de cache nos Módulos dos Clientes.
- O aumento do número de tarefas nas aplicações que usam o sistema de arquivos não causou um impacto significativo no seu desempenho, mostrando a escalabilidade do mesmo. Aumentando o número de tarefas de 1 até 1000 fez com que o tempo de resposta somente aumentasse de 5 a 11%. Além disso, aumentando o número de computadores no MPVC e/ou o número de servidores nos grupos fez com que este impacto também diminuísse.
- O sistema não conseguiu operar com aplicações que usam a memória de forma intensiva. A máquina virtual Java abortou o processamento quando tentou-se ultrapassar o limite de 1000 tarefas.
- O aumento do número de Servidores de Arquivos para Aplicações em um grupo diminuiu o tempo de resposta das operações de leitura. Para as operações de escrita o resultado não é o mesmo, devido aos mecanismos de replicação de servidores e de manutenção de coerência entre as diferentes réplicas dos dados.
- O sistema recuperou-se automaticamente ante 1 falha provocada (em várias configurações) sem influenciar significativamente na eficiência do sistema. O tempo gasto para se reconfigurar variou aproximadamente entre 25 e 58%. Este impacto é maior a medida que aumenta o número de tarefas, mas diminui quando aumenta o número de servidores que formam os Grupos de Servidores de Arquivos ou o número de computadores no MPVC.
- O sistema de arquivos impôs uma sobrecarga menor que 44% às aplicações que usaram seus recursos.

Capítulo 5

Conclusões

5.1 Principais Resultados

Existem várias propostas para o processamento paralelo usando computadores conectados em rede. Muitas destas propostas utilizam a Internet como um computador virtual, tirando proveito do uso de navegadores para WWW com possibilidade de executar *applets Java*. No entanto, a maioria destes sistemas não trata da necessidade de se ter um sistema de arquivos que garanta a viabilidade e eficiência na execução das aplicações paralelas.

Os sistemas de arquivos encontrados na literatura e em sistemas comerciais não satisfazem as principais exigências de um computador paralelo virtual baseado na Internet. O objetivo principal desta pesquisa foi conceber um sistema de arquivos distribuído que satisfizesse de forma eficiente estas necessidades. Dentre os requisitos que ele tinha que atender destacou-se: portabilidade, tolerância a falhas, consistência, escalabilidade, reconfiguração automática, eficiência, espaço de nomes global, segurança e facilidade de instalação.

Foram propostas e comparadas 3 formas de implementação para o sistema de arquivos desejado. A proposta que resultou ser mais adequada para este tipo de processamento foi o Sistema de Arquivos Distribuído baseado em Grupos de Servidores de Arquivos (SAD_GSA). Esta proposta visa satisfazer as principais exigências de plataformas maciçamente paralelas baseadas na Internet.

A portabilidade foi um dos principais objetivos do sistema. Para isso, foi usado Java, linguagem de programação independente de plataforma. Desta forma, fica mais fácil lidar com as dificuldades inerentes de se trabalhar com computadores de arquiteturas distintas.

Outro ponto muito importante que o sistema de arquivos proposto centrou sua atenção foi na escalabilidade, garantindo assim seu uso em uma plataforma para o processamento maciçamente paralelo. O uso de servidores de arquivos do tipo *stateless* é a primeira característica do SAD_GSA para aumentar sua escalabilidade. Além disso, o sistema proposto tem a capacidade de se adaptar às necessidades das aplicações que estão usando os seus recursos e à quantidade de computadores que participam do MPVC. Os Grupos de Servidores para Aplicações podem aumentar ou diminuir a quantidade de servidores que o formam para melhor atender os pedidos das aplicações. Esta característica, aliada ao mecanismo de balanceamento de carga entre os servidores de um grupo, torna o SAD_GSA

mais escalável.

A disponibilidade e a tolerância a falhas também foram aspectos levados em conta na implementação do sistema de arquivos proposto. Foi implementado um mecanismo de replicação em dois níveis. No primeiro nível encontram-se os Grupos de Servidores para Aplicações e no segundo nível o Servidor de Arquivos Central. Este mecanismo de replicação garante que, caso houver alguma falha no sistema (um servidor cair ou perder sua conexão com o resto do sistema), existam outros pontos onde as tarefas das aplicações possam ter acesso aos arquivos. O uso de servidores de arquivos do tipo *stateless*, além de aumentar a escalabilidade do sistema, também torna o sistema mais tolerantes a falhas.

Para manter as diferentes réplicas atualizadas, o SAD_GSA implementa um protocolo de atualização em cada Grupo de Servidores de Aplicações. Desta forma, qualquer servidor dentro de um grupo pode atender qualquer pedido feito ao grupo. Para o caso de surgir inconsistências entre as diferentes cópias no grupo, foi implementado um processo de reconciliação encarregado de corrigir o problema. Assim, é garantida a consistência entre toda a informação espalhada no sistema.

Além de todos estes mecanismos implementados para garantir que o sistema seja tolerante a falhas, o SAD_GSA também tem a possibilidade de se reconfigurar automaticamente ante elas. Foi implementado um mecanismo totalmente transparente para as tarefas que detecta e corrige possíveis falhas que possam acontecer, sem afetar seu funcionamento.

Todos estes mecanismos aumentam a robustez e eficiência do sistema. Mas o principal mecanismo implementado para diminuir os tempos de resposta do sistema foi o uso intensivo de cache nos Módulos dos Clientes, que também diminui o tráfego na rede e a carga dos servidores. O SAD_GSA fornece um modelo que abrange de forma geral a forma de utilização dos arquivos por aplicações paralelas, garantindo uma semântica de sessão para cada tarefa que usar um determinado arquivo.

O SAD_GSA oferece às aplicações uma notação simples para se referir aos arquivos através de um espaço de nomes global e uniforme. Além disso, esta notação é totalmente transparente e independente de localização. Para tentar obter maiores ganhos em organização e uso dos arquivos, o sistema define uma estrutura de diretórios que deve ser respeitada pelas aplicações que usam os seus serviços. Na estrutura de arquivos definida, cada usuário do sistema tem um diretório próprio, onde ele deve criar novos subdiretórios para cada uma das aplicações que vai executar. Neste diretório são armazenadas todas as informações que a aplicação necessita ou cria na sua execução.

Para garantir a segurança dos arquivos armazenados no sistema de arquivos, foi implementado o mecanismo conhecido como modo de bits de Unix para controlar o acesso aos mesmos. Existem duas permissões de acesso a um arquivo: permissões de leitura e permissões de escrita. Além disso, os acessos a um arquivo ou diretório foram divididos em três tipos: acessos do usuário que criou o arquivo, acessos dos usuários que pertencem ao mesmo grupo do proprietário do arquivo e acessos dos demais usuários. Para cada operação realizada, o sistema analisa se o usuário que fez o pedido tem as permissões requeridas para tal. Caso contrário, a operação não é completada. Desta forma, o sistema detecta de forma simples e rápida acessos não permitidos às informações armazenadas.

Como plataforma de testes do sistema de arquivo proposto foi utilizado JOIN, um

sistema para o processamento maciçamente paralelo baseado na Internet e na ampla disponibilidade de Java como linguagem independente de plataforma.

Vários testes foram feitos para verificar a validade das idéias propostas e se o sistema de arquivos implementado satisfaz as especificações para as quais foi criado. Estes testes foram divididos em 3 grupos principais. O primeiro grupo, formado por testes de configuração, foi usado para obter os melhores valores para alguns parâmetros do sistema, como por exemplo o tamanho dos blocos que são enviados pela rede. O segundo grupo teve como objetivo testar se o sistema de arquivos é escalável e tolerante a falhas, assim como as vantagem do uso intensivo de cache. O último grupo constou de aplicações reais (aplicação de busca de números primos e aplicação de multiplicação de matrizes de grandes dimensões) que fazem uso do sistema de arquivos.

Os resultados obtidos demonstraram que o sistema de arquivos proposto, sob as condições de testes usadas (MPVC formado por até 35 colaboradores e aplicações criando até 1000 tarefas), provê um serviço robusto e eficiente para um ambiente de processamento paralelo heterogêneo baseado na Internet.

No total, foram usados 35 computadores, 13 do LCA, 11 do DSEE e 11 do SIFEEC. Os testes foram realizados em horários de pouca carga de trabalho para evitar que fatores externos influenciassem nos resultados. Além disso, os computadores mais rápidos em cada laboratório foram escolhidos para executar os principais componentes do sistema de arquivos, tais como o Servidor de Arquivos Central, o Resolvedor de Endereços e os Servidores de Arquivos para Aplicações.

5.2 Trabalhos Futuros

No sistema de arquivos proposto há vários pontos que ainda podem ser trabalhados para melhorar sua eficiência. Alguns deles são mostrados a seguir.

- Definir e implementar classes diferentes de arquivos (por exemplo arquivos temporários, arquivos perenes, arquivos grandes) que permitam dar um tratamento diferenciado a cada um deles, o que poderia levar a um aumento da eficiência do sistema.

- Implementar a reconfiguração automática e outros mecanismos do sistema de arquivos usando o recurso de eventos que novas versões de JOIN oferecem. Estes mecanismos não existiam em JOIN quando implementou-se esta versão do sistema de arquivos. Fazendo uso deste recurso o sistema de arquivos ficaria mais simples, sendo mais fácil a manutenção e inclusão de novas funcionalidades.

- Estudar a possibilidade de se implementar técnicas de cache mais sofisticadas que permitam compartilhar arquivos para leitura e escrita simultaneamente. Além disso, podem ser implementados mecanismos mais eficientes de predição de futuros blocos a serem copiados no cache, que levem em conta fatores como o tamanho dos blocos lidos pelas aplicações, aumentando assim a taxa de acerto de dados no cache.

- Implementar cache de arquivos em memória. A implementação atual somente mantém em memória a lista de arquivos abertos. Implementar um cache que também mantenha em memória dados dos arquivos usados aumentaria ainda mais a eficiência do sistema de arquivos.

- Estudo aprofundado dos problemas de memória encontrados quando se aumenta o número de tarefas (e portanto o uso da memória). Este estudo inclui a forma como a memória é manipulada, o mecanismo de *garbage collector*, dentre outras características importantes da máquina virtual Java, e serviria para evitar problemas futuros.

- Implementar novos utilitários, tais como gerenciador de permissões de acesso de arquivos e diretórios, gerenciador de configurações do sistema e gerenciador de eventos.

- Realizar testes mais abrangentes que utilizem um número maior de computadores conectados em redes mais heterogêneas e mais distribuídas geograficamente.

Apêndice A

Sistemas de Arquivos Distribuídos

Neste apêndice são analisados os principais sistemas de arquivos existentes, destacando as principais vantagens e desvantagens dos mesmos e discutindo sua viabilidade de servir a plataformas maciçamente paralelas baseadas na Internet.

A.1 Sistema de Arquivos de Unix

Unix [MCF92] é provavelmente o sistema operacional de maior aceitação na comunidade científica e acadêmica. As razões desta ampla utilização são muitas, mas uma das mais notáveis é o seu sistema de arquivos [KK90a, BP88], uma das partes mais visíveis em um sistema operacional. Apesar de não ser distribuído, o Sistema de Arquivos de Unix é usado como referência pela maioria dos sistemas de arquivos atuais, devido à sua simplicidade, confiabilidade e alto desempenho. Por esta razão, ele também será revisto aqui.

A base de todo esse bom desempenho é a forma simples e eficiente como são organizados os arquivos. Unix associa a cada arquivo uma pequena tabela, chamada de *i_node*, que contém toda a informação referente ao arquivo (atributos, proteção, tamanho, dentre outras), como mostra a Figura A.1. Além das informações referentes ao arquivo, estes *i_nodes* também contêm as informações dos blocos do disco onde estão armazenados os dados do arquivo. O *i_node* tem 10 endereços de blocos diretos e 3 endereços de blocos indiretos (simples, duplo e triplo). Para arquivos de até 10 blocos são usados somente os endereços diretos. Desta forma, o Sistema de Arquivos de Unix pode localizar a informação armazenada no arquivo imediatamente. Para arquivos maiores que 10 blocos também são usados os endereços indiretos, permitindo que, mesmo para arquivos muito longos, os blocos sejam localizados em no máximo 4 acessos a disco.

Para minimizar a quantidade de acessos a disco e desta forma aumentar o desempenho, o Sistema de Arquivos de Unix implementa um cache de blocos. São mantidos em memória os blocos que estão sendo utilizados pelas aplicações, usando uma política LRU (*Least Recently Used*) para gerenciá-los e implementando algoritmos eficientes para a representação, manipulação e manutenção da consistência entre os dados mantidos neste cache e os que estão armazenados no disco. Desta forma, qualquer mudança feita em um arquivo por um processo é imediatamente vista pelos processos restantes. Esta semântica é conhecida

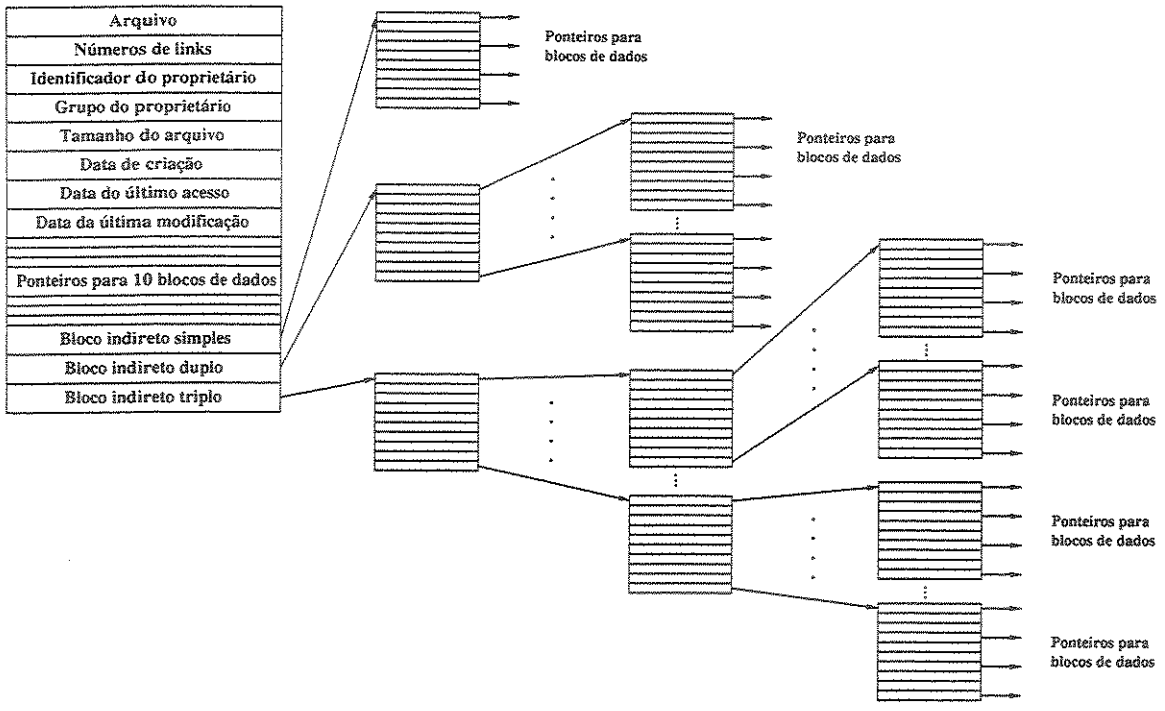

Figura A.1: Organização dos arquivos em Unix (*i_nodes*)

na literatura como semântica Unix.

Unix tem um mecanismo muito simples para controlar o acesso aos arquivos, conhecido como modo de bits de Unix. São definidos três tipos de acesso para cada arquivo: *read* para leitura, *write* para escrita e *execute* para execução. Além disso, os acessos também são separados em três grupos: acessos do usuário proprietário do arquivo, acesso dos usuários que pertencem ao mesmo grupo do proprietário do arquivo e os acesso dos usuários restantes. Com este mecanismo de controle, unido a um processo de autenticação inicial do usuário, Unix consegue controlar de uma forma simples e eficiente o acesso aos arquivos.

Todas estas características fazem do Sistema de Arquivos de Unix um sistema simples, eficiente, confiável e de alto desempenho. Além disso, a interface oferecida pelo sistema também é muito simples, facilitando seu uso. No entanto, o Sistema de Arquivos de Unix forma parte do *kernel* do sistema operacional e não pode ser utilizado separadamente, dificultando seu uso em outros sistemas.

A.2 Network File System

NFS (*Network File System*) [MBKQ96, SGK+85] é um sistema de arquivos criado pela Sun como uma solução para a necessidade de compartilhar informação em uma rede de sistemas computacionais diferentes. Ele é provavelmente o sistema de arquivo distribuído mais utilizado no mercado e encontra-se instalado em milhares de redes. NFS é um excelente sistema para LANs pequenas. É simples, eficiente, rápido, relativamente fácil de instalar

e portável, existindo versões para quase todas as plataformas computacionais atuais. No entanto, não é muito escalável e é inadequado para a manipulação de arquivos em escala maior, sendo esta a principal razão para que NFS não seja usado em WANs [WLS+85].

NFS está baseado em um modelo ponto a ponto. Qualquer computador usando NFS pode fazer o papel de servidor, de cliente ou ambos. Para compartilhar a informação, NFS oferece dois mecanismos principais: exportação e montagem de sistemas de arquivos. Estes mecanismos são oferecidos através de três componentes fundamentais: o protocolo de montagem, os servidores e os *daemons*. Os servidores [Jus89] tornam seu sistema de arquivos disponível para os demais computadores da rede, exportando-o. Os clientes podem usar o sistema de arquivos de um determinado servidor montando-o como parte de seu próprio sistema de arquivos local. Estes processos de montar e desmontar são feitos automaticamente por NFS, criando uma hierarquia do sistema de arquivos conhecida como VFS (*Virtual File System*), cuja tarefa é manipular os arquivos locais e remotos da mesma maneira. Os pedidos remotos são analisados pelo *daemon* do servidor, que dá uma resposta que depende das permissões de acesso. A Figura A.2 mostra com mais detalhes a iteração cliente-servidor em NFS.

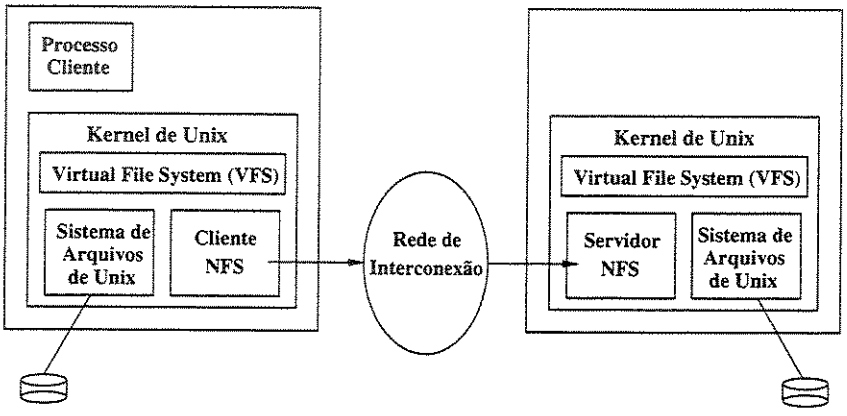

Figura A.2: Interação cliente-servidor em NFS

Historicamente NFS tem usado o protocolo UDP/IP (*Universal Datagram Protocol/ Internet Protocol*); porém a implementação da versão 3 usa TCP/IP (*Transport Control Protocol/Internet Protocol*) [PJS+94, Sun89], o que torna mais confiável a troca de informações e elimina os inconvenientes encontrados usando UDP/IP. NFS utiliza o formato XDR (*eXternal Data Representation*) para a representação externa dos parâmetros que são inseridos nas chamadas RPC (*Remote Procedure Call*) [BN84], usadas pelos clientes para ter acesso aos servidores. O fato de usar o RPC e o XDR (disponíveis gratuitamente) faz do NFS um sistema independente do sistema operacional utilizado nos computadores da rede, aumentando sua portabilidade.

Em NFS os clientes e os servidores são gerenciados separadamente e este gerenciamento é feito baseado nos arquivos. O cliente é que controla a denominação dos arquivos, não sendo possível a obtenção de um espaço de nomes global e uniforme.

NFS é um protocolo *stateless*, ou seja, os servidores não armazenam informação alguma

nem dos clientes que estão usando seus serviços nem quais arquivos eles estão utilizando, aumentando a tolerância a falhas do sistema, sem usar replicação de servidores.

NFS não é um protocolo seguro. O controle de acesso é deixado a cargo dos sistemas de arquivos locais. Um servidor, quando exporta seu sistema de arquivos, pode especificar uma lista dos *hosts* que podem montá-lo (um *host* específico, uma determinada subrede ou qualquer *host* da Internet). Para cada um destes *hosts* ou grupos de *hosts*, o sistema de arquivos pode ser disponibilizado somente para leitura ou para leitura e escrita. O acesso aos arquivos é feito usando o modelo de bits de Unix para usuários e grupos de usuários definidos pelo administrador do sistema. Os administradores dos sistemas de arquivos locais podem usar sistemas mais seguros, como o Kerberos [SNS88], por exemplo.

NFS tem cópia (cache) dos arquivos na parte do cliente. É usado um sistema de cache baseado em tempo, que assume que o arquivo estará atualizado por um período de tempo arbitrário (3 segundos para arquivos, 10 segundos para diretórios) depois que o mesmo é enviado ao cliente. Este mecanismo não garante uma consistência de cache estrita, e pode causar problemas quando um arquivo é muito usado por vários clientes simultaneamente. Umas das implementações do protocolo NFS, conhecida como *Not Quite NFS* (NQNFS) [Mac94], usa uma política de cache mais consistente, baseada em *leases* [GC89]. Quando um cliente atualiza seu cache, junto com os dados ele recebe um *lease*, que é uma autorização para usar esses dados no cache por um período de tempo determinado. Durante esse período de tempo o cliente tem a certeza de que os dados no seu cache estão atualizados e qualquer mudança no servidor será notificada. Uma vez que o período de tempo expire, o cliente tem que entrar em contato com o servidor para poder utilizar os dados no cache novamente, atualizando-os caso necessário. Esta abordagem é mais consistente que o sistema de cache baseado em tempo utilizado na maioria das implementações comerciais de NFS.

Concluindo, NFS é um sistema de arquivos mais adequado para redes pequenas, devido a sua simplicidade, eficiência, rapidez e portabilidade, oferecendo acesso aos arquivos de forma totalmente transparente. No entanto, não é escalável, não tem um espaço de nomes global e uniforme para se referir aos seus arquivos e, embora ele use cache nos clientes, implementa um mecanismo de consistência fraco. Estas características fazem com que seja inadequado para a manipulação de arquivos em uma escala maior, como em um sistema de processamento maciçamente paralelo na Internet.

A.3 Andrew File System

Outro sistema de arquivos muito popular e provavelmente o maior competidor de NFS é o AFS (*Andrew File System*) [How88, CD93], desenvolvido na Universidade de Carnegie-Mellon como o sistema de arquivos para o sistema Andrew [MSC+86], mas que atualmente é um produto comercial da Transarc. AFS foi desenvolvido para satisfazer as necessidades de compartilhar informação de computadores conectados em uma rede de campus, obtendo ótimos resultados de escalabilidade e desempenho, mas também obteve bons resultados em alguns testes feitos em WANs [SS96b, SS96a]. Isto faz de AFS um sistema de arquivos muito utilizado em vários sistemas de computação atuais. AFS tem uma arquitetura similar à de NFS; também usa XDR, RPC e VFS. No entanto, tem grandes diferenças referentes ao

seu projeto. Uma das principais diferenças é a portabilidade. AFS é um sistema baseado em Unix e que usa diretamente os serviços oferecidos pelo seu *kernel*. Isto faz de AFS um sistema pouco portável e que somente pode ser utilizado por computadores que usem este sistema operacional.

Uma outra diferença com NFS é que AFS é totalmente baseado em um modelo cliente-servidor, que faz total distinção entre os servidores e os clientes. A Figura A.3 mostra a arquitetura geral do AFS. Os dois componentes principais do modelo usado são *Vice* e *Venus*. *Vice* é um processo servidor *multi-thread*, com um *thread* para manipular cada um dos pedidos dos clientes, e que oferece todo o serviço de arquivos do sistema. *Venus* é um processo cliente que é executado em cada um dos computadores que usam AFS, ou seja, é a interface dos serviços oferecidos pelo AFS. Note que ambos componentes são implementados sobre o *kernel* de Unix.

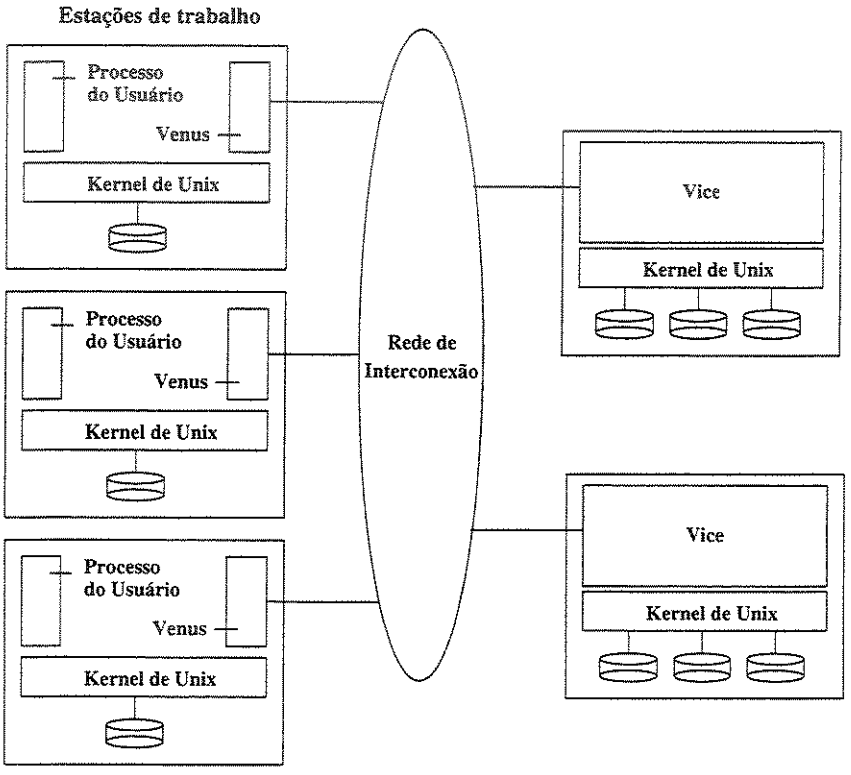

Figura A.3: Arquitetura geral de AFS

Em AFS são os servidores que determinam a hierarquia de nomes dos arquivos e a exportam aos clientes, permitindo-lhes ter um espaço de nomes global e uniforme [Eve90]. Uma das principais funções de *Venus* é tornar o espaço de nomes totalmente transparente e independente de localização. AFS também permite que múltiplos servidores possam ser agrupados em uma unidade lógica chamada célula, o que dá uma maior uniformidade ao sistema de nomes. Além disso, os arquivos também podem ser agrupados em coleções chamadas de volumes [Sid86], o que torna mais simples e eficiente o gerenciamento de toda

a informação espalhada pelos servidores. Este gerenciamento pode ser feito a partir de qualquer ponto do sistema.

AFS usa uma semântica de sessão e implementa um sistema de cache consistente para esta semântica, isto é, este mecanismo de consistência garante que, uma vez que um cliente fecha um arquivo, todas as mudanças feitas nele são visíveis em posteriores aberturas do arquivo por outros clientes. Para implementar este mecanismo de consistência, AFS usa um cache grande (10 a 20 MBytes) na parte do cliente e tem um mecanismo chamado *callback promise* para garantir que a informação do cliente esteja atualizada. Quando um cliente abre um arquivo, uma cópia dele é colocada no seu computador. Junto com o arquivo, o servidor AFS adiciona o *callback promise*, isto é, uma promessa de que o servidor notificará o cliente se o arquivo mudar e ,portanto, a cópia no seu cache ficar desatualizada. Esta promessa tem um tempo de validade e o servidor poderá cancelá-la se este tempo expirar. Quando o arquivo é aberto, *Venus* checa se está no cache e se está atualizado (ou seja, se o *callback promise* é válido). Se for uma cópia válida, *Venus* retorna à aplicação um descritor do arquivo. Caso o arquivo não esteja atualizado, *Venus* entra em contato com o servidor adequado e obtém uma cópia atualizada deste arquivo. Este mecanismo permite ao AFS obter um bom desempenho, garantindo a consistência da informação espalhada pelos seus servidores segundo a semântica utilizada.

Para aumentar o desempenho e a disponibilidades do serviço oferecido, AFS também permite replicação de servidores [HH93]. No entanto, o servidor espelho só pode ser de leitura (*read only*), e todas as atualizações têm que ser feitas no servidor principal. Isto pode ocasionar congestionamento, se vários arquivos forem modificados simultaneamente. Além disto, AFS usa servidores *stateful*. Embora estes servidores sejam muito eficientes, eles não são tolerantes a falhas e sua escalabilidade fica comprometida quando o número de clientes aumenta.

O problema da segurança foi um dos pontos considerados no projeto de AFS. Todo o tráfego entre os clientes e os servidores é cifrado e são usados mecanismos para a autenticação dos clientes do sistema, tais como o Kerberos. Além destes mecanismos de autenticação é usada uma combinação de listas de controle de acesso (*ACL - Access Control List*) e do modo de bits de Unix para controlar o acesso aos diretórios e aos arquivos que formam a hierarquia dos sistemas de arquivos dos servidores. Todos estes mecanismos tornam mais seguro o uso de AFS.

Em resumo, AFS é um sistema seguro e de alto desempenho. Ele oferece um espaço de nomes global e mecanismos de replicação de servidores, garantindo consistência na semântica implementada. No entanto, a replicação de servidores é só para leitura e o fato de seus servidores serem *stateful* faz com que ele não seja nem tolerante a falhas nem escalável o suficiente para ser usado como sistema de arquivos para arquiteturas de processamento maciçamente paralelo na Internet. Além disso, AFS é um protocolo proprietário, não disponível gratuitamente, e está implementado sobre Unix, fatos que dificultam a sua portabilidade.

A.4 Distributed File System

DFS (*Distributed File System*) [HKM+88] é baseado no mesmo projeto que AFS, e ambos são produtos comerciais da Transarc. DFS foi criado para ser o componente mais importante de *OSF/DCE (Open Software Foundation/Distributed Computing Enviroment)*. Por este motivo, o objetivo principal de DFS foi aumentar os níveis de escalabilidade de AFS [Tob89], permitindo não somente seu uso em redes de campus e organizações, mas também permitindo compartilhar informação entre diferentes organizações espalhadas geograficamente, mesmo no mundo todo. Este aumento por parte do DFS das facilidades oferecidas por AFS tornaram mais complexos seu projeto e utilização.

DFS também tem uma arquitetura cliente-servidor. Tanto os servidores como os clientes compartilham os arquivos através dos serviços oferecido pelo *DCE Base Services*. Os servidores DFS executam o *Enhanced File Service* (EFS) para implementar os serviços principais do sistema de arquivos, como replicação, monitoramento, reiniciar servidores, dentre outros.

DFS oferece uma administração centralizada dos recursos do sistema. A partir de qualquer ponto do sistema podem ser gerenciados todos os recursos de forma simples, eficiente e segura. Todo este gerenciamento é feito usando uma aplicação gráfica chamada de *Scout*, que permite monitorar e controlar qualquer quantidade de servidores espalhados no sistema, reportando quais servidores estão recebendo mais pedidos dos clientes, se os servidores estão funcionando corretamente, quanto espaço de disco eles ainda têm disponível, entre outras informações importantes. Desta forma, e ajudados por mecanismos implementados para manter o desempenho individual dos servidores, os administradores do sistema podem reajustar as cargas de trabalho e aumentar o desempenho do sistema.

Em DFS múltiplos servidores podem ser agrupados em células e múltiplos domínios administrativos podem ser especificados sobre uma célula. Isto permite a expansão de uma célula por uma empresa, mantendo uma administração independente de cada divisão individual. Também é possível agrupar os arquivos em conjuntos de arquivos (*fileset*), oferecendo uma forma uniforme e independente de localização para denominar os arquivos.

A principal característica que diferencia DFS de AFS é a forma como o cache é implementado. Em DFS os arquivos também são descarregados nos clientes, usando um cache grande (10 a 20 MBytes) para aumentar o desempenho e diminuir o tráfego na rede. No entanto, a forma de implementar os mecanismos de coerência desta informação armazenada nos cache dos clientes é o que diferencia DFS de AFS. DFS dá *tokens* aos clientes para leitura e escrita. Ter um *token* de leitura garante ao cliente que a cópia que ele tem do arquivo está atualizada. Um *token* de escrita permite aos clientes acessos de escrita aos arquivos. Estes tokens permitem ver todas as mudanças nos arquivos com uma granularidade arbitrariamente pequena e garantem a coerência entre todas as cópias.

DFS também oferece replicação de servidores para distribuir a carga entre estes servidores, evitando possíveis pontos de gargalo no sistema. Embora DFS ofereça níveis maiores de escalabilidade que AFS, os servidores ainda são *stateful*, precisando de mecanismos complexos, e às vezes demorados, de recuperação ante possíveis falhas. DFS oferece uma aplicação chamada de *DFS Local File System* que permite aos servidores reiniciarem-se o mais rápido possível após uma falha.

O DFS também é um protocolo seguro [Sat89] e, da mesma maneira que AFS, ele mantém estritos mecanismos de segurança baseados em esquemas de autenticação dos clientes, na criptografia de toda a informação que é enviada pela rede e na combinação de listas de controle de acesso (*Control Access List*) e do modo de bits de Unix para controlar o acesso aos diretórios e aos arquivos.

Concluindo, DFS é um sistema com um bom projeto, mas relativamente complexo, o que dificulta a sua ampla adoção e utilização em grande escala. Ele oferece um espaço de nomes global e uniforme para se referir aos arquivos e usa cache nos clientes, mantendo a consistência das diferentes cópias. Além disso, também oferece replicação de servidores e mecanismos para balancear a carga de trabalho entre eles. No entanto, estes mecanismos de balanceamento são manuais e são os administradores quem fazem a distribuição de carga, tarefa difícil em um sistema maciçamente paralelo de grande escala. Além disso, DFS usa servidores *stateful*, o que diminui a tolerância a falhas e escalabilidade dos mesmos, e é implementado somente para plataformas Unix, o que dificulta seu uso em um MPVC para máquinas heterogêneas

A.5 Novell Netware

Novell alavancou o processamento em rede com a criação do primeiro sistema operacional de rede de sucesso comercial, o Netware. Com a criação do Netware, Novell tornou-se um dos líderes dos sistemas operacionais e softwares para rede locais, com mais de 40 milhões de usuários. Atualmente uma grande quantidade de empresas usam redes Novell para gerenciar seus recursos. Com a sua última versão, o Netware 5, Novell incorporou novos recursos para possibilitar a utilização de Netware em Intranets corporativas e até na Internet. Dentre os novos serviços oferecidos, dois deles são os principais encarregados da eficiência e bom desempenho do Netware: o *Novell Directory Service* (NDS) [Nov98d] e o *Novell Storage Service* (NSS) [Nov98e].

O NDS é o serviço de diretórios do Netware. Um serviço de diretórios é em essência uma base de dados de objetos que armazena os dados de cada um dos recursos existentes em uma rede e os seus usuários. Geralmente estes objetos são organizados de uma forma hierárquica, possibilitando uma melhor organização da informação da rede e reduzindo a complexidade na sua manipulação. Cada um destes objetos pode ter relação com qualquer outro objeto, relação controlada pelo serviço de diretórios usando dois mecanismos principais: a autenticação e a autorização. Em NDS o usuário se registra uma vez mediante um identificador e uma senha. Logo após, ele obtém acesso de forma transparente a todos os recursos para os quais ele tem permissão de acesso. Desta forma, o usuário pode ter acesso a aplicações, arquivos, impressoras e outros recursos distribuídos na rede local de uma maneira rápida, fácil e eficiente. Além disso, os administradores do sistema podem gerenciar todos os recursos espalhados na rede a partir de qualquer ponto do sistema usando uma ferramenta gráfica chamada de *NWAdmin*, que permite diminuir o custo, às vezes alto, destas operações.

A escalabilidade e a tolerância a falhas são outras características do NDS. O NDS é um serviço distribuído que usa mecanismos de replicação. Ele divide a informação em

segmentos (chamados de partições) e os distribui pela rede. Além disso, podem ser feitas tantas cópias ou réplicas destas partições quantas sejam necessárias. Se uma partição primária ficar inacessível, são usadas outras cópias espalhadas na rede. Desta forma, o NDS garante um alto nível de tolerância a falhas e escalabilidade, sendo capaz de gerenciar redes com um grande número de recursos e usuários. Diferentes de outras redes, o Netware tem a fama de ser um dos poucos sistemas que trabalha com mais eficiência quando o número de usuários que gerencia é grande. Isto é devido, em grande parte, ao NDS.

O NDS é um sistema de diretório seguro [Nov98f]. Ele usa mecanismos de autenticação baseados em RSA (algoritmo de criptografia de chave pública [RSA78, Sta95]). Estes mecanismos de autenticação são orientados a sessões. Cada usuário tem seus próprios níveis de acesso sobre os recursos da rede, os quais são definidos pelos administradores. O NDS permite definir acessos sobre uma subárvore da árvore de diretórios, os quais são herdados por todos os objetos que pertencem a essa subárvore. Esta forma simplifica o processo de definição dos níveis de acesso dos recursos gerenciados e diminui o custo de administração.

Por último, o NDS usa vários protocolos que constituem padrões atuais, o que aumenta sua portabilidade. Entre os principais protocolos utilizados destaca-se o LDAP *(Lightweight Directory Access Protocol)* [Nov98a], que apesar do pouco tempo de existência, é o padrão para o acesso à informação de diretórios pelas aplicações baseadas na Internet. Além do LDAP, o NDS implementa outros protocolos como o DNS *(Domain Name Service)* [AL94], o RADIUS *(Remote Authentication Dial-In Service)*, o SMB *(NT Server)*, dentre outros. A implementação destes protocolos torna o NDS um dos serviços de diretórios mais usados [Nov98b].

O NSS é um serviço de arquivos de 64 bits. NSS consegue trabalhar com arquivos de tamanho grande (permite arquivo de até 8 TBytes) e com grandes quantidades de volumes, diretórios e arquivos. Teoricamente, o NSS não tem limite no número de volumes que pode montar; o limite é imposto pela capacidade de armazenamento. Além disso, o NSS implementa algoritmos para a utilização máxima do espaço de armazenamento dos seus servidores. Todos os servidores usados são *stateless*, o que aumenta a tolerância a falhas e a escalabilidade dos mesmos.

Novell oferece acesso aos recursos gerenciados para as principais plataformas existentes, tais como DOS, Windows 3.1/95/98/NT, Macintosh e Unix [Nov98c]. Estes clientes podem usar todos os serviços oferecidos, como o NDS e o NSS. O processo de instalação destes clientes é relativamente simples; no entanto a sua configuração é complexa.

Concluindo, Netware é um dos líderes atuais em serviços de redes locais e continua seu crescimento oferecendo novas soluções para Intranet e para o gerenciamento de recursos através da Internet. Ele é eficiente, escalável e tolerante a falhas, oferecendo serviços de qualidade. No entanto, a sua utilização requer a instalação de software específico para cada plataforma em particular e a sua configuração não é um processo trivial. Além disso, é um sistema orientado a computadores pessoais e redes locais que inicialmente não foi implementado sobre TCP/IP, fato que dificultou sua disseminação na Internet. E também é um protocolo proprietário. Todos estes fatos tornam difícil sua utilização como sistema de arquivos para um computador virtual baseado na Internet.

A.6 Microsoft Windows NT

Windows NT [SC98, Moh99] é a solução desenvolvida pela Microsoft de um sistema operacional de rede orientado a computadores pessoais. Atualmente conta com milhões de usuários por todo o mundo, sendo o maior competidor de Novell na área de sistemas operacionais de redes para computadores pessoais. Windows NT usa uma interface gráfica muito similar às utilizadas por outros sistemas operacionais da Microsoft (Windows 95, Windows 98, etc.), o que facilita seu uso e entendimento. Windows NT tem dois componentes principais, os servidores (*Windows NT Server* [Min00]) e as estações de trabalho (*Windows NT Workstations* [Car99]). Diferente de outros sistemas operacionais de rede, como Novell, um servidor NT também pode ser usado como uma estação de trabalho, aumentando assim suas funcionalidades. Além disso, a instalação e administração dos seus componentes é relativamente simples se comparada com outros sistemas similares. Estas são algumas das principais razões da boa aceitação de Windows NT como sistema operacional para um ambiente de redes.

Embora Windows NT use uma interface gráfica similar a sistemas como Windows 95, sua arquitetura é completamente diferente. Ele apresenta um modelo que isola os processos do sistema dos demais processos das aplicações, chamado de *kernel mode*. Assim, os processos das aplicações não têm acesso direto aos recursos do sistema, minimizando a ocorrência de falhas comuns em outros sistemas operacionais da Microsoft. Além disso, Windows NT tem seu próprio sistema de arquivos, o NTFS (*NT File System*). NTFS é um sistema de arquivos transacional (similar aos bancos de dados transacionais), possibilitando que o sistema possa se recuperar ante possíveis falhas antes que dados errados sejam escritos no disco. Também provê controle total sobre seus arquivos e diretórios, usando listas de controle de acesso.

Windows NT foi inicialmente projetado como um sistema operacional de rede baseado na arquitetura Intel. No entanto, atualmente pode ser instalado em plataformas baseadas em arquiteturas Alpha e PowerPC. Além disso, oferece suporte para muitos dos principais protocolos de redes existentes, como o protocolo IPX/SPX da Novell, o TCP/IP, dentre outros. Estes fatos aumentam a portabilidade do Windows NT.

O principal mecanismo de tolerância a falhas implementado em Windows NT é o uso de discos "espelhos". Em cada servidor NT pode ser instalado um segundo disco que contém as mesmas informações do disco principal. Se alguma falha ocorrer no primeiro disco, o disco espelho começa a ser usado. No entanto, se o computador ficar fora do ar, esta solução não resolve o problema. Por isso, também podem ser usados servidores espelhos, que são usados caso o servidor principal falhe. O maior problema do uso de discos e servidores espelhos é a necessidade de mais recursos, que somente serão utilizados em caso de falhas.

Em Windows NT, a administração de todos os recursos é centralizada. Ele oferece várias ferramentas gráficas para gerenciar a partir de um único ponto de sistema todos os servidores o estações de trabalho instaladas, os usuários do sistema, os diferentes serviços oferecidos (como por exemplo o serviços de impressão), dentre outros. Esta administração centralizada permite minimizar o custo de administração quando a rede gerenciada é grande.

Windows NT também oferece um conjunto completo de serviços de Internet, possibi-

litando que qualquer servidor NT possa ser usado para executar um serviço para Web de forma eficiente. O IIS (*Internet Information Services* [Mic98], incluído com o Windows NT) é um servidor Web capaz de gerenciar sites com um grande números de pedidos. O próprios site da Microsoft é gerenciado usando este serviço. Windows NT também oferece soluções para DNS, Gopher [AML+93] e FTP. Além do IIS, Windows NT inclui um Servidor Gopher e um Servidor FTP, que permite implementar serviços gopher e transferência de arquivos. Todos estes serviços oferecidos por Windows NT o tornam um sistema operacional com amplas possibilidades.

Concluindo, Windows NT é um dos sistemas operacionais para redes locais mais usados atualmente, e embora seja um sistema relativamente jovem, o número de usuários em todo o mundo que usam seus recursos aumenta de forma acelerada. Ele tem uma interface gráfica muito conhecida e utilizada (a interface gráfica dos sistemas operacionais da plataforma Windows) e apresenta um modelo mais tolerante a falhas que os restante produtos da Microsoft. Windows NT tem um sistema de arquivos transacional que usa listas de controle para gerenciar seus arquivos e diretórios. Além disso, permite a administração centralizada de todos os recursos do sistema e oferece um conjunto de serviços para Internet, como o IIS, servidor Gopher e servidor FTP. No entanto, Windows NT possui requerimentos de hardware, tanto de CPU como de memória, muito altos, precisando de processadores Pentium ou similar e 32 MBytes de memória para obter um resultado aceitável. Além disso, ele é um protocolo proprietário e não está disponível gratuitamente, aspectos que limitam seu uso em uma plataforma para o processamento maciçamente paralelo na Internet.

A.7 Coda

Coda [SKK+90, KS91] é um sistema de arquivos que visa obter um alto nível de disponibilidade em ambientes formados por um grande número de computadores, mantendo o desempenho. Com este objetivo, Coda usa dois mecanismos principais: a replicação de servidores e as operações de desconexão. Coda é o sucessor de AFS, e muitos aspectos nos dois projetos são similares. Na ausência de falhas, Coda é muito similar a AFS, mas em sua presença, ele procura obter uma disponibilidade máxima, aumentando os níveis de tolerância a falha do seu predecessor.

Coda também tem uma arquitetura cliente-servidor. Ele faz completa distinção entre servidores e clientes. Os servidores, geralmente em pequeno número, são computadores seguros e estáveis fisicamente que executam um software também seguro e monitorado pelo sistema. Os clientes são numerosos e geograficamente distribuídos. A comunicação entre os servidores e os clientes é feita usando RPC. Tanto os servidores como os clientes executam Unix como sistema operacional, limitando o uso de Coda somente para esta plataforma. A arquitetura geral de Coda é similar à arquitetura de AFS, como mostra a Figura A.4. Também está formado por dois componentes fundamentais, *Venus* e *Vice*, que têm as mesmas funções que em AFS.

Coda também oferece um espaço de nomes global e uniforme para referenciar-se aos seus arquivos e o disco local dos clientes é usado para criar um cache de arquivos. Tanto os arquivos como os diretórios são armazenados totalmente no cache do cliente. Futuros

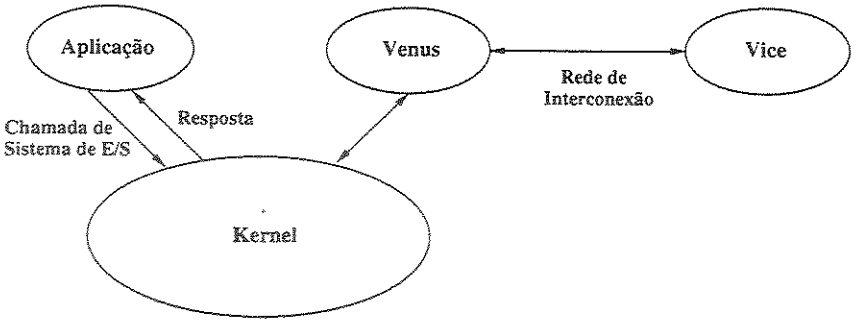

Figura A.4: Arquitetura geral de Coda

acessos aos arquivos são realizados no cache para aumentar o desempenho do sistema. Uma vez que a cópia de um arquivo é armazenada no cache, o cliente recebe um *callback promise*, isto é, uma garantia de que será avisado de qualquer modificação feita no arquivo por qualquer outro cliente, garantindo desta forma a consistência de toda a informação utilizada. Os servidores em Coda também são *stateful*, dificultando a sua escalabilidade e recuperação ante possíveis falhas.

Possivelmente a maior diferença entre AFS e Coda surja quando aparece uma falha no sistema. Neste caso, Coda visa obter o maior nível possível de disponibilidade. Para isto, ele nega o acesso a uma cópia do arquivo somente se tem a certeza de que a cópia está desatualizada. A replicação de servidores é usada de forma intensiva para obter este resultado. Como em AFS, o volume é a unidade de replicação em Coda. Um volume é uma coleção de arquivos que é armazenada em vários servidores. É usado um mecanismo de vetores para manter a consistência entre todas estas cópias. O conjunto de servidores que mantém cópias de um volume é chamado de Grupo de Armazenamento do Volume (*VSG - Volume Storage Group*). Para cada volume em Coda é mantido o subconjunto de todos os VSG disponíveis nesse momento, chamado de Grupo de Armazenamento do Volume Acessível (*AVSG - Accessible Volume Storage Group*). Na ausência de falhas, o VSG e o AVSG são idênticos. Como todos os servidores são monitorados pelo sistema, quando um servidor ou seu enlace falham (determinado por um *timeout*), ele é eliminado do ASVG. Uma vez que é reiniciado, o servidor é adicionado novamente ao vetor. Quando um arquivo é modificado, Coda espalha as modificações por todos os servidores do ASVG aos quais o arquivo pertence. Esta abordagem permite uma maior disponibilidade da informação e uma maior garantia de que a cópia em cada um destes servidores esteja atualizada. É implementado um mecanismo de RPC em paralelo para tornar mais eficiente esta propagação do arquivo. Quando um cliente abre um arquivo para leitura, Coda seleciona um servidor preferencial (*preferred server*) no ASVG que atenderá o pedido do cliente. Este servidor é selecionado segundo os critérios de proximidade física, carga e velocidade dos servidores. Coda garante que este servidor preferencial sempre tenha a cópia mais atualizada do arquivo. Dependendo das falhas ocorridas, os clientes podem ter diferentes ASVG.

Concluindo, Coda é um sistema que herda o bom desempenho do AFS, aumentando a tolerância a falhas e a disponibilidade dos seus servidores mediante utilização de mecanis-

mos de replicação e operações de desconexão. No entanto, ele utiliza servidores *stateful*, o que diminui sua escalabilidade, e somente é implementado em plataformas Unix, dificultando sua portabilidade. Estas duas características, além da necessidade de um processo de instalação complexo, limita seu uso em MPVCs baseados na Internet, onde a escalabilidade e a portabilidade são duas das características mais desejadas.

A.8 Bayou

Bayou [DPS+95] é um projeto da Xerox PARC que visa compartilhar dados através de usuários móveis. Ou seja, Bayou é um sistema de arquivos para ser executado em ambientes de computação móveis, onde a conectividade é muito baixa e os computadores participantes são geralmente computadores portáteis (*laptop*) que não têm uma conexão física permanente à rede, se conectando através de meios de comunicação sem fio tais como celulares e transmissões por rádio. A característica fundamental deste tipo de sistema é que os computadores podem estar, voluntária ou involuntariamente, sem conexão durante longos períodos de tempo com um ou vários dos recursos com os quais ele quer compartilhar informação. Portanto, o objetivo fundamental de Bayou é oferecer mecanismos que permitam aos usuários móveis ler e escrever de uma forma simples e eficiente dados que estão sendo compartilhados.

Como estes computadores portáteis têm pouca capacidade de armazenamento para manter cópias de todos os dados de que precisam, a arquitetura de Bayou está baseada na divisão de funcionalidade entre seus dois componentes fundamentais, os servidores e os clientes. Os servidores armazenam uma ou várias coleções de dados, chamadas também de banco de dados (*database*) e os clientes usam estes dados nas suas operações. Qualquer um dos cliente poderá se comunicar com qualquer um dos servidores e qualquer computador, inclusive os *laptops*, poderão fazer o papel de servidor, dependendo das suas capacidades. Esta arquitetura é semelhante à arquitetura de NFS, onde qualquer computador pode fazer o papel de cliente, de servidor ou ambos. Desta forma, Bayou visa aumentar a disponibilidade dos seus serviços.

Neste tipo de sistema, a replicação de informação é fundamental. Para aumentar as possibilidades de leitura e escrita dos usuários, mesma quando eles estejam sem conexão com os outros computadores do sistema, Bayou implementa o esquema de replicação *read-any/write-any* (ler de qualquer um/escrever em qualquer um). Usando este esquema, Bayou permite aos seus usuários ler a partir de e escrever para qualquer uma das cópias dos dados espalhadas pelos distintos servidores. Bayou utiliza um protocolo *anti-entropy*, também chamado de processo de reconciliação, para sincronizar o sistema de arquivos. O *anti-entropy* garante que todas as cópias de um determinado banco de dados convergirão a um mesmo estado. Para obter este resultado, cada servidor desse banco de dados não somente receberá todas as operações de escrita; ele também manterá estas operações ordenadas de uma forma consistente. Este processo de reconciliação é feito entre dois servidores (*peer-to-peer anti-entropy*). Cada servidor periodicamente escolhe outro servidor para atualizarem suas cópias. A seleção do servidor depende da carga de trabalho e do custo e benefício desta operação. Portanto, Bayou não garante que as escritas sejam propagadas pelas diferentes

réplicas rapidamente, oferecendo e implementando uma consistência fraca entre elas.

No entanto, Bayou é capaz de detectar conflitos de atualização entre as réplicas de um determinado banco de dados [TTP+95]. Estes possíveis conflitos ocorrem quando dois ou mais clientes tentam escrever em um mesmo servidor (*conflito write-write*) ou quando um cliente tenta atualizar um dado baseado em uma leitura que já está desatualizada (*conflito read-write*). Bayou detecta estes conflitos com ajuda das aplicações, mediante um mecanismo chamado de conjunto de dependência (*dependency set*). Cada operação de escrita está formada pelos dados que serão escritos ou modificados e um conjunto de dependência. Um conjunto de dependência é um conjunto de *queries* e seus resultados esperados, que são fornecidas pelas aplicações. Um conflito é detectado quando o servidor onde está armazenado o banco de dados executa essas *queries* e não obtém o valor esperado. Bayou também oferece meios para resolver estes conflitos. Cada operação de escrita contém um procedimento, chamado de *mergeproc*, também fornecido pelas aplicações. Este procedimento é invocado quando é detectado um conflito e sua tarefa é resolvê-lo. O *mergeproc* lê o banco de dados armazenado no servidor e resolve o conflito gerando um conjunto de atualizações que sejam apropriadas para o conteúdo do banco de dados. Resumindo, uma operação de escrita em Bayou está formada pelos dados que serão atualizados, um conjunto de dependência e um *mergeproc*. O conjunto de dependência e o *mergeproc* estão em íntima relação com as semânticas usadas pelas aplicações. A verificação das dependências, a execução do *mergeproc* e a atualização dos dados é uma operação atômica em relação a outras operações realizadas por outros clientes nesse servidor.

Desta forma, Bayou garante que cada aplicação individual tenha uma visão de um banco de dados em um servidor consistente com as suas ações nesse banco de dados. Assim, ele consegue minimizar os problemas de inconsistência gerados pelo uso do esquema de *read-any/write-any*, mantendo as vantagens do uso de mecanismos de consistência fraca, tais como a disponibilidade, escalabilidade, simplicidade e operações de desconexão.

Resumindo, Bayou é um sistema de arquivos que permite a usuários móveis ler e escrever dados que estão sendo compartilhados através de uma rede formada fundamentalmente por computadores portáteis e sem uma conexão fixa. Bayou oferece um esquema de replicação *read-any/write-any* e, para garantir a consistência entre os diferentes servidores, implementa um processo complexo e que envolve muitos fatores. Este esquema de replicação, embora garanta uma alta disponibilidade e escalabilidade do sistema, somente garante consistência para um número pequeno de servidores. Já para muitos servidores, a consistência entre as diferentes réplicas espalhadas pelo sistema não é garantida.

A.9 Sprite

Sprite [Nel86] é um sistema operacional implementado na Universidade de Berkeley, Califórnia, como parte do projeto SPUR, um multiprocessador de estações de trabalho de alto desempenho. O Sistema de Arquivos de Sprite [Wel90, BO91] oferece suporte para o acesso a sistemas de arquivos remotos de forma transparente, isto é, ele oferece uma interface bem definida que permite aos seus usuários tratar os arquivos locais e remotos da mesma forma. Vários dos sistemas de arquivos atuais, tais como NFS e WebFS, utilizam a interface *Vnode*

(ou outra similar) para implementar esta funcionalidade. A principal característica destes sistemas é que eles estão orientadas ao acesso a arquivos e, portanto, as operações relacionadas com a denominação de arquivos e as operações de entrada/saída estão combinadas na mesma interface. Sprite, embora vise obter o mesmo resultado, oferece duas interfaces separadas, uma para todas as operações relacionadas com a nomeação de arquivos e outra para as operações de entrada/saída [Wel92]. Com esta abordagem Sprite consegue acesso, usando as mesmas interfaces, não somente a sistemas de arquivos remotos, mas também a outros recursos, tais como periféricos e canais de comunicação inter-processos (*IPC - Inter Process Comunication channels*), característica inexistente em outras arquiteturas.

Sprite é uma arquitetura cliente-servidor e não requer que os computadores clientes possuam um sistema de arquivos privado. É usada uma abordagem baseada em objetos para representar os diferentes componentes do sistema. O *kernel* do sistema mantém uma estrutura de dados na memória principal com todos os objetos gerenciados. É criado um descritor de objeto (*object descriptor*) para especificar os objetos, formado por cinco campos principais: o tipo, o uid (*user id* - identificador do usuário), o serverID (identificador do servidor), um contador de referências e um bit de bloqueio (*lock bit*). Cada objeto é identificado pelos três primeiros campos.

A Figura A.5 mostra a arquitetura geral de Sprite. A interface para denominar os arquivos, chamada de *Pathnames interface*, oferece três possibilidades diferentes. O servidor especificado pelo nome pode ser o servidor local. Neste caso, a operação é resolvida com uma chamada do *kernel* local através do módulo *namei*. Se o servidor é remoto, é usado o protocolo RPC [Wel86] para estabelecer uma comunicação entre os *kernels* dos *hosts* envolvidos e transferir o nome ao servidor remoto. O módulo RPC é o encarregado desta operação. Por último, o servidor pode ser um processo de um usuário. Neste caso o módulo *Upcall* é o encarregado da interação com o processo do usuário requerido.

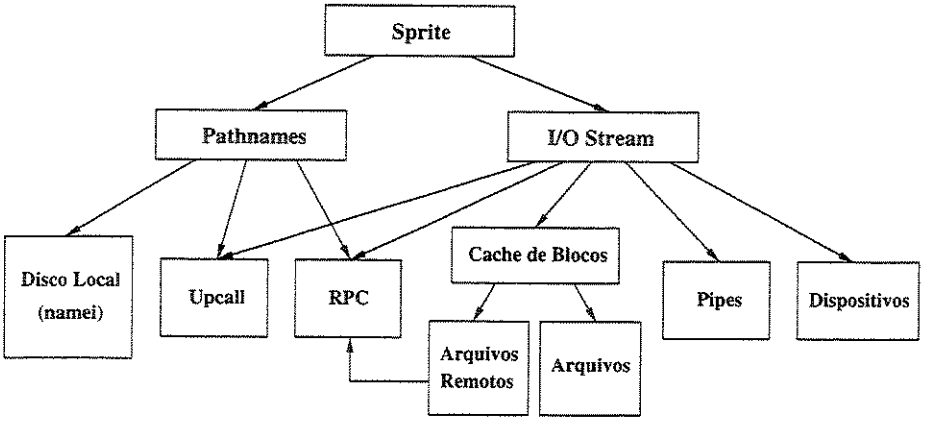

Figura A.5: Arquitetura geral de Sprite

São usados mecanismos de tabelas de prefixo (*Prefix Table Mechanism*) [WO86] para implementar um espaço de nomes global e uniforme. Este mecanismo substitui o mecanismo de montagem de sistema de arquivos de Unix e garante que os servidores exportem seus domínios de forma consistente para todos os *hosts* do sistema, o que faz com que todos

os processos que estão sendo executados vejam exatamente o mesmo espaço de nomes. O uso deste mecanismo torna mais simples a implementação dos clientes e dos servidores. No entanto, ele utiliza *broadcast* para a localização dos servidores, devido a sua simplicidade, ao invés de usar outros serviços mais sofisticados, como DNS (*Domain Name Service*).

Na manipulação dos arquivos são considerados três tipos de arquivos diferentes. O primeiro tipo é usado para representar arquivos regulares, diretórios e enlaces simbólicos. Para este tipo de arquivo o Sprite usa algoritmos de consistência de cache, os quais serão explicados mais a frente. O segundo tipo de arquivo é usado para representar os dispositivos e periféricos que são suportados pelo Sprite. O último tipo de arquivo é usado para os pseudo-dispositivos, forma pela qual são chamados alguns dispositivos que são gerenciados nos usuário [WO88]. É usada a interface de entrada/saída, chamada de *I/O Streams*, para se ter acesso a todos estes tipos de arquivos diferentes de uma forma totalmente uniforme.

O Sprite usa um protocolo *stateful*. Os servidores armazenam dados do estado dos clientes que estão usando os seus recursos. Os clientes também armazenam informação dos servidores, que os ajudam a detectar possíveis falhas nesses servidores, dentre outras funções. O uso de um protocolo *stateful* faz com que a escalabilidade do sistema de arquivos fique comprometida quando o número de clientes aumenta. Além disso, torna muito difícil a recuperação ante possíveis falhas nos clientes e nos servidores, fazendo necessário a implementação de mecanismos complexos para a sua recuperação.

Sprite utiliza, como a maioria dos sistemas de arquivos, cache no cliente para diminuir a carga dos servidores e o tráfego na rede [NWO88]. Sprite mantém as cópias dos arquivos na memória principal do cliente para permitir o uso do sistema em estações de trabalho que não tenham disco e para aumentar o desempenho. O tamanho desta memória é variável, para se adaptar melhor às capacidades de memória dos diferentes *hosts*. Para obter bons resultados, o cache em Sprite é organizado usando blocos de 4 KBytes (e não arquivos inteiros), os quais são endereçados virtualmente usando um identificador único fornecido pelo servidor e o número do bloco no arquivo. Usando este endereçamento virtual, Sprite visa obter um maior nível de flexibilidade e facilidade na implementação dos mecanismos de consistência destes cache. É usada uma política de escrita demorada (*delay write back*) para armazenar as modificações dos caches no servidor, a qual tem um desempenho maior que outras políticas como *write through*, e permite que o arquivo seja apagado em um período curto depois da atualização, não sendo necessária nenhuma comunicação com o servidor (esquema similar ao Unix). A cada 30 a 60 segundos, todos os blocos que foram modificados são enviados ao servidor para serem atualizados. No entanto, usando esta política, dados podem ser perdidos quando um cliente ou um servidor fica fora do ar. Sprite também implementa mecanismos para manter a consistência do cache. Estes mecanismos garantem que todas as leituras sempre sejam feitas na cópia mais atualizada dos dados. Os servidores são usados como os pontos centralizados de controle desta consistência de dados. Cada servidor garante a consistência dos caches relacionados com qualquer um dos seus arquivos, e os clientes interagem diretamente com ele (não é permitida interações entre os clientes). Esta abordagem, dentre outros aspectos, proíbe que os clientes possam abrir um arquivo sem contatar o servidor.

Concluindo, a principal característica do Sistema de Arquivos de Sprite é que ele oferece

duas interfaces separadas para as operações relacionadas com a denominação de arquivos e para as operações de entrada/saída, o que dá mais flexibilidade à manipulação tanto de arquivos como de outros dispositivos e periféricos. Ele também oferece um espaço de nomes uniforme e faz uso intensivo de cache para aumentar o desempenho. No entanto, Sprite usa um protocolo *stateful*, que limita a escalabilidade do sistema e torna complexa a recuperação ante possíveis falhas dos seus componentes, duas características fundamentais para um sistema de arquivos apropriado para uma plataforma maciçamente paralela na Internet.

A.10 Ficus

Ficus [GHM+90, Guy91] é um sistema de Arquivos distribuído que utiliza intensivamente mecanismos de replicação otimistas [PGJH90]. Os mecanismos de replicação otimistas são aqueles onde qualquer réplica de um arquivo pode ser atualizada a qualquer momento, o que possibilita que quando um usuário quiser atualizar um arquivo, somente precisa ter uma réplica disponível para fazê-lo. Embora este mecanismo ofereça maior flexibilidade e disponibilidade ao sistema de arquivos, ele pode gerar possíveis inconsistências entre as diferentes réplicas. Ficus oferece mecanismos para detectar e corrigir possíveis conflitos [RHR+94], sempre que o custo da operação seja baixo e não afete consideravelmente o desempenho do sistema.

Ficus agrupa as subárvores de arquivos em volumes. Um volume pode ter múltiplas réplicas, que se comunicam para manter a consistência entre elas. Uma vez que uma réplica é modificada, o sistema notifica as restantes da atualização, mediante um processo de propagação das atualizações. Se por algum motivo (particionamento da rede, perda do enlace de um *host* ou grupo de *hosts* com a rede, dentre outros) todas as réplicas não puderem se comunicar, podem surgir conflitos. Ficus garante que todos estes conflitos serão detectados e examina se é possível resolvê-los de forma automática pelo sistema.

Diferentes tipos de conflitos são detectados por Ficus, os quais são mostrados a seguir.

- Conflitos *Update/Update*
- Conflitos de nomes
- Conflitos *Remove/Update*

Para uma maior compreensão destes conflitos, serão consideradas somente duas réplicas diferentes de um determinado volume, isto é, um determinado volume mantido em dois pontos do sistema, como por exemplo, dois computadores. Os conflitos *Update/Update* ocorrem quando as réplicas do volume perdem a comunicação entre si e ambas são modificadas de forma diferente por usuários diferentes, como mostra a Figura A.6. Uma vez que a comunicação entre as réplicas é restabelecida, Ficus detecta o conflito entre as duas réplicas.

Um caso particular de conflitos *Update/Update* pode ocorrer em diretórios. Quando as réplicas perdem a comunicação, os diretórios podem ser atualizados de forma diferente.

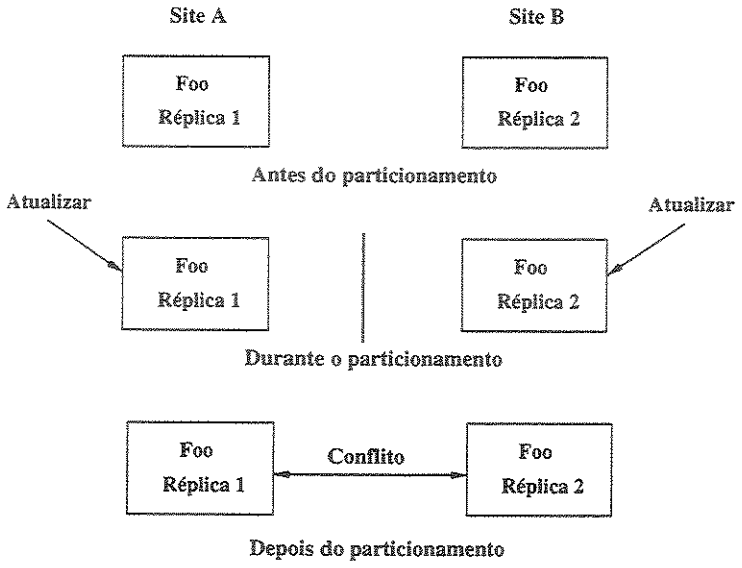

Figura A.6: Conflito *Update/ Update* em Ficus

Estes tipos de conflitos são os mais simples para serem resolvidos, já que basta que o sistema seja capaz de, uma vez restabelecida a comunicação entre as réplicas, produzir uma união dos diretórios e propagar as atualizações realizadas para todas as réplicas. O problema acontece quando dois arquivos com o mesmo nome são criados de forma independente nestas réplicas, já que as entradas em um diretório têm que ser únicas. Estes conflitos são conhecidos como conflitos de nomes.

A Figura A.7 mostra detalhes do último tipo de conflitos detectado por Ficus. Quando as réplicas perdem a comunicação entre si, uma das réplicas pode modificar um determinado arquivo e a outra apagá-lo. Desta forma, uma vez que as réplicas possam se comunicar novamente, as atualizações feitas pela primeira réplica no arquivos poderão ser perdidas. Este conflito é conhecido como conflito *Remove/ Update* e Ficus assume uma semântica chamada de *no lost update semantics*, isto é, as atualizações geradas no arquivo pela primeira réplica não são descartadas quando a segunda réplica apaga o arquivo.

Ficus detecta todos estes conflitos usando um mecanismo chamado de *version vector* [JPR+83]. Cada réplica de um arquivo tem um vetor que mantém o histórico de todas as atualizações realizadas nesse arquivo. Possíveis conflitos entre duas réplicas são detectados comparando os seus respectivos vetores. A maioria dos conflitos detectados podem ser resolvidos automaticamente por Ficus sem a participação do usuário. Um processo, chamado de *reconciliation*, é executado em *background* periodicamente nos *hosts* que mantêm essas réplicas ou cada vez que são reinicializados depois de uma falha. Este processo compara todos os arquivos e diretórios na réplica local do volume com o volume original, detectando possíveis conflitos entre elas. Se um conflito é detectado, programas especiais, chamados de *resolvers*, são encarregados de tentar resolvê-lo. Algumas experiências foram realizadas para testar todas estas idéias, cujos resultados podem ser encontrados em [HJGP92]. Ficus obteve bons resultados para ambientes de computação uniformes, onde conflitos entre

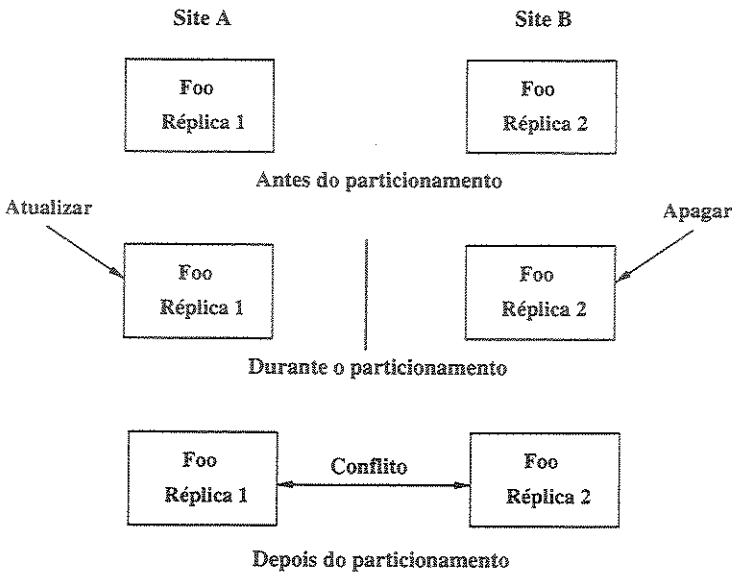

Figura A.7: Conflito *Remove/Update* em Ficus

réplicas não são muito freqüentes.

Além de todos estes mecanismos de detecção de conflitos, Ficus também oferece um espaço de nomes totalmente transparente para os seus clientes [GJHP90], isto é, todos os componentes do sistema de arquivos suportam o mesmo espaço de nomes, o que faz com que todos seus usuários tenham uma visão global e uniforme do sistema de arquivos. Além disso, o espaço de nomes utilizado também é transparente do ponto de vista da replicação, ou seja, os clientes não ficam cientes da existência de diferentes cópias dos arquivos espalhadas por diferentes estações de trabalho.

Concluindo, Ficus é um sistema de arquivos que oferece aos seus usuários um espaço de nomes global e uniforme, totalmente transparente de localização, mesmo sob o ponto de vista de replicação. Também oferece mecanismos para a detecção de diferentes tipos de conflitos e, sempre que possível, uma forma de solucioná-los sem necessidade de intervenção dos usuários. No entanto, o processo de reconciliação implementado para detectar conflitos entre diferentes réplicas somente pode ser executado entre duas réplicas; para arquivos ou grupos de arquivos que tenham mais de uma réplica este processo é ineficiente, podendo gerar inconsistências. Além disso, ele somente foi testado em ambientes de computação estáveis e uniformes, cenário que não é encontrado na Internet.

A.11 WebNFS

Um dos primeiros DFS baseados na Web foi WebNFS [Sun97a, Sun96] da Sun. O principal objetivo do WebNFS foi eliminar a sobrecarga imposta aos clientes pelo NFS na hora da conexão a um servidor. Desta forma, WebNFS aumentaria a escalabilidade do NFS, permitindo seu uso na Internet [MSC+90]. WebNFS é baseado no SunSoft NFS versão 3,

com algumas modificações importantes. Talvez uma das maiores vantagens do WebNFS é que ele permite acessos através da Web (através dos *browsers* e dos *applets Java*) de todos os servidores NFS que permitam acesso a seus arquivos.

Para alcançar seus objetivos, o WebNFS decrementa o número de conexões que um cliente e um servidor têm que estabelecer, o que é um dos principais problemas de escalabilidade em NFS. Várias modificações importantes nos clientes [Cala] e nos servidores [Calb] foram feitas no NFS para obter os níveis de escalabilidade desejados em WebNFS. A primeira mudança foi que em WebNFS tem-se um gerenciador de arquivos público (*public filehandle*) único para cada arquivo, o que faz com que não exista nenhuma negociação entre o cliente e o servidor para sua determinação. Usando este gerenciador, qualquer cliente pode ter acesso ao arquivo. Além disso, em WebNFS não é necessário a utilização do protocolo de montagem de diretórios para se obter acesso a um determinado servidor. Um cliente usa o gerenciador do arquivo público para localizar um arquivo em um determinado diretório sem necessidade de montar o sistema de arquivos desse servidor. Além disso, WebNFS usa um mecanismo chamado de *Multi-component Lookup* para, dado um nome completo de um arquivo, obter seu gerenciador público. A Figura A.8 mostra as diferenças entre este mecanismo e aquele usado pelo NFS. Todas estas modificações aumentam a escalabilidade do WebNFS e permitem seu uso em uma escala maior.

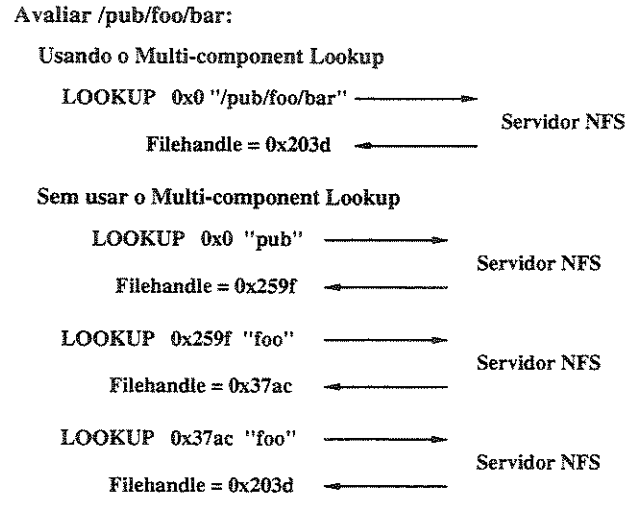

Figura A.8: Mecanismo *Multi-component Lookup* usado em WebNFS

Um dos maiores problemas de WebNFS é que ele não tem nenhum tipo de hierarquia de arquivos, baseando-se na organização da Web. Apesar de WebNFS ser um possível candidato no caso da substituição do protocolo HTTP pelo protocolo NFS para a transferência de arquivos na Web [Par97], ele não é um bom candidato para um sistema de arquivos destinado a servir um sistema paralelo baseado na mesma.

Em resumo, WebNFS consegue eliminar as principais dificuldades do NFS, que impossibilitavam seu uso na Internet, e é um bom candidato para um novo protocolo de transferência de dados na mesma. No entanto, herda outros problemas do NFS que impossibilitam seu uso em um sistema de processamento maciçamente paralelo, tais como a

implementação de mecanismos de consistência de informação fracos e o não oferecimento de mecanismos de replicação de servidores e de um espaço de nomes global e uniforme aos seus usuários.

A.12 WebFS

WebFS [VEA96, YCE+97] é outro exemplo de sistema de arquivos distribuído baseado na Web. WebFS foi desenvolvido para ser o sistema de arquivos do WebOS [VAD+96, VDA96], sistema operacional desenvolvido na Universidade de Berkerley, Estados Unidos. Ele foi projetado para ser um sistema mais amplo que WebNFS e sua intenção é ser mais que um substituto para o HTTP como protocolo de transferência. WebFS está projetado para ser usado tanto para aplicações baseadas na Web como para transferência de arquivos na mesma, ficando mais próximo do conceito de sistema distribuído baseado na Web que WebNFS.

O principal objetivo do WebFS é oferecer às aplicações acesso aos seus arquivos através de um espaço de nomes global, usando uma interface que suporte as principais operações sobre arquivos e diretórios e que tenha um desempenho semelhante aos sistemas de arquivos locais. Para lograr este objetivo, a arquitetura do WebFS está baseada na Camada *Vnode* do Unix (*Unix Vnode Layer*) [Kle86], com um estreito relacionamento com a memória virtual, permitindo o uso intensivo de cache no cliente. Desta forma, WebFS pode ser portado para diferentes variantes do Unix.

A arquitetura do sistema está dividida em duas partes: um *daemon* no usuário e um módulo *Vnode*. Quando uma aplicação realiza uma operação de entrada/saída, o sistema intercepta a chamada e transforma-a em uma operação no módulo *Vnode*. Logo após, o módulo procura o arquivo no cache. Se ele estiver no cache, a operação é completada. Caso contrário, o *daemon* é o encarregado de procurar o arquivo em um servidor WebFS ou HTTP remoto. Esta arquitetura é muito simples e tem um bom desempenho. No entanto, somente é possível ser implementada em arquiteturas Unix, o que limita uma ampla utilização. A Figura A.9 mostra mais detalhes da arquitetura do WebFS.

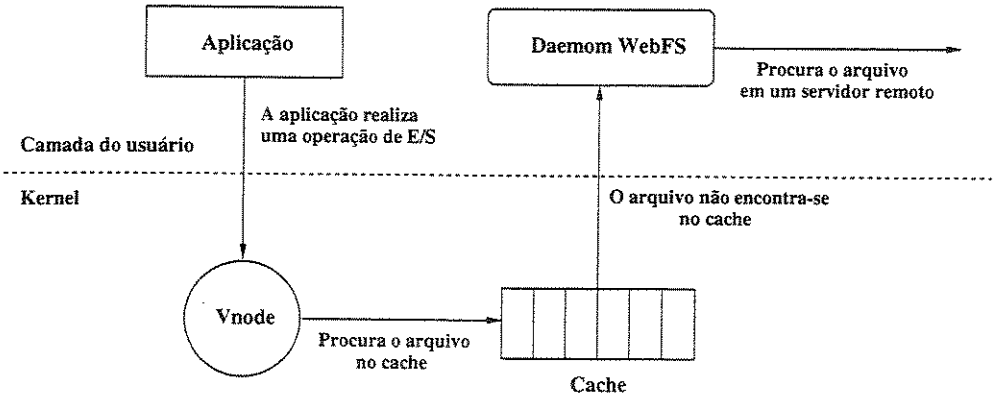

Figura A.9: Arquitetura Geral de WebFS

WebFS tem um espaço de nomes global para os seus arquivos, usando a notação URL. É definido um diretório raiz que contém o nome de todos os servidores WebFS ou HTTP que o cliente usa. As entradas neste diretório são criadas sob demanda, isto é, somente são criadas quando o cliente quer acesso a um determinado servidor. Os servidores WebFS são servidores *stateful*, diminuindo a tolerância a falhas e a escalabilidade do sistema quando o número de usuários aumenta.

Uma das vantagens de WebFS é que as regras de transporte utilizadas podem ser as de WebFS, HTTP, ou de outro protocolo diferente, como por exemplo, WebNFS, o que dá uma grande flexibilidade e portabilidade. WebFS usa mecanismos de criptografia de chave pública para autenticar tanto os servidores como os clientes do sistema, e assim cada um deles é identificado pela sua chave pública. Para cada arquivo há uma lista de controle de acesso (*ACL - Access Control List*) associando-o aos usuários. WebFS oferece utilitários para o gerenciamento das listas de controle e um sistema de autenticação e de controle de acesso, chamado de CRISIS [BVAD98], que implementa os mecanismos de segurança anteriormente citados.

WebFS tem um mecanismos de cache robusto, suportando múltiplos tipos [VA98, DWAP94]. O primeiro tipo de cache oferecido é o mecanismo de *callbacks*, muito similar ao implementado pelo AFS, adequado para compartilhar arquivos em pequena escala, onde é possível que o servidor atenda a todos os pedidos dos clientes. Os servidores mantêm uma tabela com os *hosts* que estão fazendo uso dos seus arquivos. Se um determinado *host* modifica um arquivo, o servidor envia um *callback* a todos os demais *hosts* que estão fazendo uso do arquivo para invalidarem suas cópias em cache. Este mecanismo oferece uma coerência relativamente boa para pequenos níveis de compartilhamento de informação. Quando é necessário ter-se escalabilidade para um número grande de usuários distribuídos geograficamente, este mecanismo não é adequado, usando-se um segundo modelo, o *append only*. Neste mecanismo as mudanças feitas nos arquivos são distribuídos pelos servidores de forma preguiçosa. Este mecanismo trabalha bem para dados independentes do tempo ou em situações onde tem-se muitos usuários com acesso somente de leitura e uns poucos com acesso de escrita, permitindo que todos os usuários tenham acesso às mudanças feitas em um determinado arquivo na mesma ordem em que foram realizadas. Por último, o terceiro tipo de cache está baseado na utilização de difusão (*multicast*). WebFS suporta dois canais diferentes de multicast. Um canal é utilizado para os clientes que só querem saber se a informação que eles têm está atualizada (por este canal os servidores enviam as invalidações das cópias para esses clientes). O segundo canal é usado para os clientes que querem conhecer todas as mudanças sobre os arquivos (os servidores usam este canal para enviar estas mudanças a todos esses clientes). O uso destes dois canais simplifica o projeto dos servidores e facilita a recuperação ante possíveis falhas. Para o futuro, WebFS prevê permitir às aplicações escolher qual tipo de mecanismo elas necessitam.

Concluindo, WebFS é um sistema de arquivos que permite o acesso de aplicações Unix a um espaço de nomes global e uniforme, oferecendo vários mecanismos de coerência da informação armazenada no cache dos clientes. No entanto, tem algumas características, tais como a relativa complexidade da instalação e o uso de servidores *stateful*, que diminui sua escalabilidade e tolerância a falhas e impossibilita seu uso em plataformas de processamento

paralelo.

A.13 Ufo

Outro sistema de arquivos importante é Ufo [AISS97b]. Ufo é um sistema de arquivos distribuído que permite o acesso remoto dos usuários a arquivos através dos protocolos FTP e HTTP. Seu objetivo é que os seus clientes possam trabalhar com arquivos remotos como se eles fossem locais. Ele é baseado na interceptação de chamadas do sistema operacional (*system calls*) e é quase completamente implementado nos cliente. Ele intercepta as chamadas relacionadas com acesso a dados e modifica a sua ação. Ufo é um processo que é executado nos computadores dos usuários com o objetivo de oferecer um serviço de arquivos a qualquer outro processo sendo executado nesse computador. Para que os demais processos do usuário tenham acesso aos serviços oferecidos por Ufo, eles têm que se conectar a ele (*attaching*). Logo após este processo de conexão entre o processo do usuário e o Ufo, este último intercepta todas as chamadas relacionadas a entrada/saída do processo do usuário para executá-las. Cada processo é tratado de forma independente e, em um mesmo computador, podem existir processos que utilizem e que não utilizem Ufo.

Como é mostrado na Figura A.10, Ufo está formado por dois componentes: o *Catcher* e o Módulo Ufo. O *Catcher* é o encarregado de interceptar as chamadas e enviá-las ao Módulo Ufo, que implementa o acesso remoto aos arquivos. O Módulo Ufo está formado por três camadas. A primeira camada é chamada de Serviços de Arquivos (*File Services Layer*), encarregada de identificar o arquivo remoto. A camada intermediária é a Camada de Cache (*Caching Layer*), que manipula o cache local de arquivos. E por último está a Camada de Protocolos (*Protocol Layer*), que contém os diferentes protocolos suportados pelo Ufo, tais como o HTTP, FTP, NFS, dentre outros. Além destes protocolos, Ufo permite que novos protocolos sejam agregados de uma forma relativamente simples. Se uma aplicação realiza uma chamada de sistema que é uma operação de entrada/saída de arquivos, esta chamada é interceptada pelo *Catcher*; caso contrário, ela vai diretamente ao *kernel* do sistema. Para as chamadas interceptadas, Ufo determina se o arquivo é local ou remoto. Se o arquivo é local, a operação é resolvida imediatamente. Se o arquivo é remoto, Ufo cria uma cópia do arquivo no cache local, modifica a chamada e a envia ao *kernel* para ser atendida como uma chamada local. Assim, as aplicações utilizam os arquivos locais e remotos de uma forma totalmente transparente.

Ufo oferece três formas diferentes de especificar os nomes dos arquivos. A primeira é a notação URL, usada geralmente pelas aplicações que acessam arquivos em servidores HTTP e FTP. Como muitas aplicações não suportam a notação URL, Ufo oferece uma segunda notação, chamada de notação regular, cuja sintaxe desta notação é */protocol/user@host/filename*. A última notação oferecida por Ufo é baseada em pontos de montagem (*mount points*). Usando esta notação, os usuários podem definir pontos de montagem em vários servidores, para facilitar futuros acessos aos arquivos contidos nesses servidores. Para criar um ponto de montagem, a aplicação tem que executar uma instrução específica, como por exemplo: *local /csftp remote / machine ftp.cs.ucsb.edu method FTP*. Esta notação quer dizer que acessos relativos a */csftp* estão se referindo ao diretório raiz

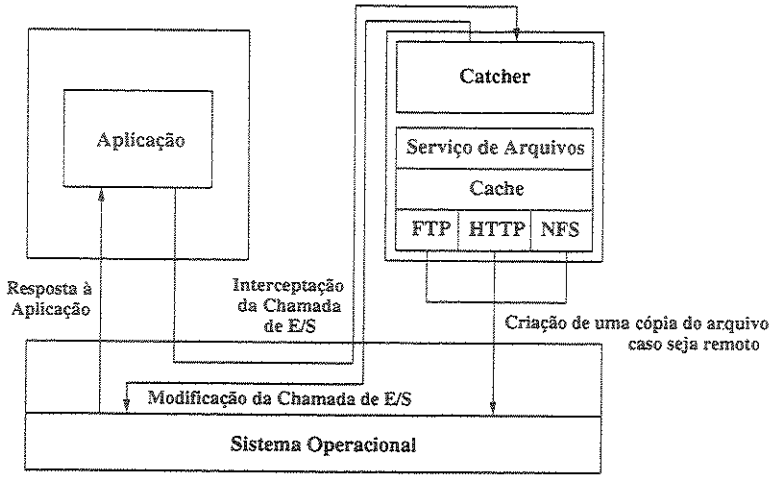

Figura A.10: Arquitetura geral de Ufo

do *host* ftp.cs.ucsb.edu, usando o protocolo FTP. Todos estes pontos de montagem são armazenados em uma tabela. Fica claro que Ufo oferece um espaço de nomes não uniforme e dependente de localização.

Como já foi explicado, quando um usuário utiliza um arquivo, Ufo cria uma cópia no cache local. Está cópia é usada em futuros acessos ao arquivo durante o tempo em que ela estiver atualizada. Se a cópia do arquivo é modificada, Ufo utiliza o mecanismo de cópia demorada (*write-back caching with delayed writing*) para enviar o arquivo modificado de volta ao servidor remoto, quando ele for fechado. Para garantir consistência, UFO utiliza uma política baseada em tempo (*timeout*). É definido um tempo máximo durante o qual a cópia no cache está atualizada para leitura (*Tread*) e um tempo máximo de espera para escrever um arquivo modificado de volta ao servidor (*Twrite*). Estes tempos garantem que uma vez copiado o arquivo no cache, ele estará atualizado por um tempo *Tread* segundos, e que modificações feitas no arquivo serão escritas de volta ao servidor *Twrite* segundos depois de fechado o arquivo. Além disso, estes tempos são determinados pelas aplicações para adaptá-los a suas necessidades (os tempos podem ser zero). Como Ufo somente oferece acesso de leitura nos servidores HTTP e acesso de leitura e escrita nos servidores FTP, este mecanismo pode ser suficiente para obter níveis de consistência adequados para as suas necessidades.

O principal problema de Ufo é a questão do desempenho. A interceptação de chamadas de sistemas gera uma sobrecarga às vezes inaceitável no sistema operacional, o que limita a ampla utilização do mesmo. Além disso, este modelo não funciona corretamente com alguns tipos de aplicações, como por exemplo aplicações que fazem muitas chamadas ao serviço de arquivos do sistema operacional. Ufo somente foi implementado para plataformas Unix (especificamente Solaris), o que torna difícil sua portabilidade.

Por Ufo estar implementado quase completamente no nível do usuário, a segurança depende dos acessos dos seus usuários ao sistema operacional, neste caso o Unix, o que não introduz novos problemas. O usuário é autenticado no seu primeiro acesso aos servidores

e as chaves dos usuários são armazenadas localmente. O único problema é que os arquivos armazenados no cache podem ser utilizados por outros usuários que normalmente não têm acesso ao mesmo. Para resolver este problema, Ufo também implementa, de forma similar a Unix, permissões de acesso para estes arquivos.

Em resumo, Ufo é um sistema baseado em Unix que utiliza mecanismos de interceptação de chamadas de sistema para oferecer acesso a arquivos remotos através dos protocolos HTTP e FTP. Ufo, apesar da sobrecarga imposta pela interceptação de chamadas de sistema, obtém um bom desempenho mediante o uso intensivo de cache local nos computadores usuários. No entanto, apesar de Ufo poder ser uma opção para usuários independentes, ele só é implementado em plataformas Unix (Solaris), fazendo difícil seu uso em sistemas de processamento maciçamente paralelo baseado na Internet.

A.14 Jade

Jade [Rao91] centra sua atenção em dois dos problemas mais importantes a serem resolvidos por um sistema de arquivos distribuído: a resolução de nomes e o acesso remoto a recursos através da rede. Para isto, Jade adota um modelo simples, oferecendo um sistema lógico que integra uma coleção de sistemas de arquivos físicos já existentes, ou seja, cada um dos sistemas de arquivos mantém sua independência. Este modelo é heterogêneo, já que permite o uso de diferentes protocolos de comunicação entre os servidores e os clientes. Os protocolos suportados são os protocolos NFS, AFS, FTP e UFS (*Unix File System*). Além disso, Jade adota a abordagem de que nenhum dos sistemas de arquivos que formam esse sistema lógico precise de modificação em seus programas ou em suas formas de administração.

Jade usa uma estrutura hierárquica para mostrar o espaço de nomes aos seus usuários. Cada usuário define seu espaço de nomes, criando uma coleção de pequenos espaços de nomes orientados a cada usuário, ao invés de um espaço de nomes global e uniforme para todo o sistema. Ou seja, cada usuário tem um visão somente do espaço de nomes por ele definido e todas as resoluções de nomes são realizadas no contexto desse espaço de nomes. Cada usuário escolhe os sistemas de arquivos físicos aos quais ele quer ter acesso e forma seu sistema de arquivos privado lógico. Este sistema de arquivos lógico do usuário pode ser dividido em múltiplos domínios, cada um deles implementado por um sistema de arquivos físico diferente. Jade suporta a operação de montagem, que monta um determinado sistema de arquivos físico no espaço de nomes lógico dos usuários. Jade permite que um mesmo sistema de arquivos físico possa ser montado em vários sistemas de arquivos de Jade, cada vez em um lugar diferente, o qual depende da forma como os usuários criaram seu espaço de nomes. Para manter a independência dos sistemas de arquivos físicos, nenhuma informação sobre este processo de montagem é mantida por eles; esta é uma tarefa do sistema de arquivos de Jade.

Jade usa o que ele chama de *skeleton directory* [RP93]. Cada um destes diretórios mantém uma lista de referências e cada referência identifica um ponto de montagem no sistema de arquivos. Estas referências estão formadas da maneira mostrada a seguir.

```
<Protocol, Server, Handle, Token>
```

Protocol: identifica o protocolo usado para se ter acesso ao sistema de arquivos montado, por exemplo NFS, AFS, UFS ou FTP.

Server: especifica o *host* que oferece os serviços do sistema de arquivos montado.

Handle: é o descritor usado pelo servidor para identificar a raiz do sistema de arquivos montado.

Token: provê informação de autenticação.

Jade permite montagens simples e múltiplas, assim como montagem de sistemas de arquivos lógicos definidos por outros usuários. A Figura A.11 mostra exemplos destes três tipos de montagens. Além disso, Jade mantém listas de *tokens* nos usuários. Cada *token* nessa lista está formado pelo nome do usuário (*login name*) e uma chave de autenticação (*password*). Desta forma, o sistema de arquivos implementa seu mecanismo de segurança.

Montagem simples

{ <NFS, meg.cs.arizona.edu, /usr/john/jade> }

Montagem múltipla

{ <UFS, jag.cs.arizona.edu, /usr/john/bin>
<NFS, meg.cs.arizona.edu, /usr/john/bin>
<UFS, jag.cs.arizona.edu, /usr/bin> }

Figura A.11: Tipos de montagem de sistemas de arquivos físicos em Jade

Para o acesso a arquivos remotos Jade faz uso intensivo do cache. Cada usuário pode escolher qual dos sistemas de arquivos físicos vai ser seu servidor de cache. Esta escolha é feita baseada na proximidade dos possíveis servidores. Note que o próprio disco do usuário é considerado em Jade como um sistema de arquivos e pode ser utilizado como servidor de cache. A principal vantagem desta abordagem é que qualquer computador pode usar estes mecanismos de cache, mesmo se não tem disco para manter as cópias dos arquivos que utiliza. Além disso, Jade também permite a troca dinâmica dos servidores de cache, aumentando a flexibilidade do modelo de cache do sistema de arquivos. Quando um arquivo é aberto, Jade procura no cache uma cópia válida do arquivo. Se a cópia existe, ela é utilizada; caso contrário, uma cópia atualizada do arquivo é procurada no sistema de arquivos apropriado. Todas as operações de leitura e escrita no arquivo são realizadas sobre o cache, nunca diretamente sobre o arquivo original, o que diminui o tráfego na rede. Jade oferece mecanismos de consistência fracos que dependem de como foram implementados em cada um dos sistemas de arquivos físicos. Por exemplo, técnicas como *callbacks* não podem ser usadas, já que não são suportadas por todos os protocolos utilizados (somente o AFS suporta técnicas de *callbacks*). A Figura A.12 mostra mais detalhes do mecanismo utilizado.

Jade implementa toda esta camada lógica entre os sistemas de arquivos físicos e os seus usuários através de dois componentes principais, o *Name Space Manager* e o *Access Manager*. O *Name Space Manager* provê um serviço de diretórios que mapeia o nome lógico

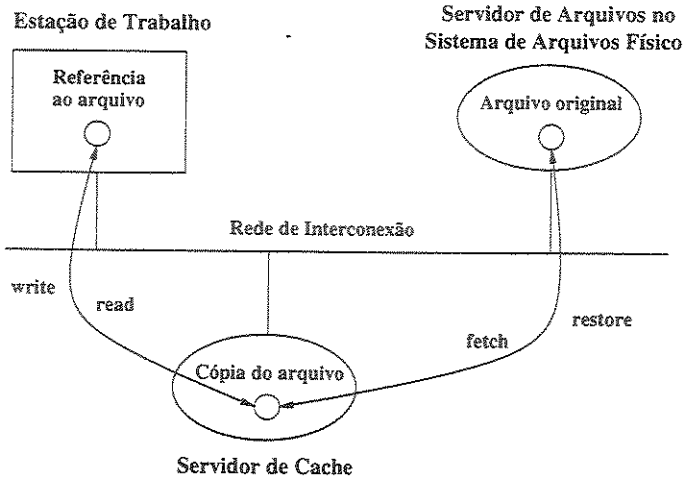

Figura A.12: Modelo de acesso aos dados em Jade

fornecido pelo usuário em uma referência ao arquivo no sistema de arquivos apropriado. Uma vez obtida a referência ao arquivo, o *Access Manager* permite o acesso ao arquivo criando uma cópia completa do arquivo no servidor de cache do usuário. Como Jade integra sistemas de arquivos físicos diferentes em um sistema lógico único, cada um deles com um protocolo de acesso diferente, é definida uma interface uniforme para acomodar todos estes protocolos. Jade implementa um agente para cada protocolo de acesso diferente que mapeia cada função definida nessa interface uniforme em operações para cada protocolo de acesso em particular. Desta forma, os sistemas de arquivos que formam a coleção não precisam fazer modificação alguma nos seus *kernels*. Esta abordagem de implementar o sistema de arquivos no usuário permite uma maior portabilidade de Jade, mas o desempenho é menor em comparação com abordagens baseadas em *kernel*.

Concluindo, Jade é um sistema de arquivos que implementa o serviço de diretórios (obter a referência de um arquivo pelo seu nome lógico) de forma completamente independente, tanto do ponto de vista do projeto como da implementação do serviço de acesso aos arquivos, o que dá maior flexibilidade ao sistema. Além disso, a implementação no nível de usuário de todos os seus mecanismos permite uma maior portabilidade. No entanto, ele não oferece um espaço de nomes global e uniforme para todos os usuários do sistema de arquivos e os mecanismos de consistência dos caches são de responsabilidade dos sistemas de arquivos físicos, herdando seus problemas. O mesmo acontece com a segurança. Jade não implementa seu próprio mecanismo de segurança, deixando-o por conta dos sistemas de arquivos usados. Estas desvantagens fazem com que Jade não seja adequado para uma arquitetura de processamento maciçamente paralelo.

A.15 IFS

IFS (*Internet File System*) [RC95] é um sistema que visa aumentar o alcance dos sistemas de arquivos de redes locais, permitindo o acesso destes sistemas pela Internet. Para satis-

fazer seus objetivos, IFS oferece aos usuários um modelo para compartilhar informação de diferentes *hosts* na Internet através dos protocolos mais usados atualmente. IFS adota a abordagem usada em Unix de que "tudo é arquivo" para manipular os recursos da Internet. Desta forma, ele consegue tratar os recursos disponíveis na teia mundial da mesma maneira como são tratados os demais recursos mantidos pelo sistema de arquivos. Como resultado da aplicação deste modelo, os usuários de IFS podem programar e usar os recursos da Internet mediante o uso da API, dos comandos e dos utilitários oferecido pelo sistema de arquivos. IFS é implementado usando como base o n-DFS (*Multiple Dimensional File System*) [FKR94, KK90b], sistema de arquivos que permite agregar novos serviços a um sistema de arquivos já existente. Este sistema de arquivos é uma camada lógica entre o sistema operacional e as aplicações dos usuários, que intercepta as chamadas de sistema e as envia ao serviço correspondente. Entre os serviços mais importante oferecidos por n-DFS estão: serviço de nomes e semântica [GJSO91], serviço de monitoramento das operações do sistema operacional e das comunicações, e serviço de replicação e sincronização de réplicas entre sistemas de arquivos [RS95].

A Figura A.13 mostra a arquitetura geral de IFS. O sistema está formado por dois componentes fundamentais: o *Shared Library* e o*Access Server*. O *Shared Library* (biblioteca compartilhada) intercepta as chamadas de sistema feitas pelas aplicações e, antes de enviá-las ao *kernel* do sistema operacional, comunica-se com o *Access Server* para obter o descritor do arquivo desejado. Alguns sistemas operacionais oferecem facilidades de ligação dinâmica de bibliotecas compartilhadas. Portanto, as aplicações podem usar o *Shared Library* sem modificações. No entanto, nem todos os sistemas operacionais oferecem estas facilidades, o que faz necessário que todas aplicações tenham que ser modificadas para poder fazer uso dos recursos desta biblioteca. O *Access Server* seleciona o protocolo adequado, comunica-se com o servidor remoto através da Internet, pega o arquivo e o armazena no cache do sistema de arquivos local. IFS implementa um agente diferente para cada um destes protocolos. A comunicação entre os componentes do IFS é realizada usando IPC (*Inter Process Communication*).

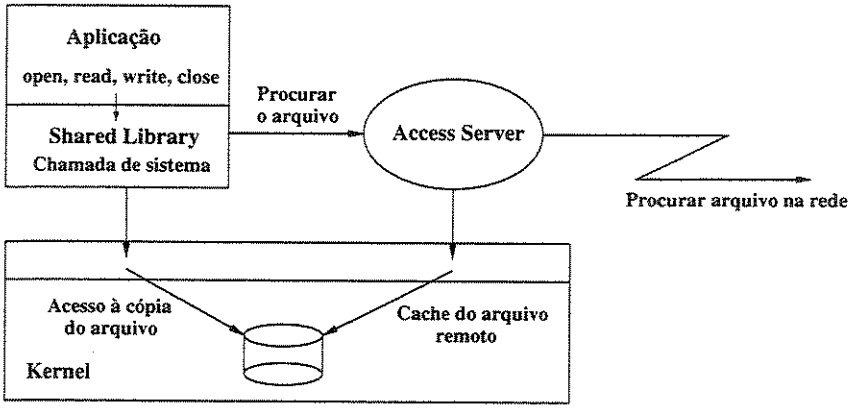

Figura A.13: Arquitetura geral de IFS

IFS utiliza a notação URL para referir-se aos arquivos na Internet. No entanto, IFS

não implementa um espaço de nomes global para todo o sistema. Ele permite que os usuários gerenciem seus arquivos em um espaço de nomes pessoal e privado, chamado de PNS (*Personal Name Space*). Cada usuário monta seu PNS em um nó do sistema de arquivos local. A estrutura de PNS é mostrada a seguir.

```
<MountPoint, CacheDirectory>
```

MountPoint: ponto dentro do sistema de arquivos local onde o PNS é montado.

CacheDirectory: diretório local onde as cópias dos arquivos utilizados são armazenadas.

Para ter acesso aos arquivos nos servidores remotos, IFS utiliza vários dos protocolos mais usados atualmente na Internet. Ele implementa os protocolo FTP, HTTP, Gopher, NNTP (*Network News Transfer Protocol*) [KL86] e RSH (*Remote Shell*) [Ylo95]. Estes protocolos de acesso são divididos em dois grupos, os protocolos de acesso orientados a sistemas de arquivos e os protocolos não orientados a sistemas de arquivos. Os protocolos FTP e RSH entram na primeira categoria; HTTP, Gopher e NNTP entram na segunda.

Similar a outros sistemas de arquivos sobre a Internet, IFS faz uso intensivo de cache. Os arquivos usados pelas aplicações são copiados completamente no cache do sistema de arquivos local quando eles são abertos. IFS implementa uma semântica de sessão. Depois de aberto um arquivo, todas as leituras e escritas são realizadas na cópia contida no cache, sem necessidade de se comunicar com o servidor. Quando o arquivo é fechado, o arquivo é mandado de volta para o servidor. IFS não suporta controle de sincronização entre diferentes cópias de um mesmo arquivo.

Por último, IFS é um sistema de arquivos seguro. Ele utiliza o programa *F-Secure SSH* [FKN+95, Fel97] para tratar o problema da segurança. Basicamente, o SSH é um mecanismo de transporte que prove autenticação tanto dos servidores como dos usuários, além de criptografar toda a informação transmitida pelos seus canais. IFS também suporta a tecnologias chamada de *TCP/IP port forwarding technology*, que permite fazer conexões seguras sobre canais inseguros, o que aumenta a confiabilidade e a segurança do sistema.

Resumindo, IFS é um sistema de arquivos que visa expandir a capacidade dos sistemas de arquivos para que possam obter acesso aos recursos da Internet. Embora o sistema ofereça acesso aos arquivos através dos protocolos mais usados na Internet e use a notação URL para referir-se a eles, a implementação é baseada na utilização de bibliotecas compartilhadas, o que, em muitos casos, torna necessário fazer modificações nas aplicações para que elas possam utilizar os recursos oferecido pelo sistema. Além disso, embora seja usada a notação URL, IFS não oferece um espaço de nomes global e uniforme para todas as aplicações, já que os usuários gerenciam os seus arquivos em um espaço de nomes pessoal e privado, o que dificulta a sua utilização em um sistema de processamento maciçamente paralelo que visa utilizar os computadores da Internet como um computador virtual.

A.16 Comparação entre os Sistemas de Arquivos Distribuídos

A tabela a seguir mostra uma comparação entre os diferentes sistemas de arquivos distribuídos, destacando os pontos fortes e fracos de cada um deles. Os aspectos a comparar são citados a seguir. Eles foram enumerados para facilitar a apresentação dos dados.

1.- Tipo de rede onde é usado (LAN, WAN, Internet)
2.- Instalação e configuração (Simples, Média, Complexa)
3.- Forma de uso (Simples, Médio, Complexo)
4.- Portabilidade (Sim, Não)
5.- Desempenho (Alto, Médio, Baixo)
6.- Disponibilidade da informação (Alta, Média, Baixa)
7.- Consistência e coerência da informação (Sim, Não)
8.- Uso de cache nos clientes (Sim, Não)
9.- Tolerante a falhas (Sim, Não)
10.- Reconfiguração automática ante falhas (Sim, Não)
11.- Escalabilidade (Sim, Não)
12.- Espaço de nomes global e uniforme (Sim, Não)
13.- Confiabilidade (Alta, Média, Baixa)
14.- Controle de acesso (Modo de bits de Unix, Kerberos, Access Control List)
15.- Gerenciamento (Simples, Médio, Complexo)
16.- Tipo de servidores (stateLess, stateFull)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
NFS	L	S	S	S	A	A	N	S	N	N	N	N	A	M-K	M	L
AFS	L	S	S	N	A	A	S	C	N	N	N	S	A	K-A	S	F
DFS	L	C	C	N	A	A	S	T	N	S	S	S	A	M-A	S	F
Novell	L	C	S	S	A	A	S	N	S	S	S	S	A	A	S	L
NTFS	L	M	S	S	A	A	S	N	S	S	S	S	A	A	S	L
Coda	W	C	M	N	A	A	S	N	S	S	N	S	A	K-A	M	F
Bayou	W	M	S	N	M	A	N	N	S	S	S	N	M	A	M	L
Sprite	W	M	S	N	A	A	S	S	N	N	N	S	A	A	M	F
Ficus	W	S	S	N	M	A	N	N	S	S	N	S	A	A	S	L
WebNFS	I	S	S	S	A	A	N	S	N	N	S	N	A	M-K	M	L
WebFS	I	C	S	N	A	A	S	S	N	N	N	S	A	A	S	F
Ufo	I	S	S	N	M	A	S	S	S	S	S	S	A	A	M	L
Jade	I	S	S	S	A	A	S	S	S	N	S	N	B	A	S	L
IFS	I	S	M	N	A	M	N	N	S	S	S	N	A	A	M	L

Tabela A.1: Comparação entre os sistemas de arquivos distribuídos estudados

Apêndice B

Diagrama de Classes

A seguir são mostrados os diagramas das classes do SAD_GSA, obtidas com o uso da versão em domínio público da ferramenta *Poseidon for UML* [AG02], baseada no projeto de código aberto ArgoUML [Rob02]. Os diagramas apresentados são os seguintes.

- Diagrama da distribuição das classes em pacotes
- Diagrama de classes do pacote filesystem.kernel
- Diagrama de classes do pacote filesystem.utils.netutils
- Diagrama de classes do pacote filesystem.utils.fsutils
- Diagrama de classes do pacote filesystem.utils.ziputils
- Diagrama de classes do pacote filesystem.commands
- Diagrama de classes do pacote filesystem.distributed.server
- Diagrama de classes do pacote filesystem.distributed.coordinator
- Diagrama de classes do pacote filesystem.distributed.graphical
- Diagrama de classes do pacote filesystem.distributed.application
- Diagrama de classes do pacote filesystem.distributed.worker

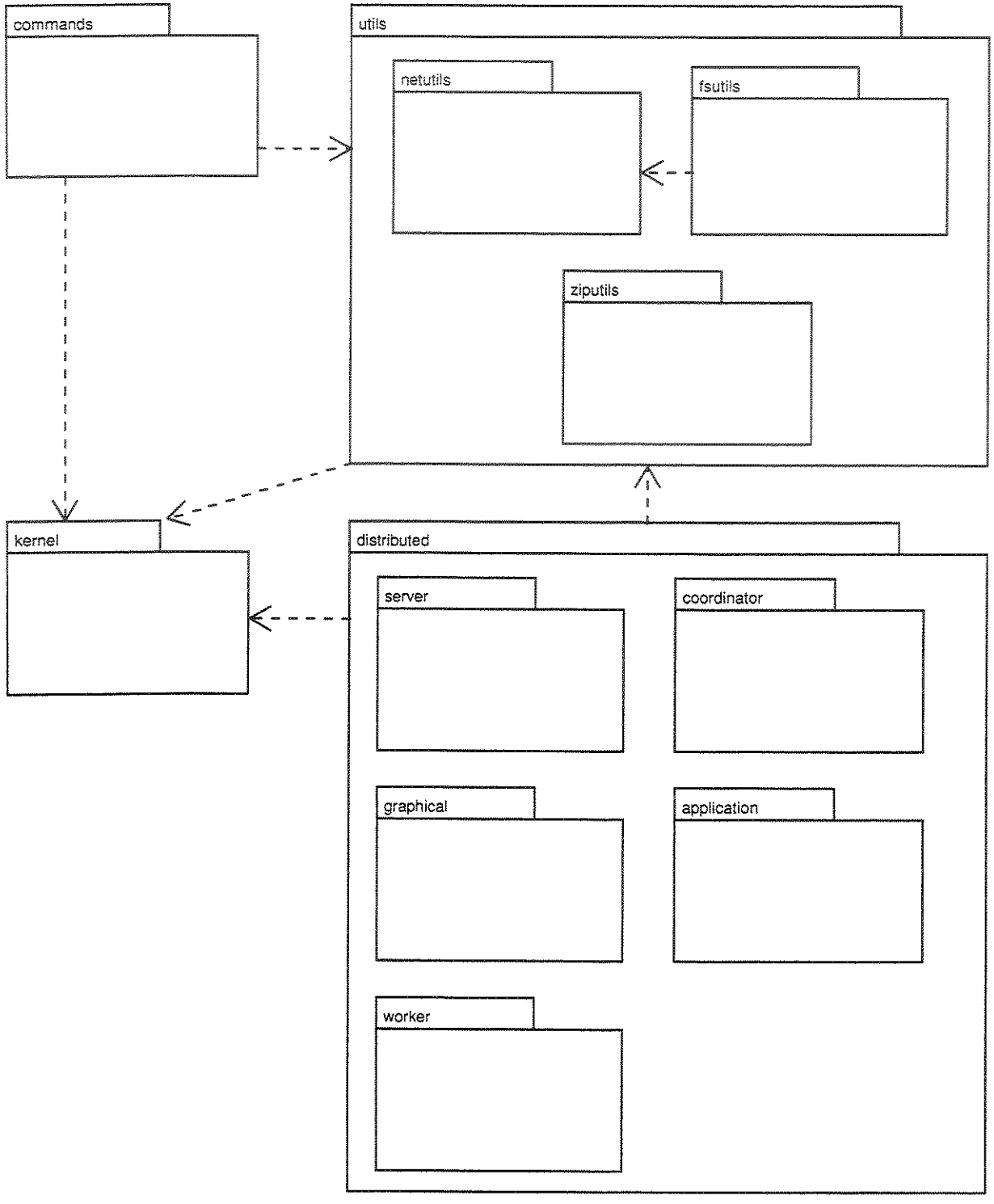

Figura B.1: Distribuição das classes em pacotes

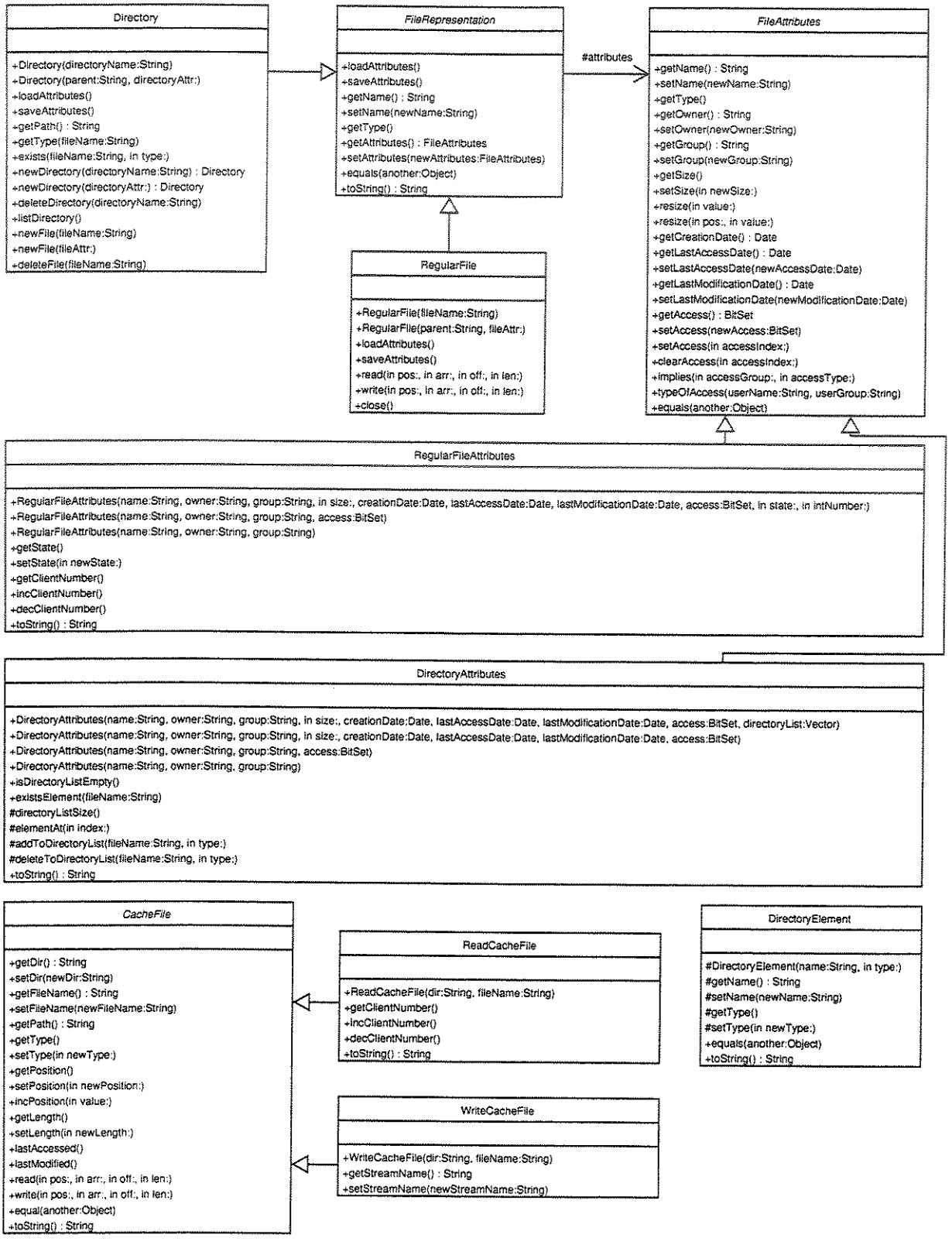

Figura B.2: Classes do pacote filesystem.kernel

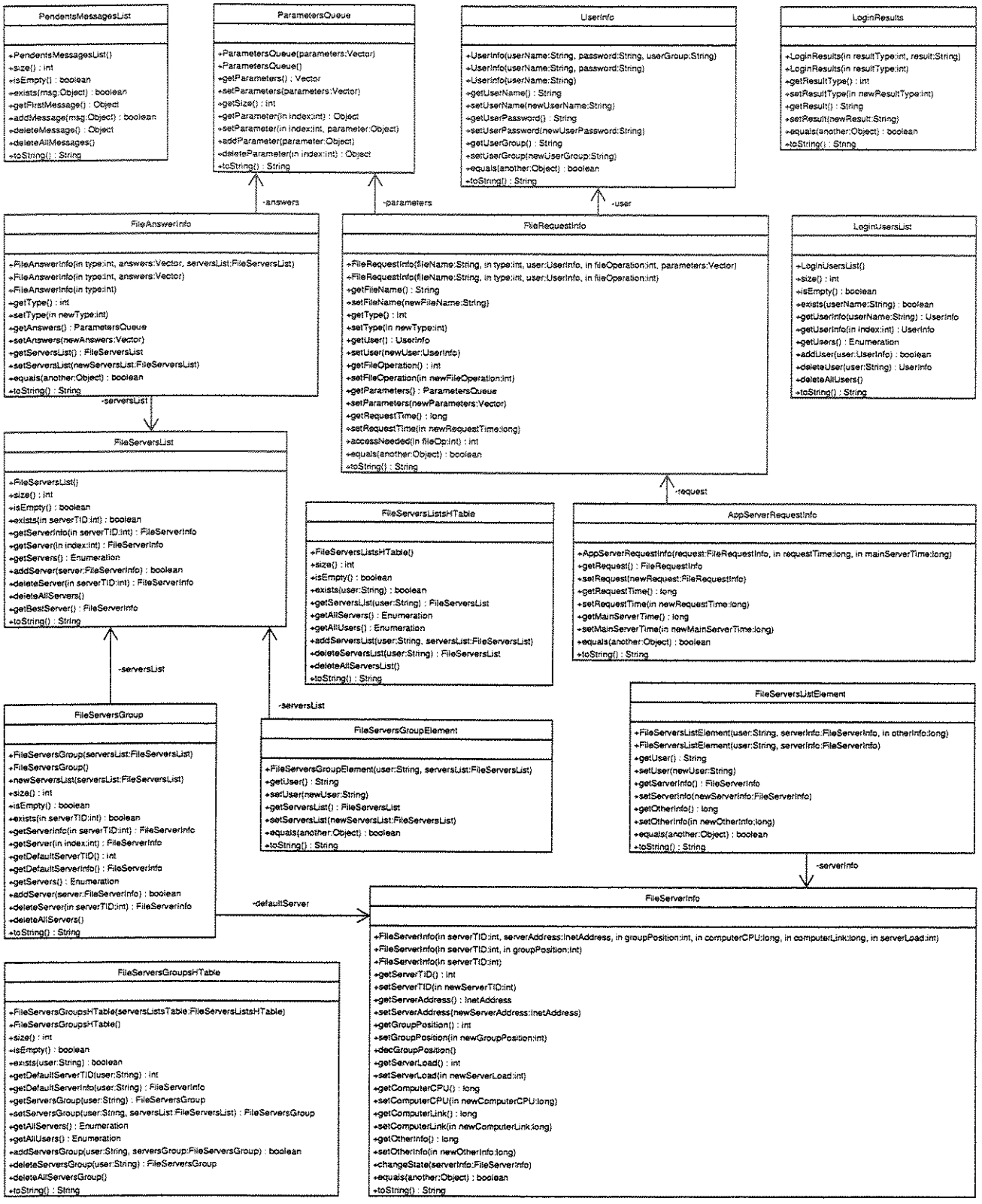

Figura B.3: Classes do pacote filesystem.utils.netutils

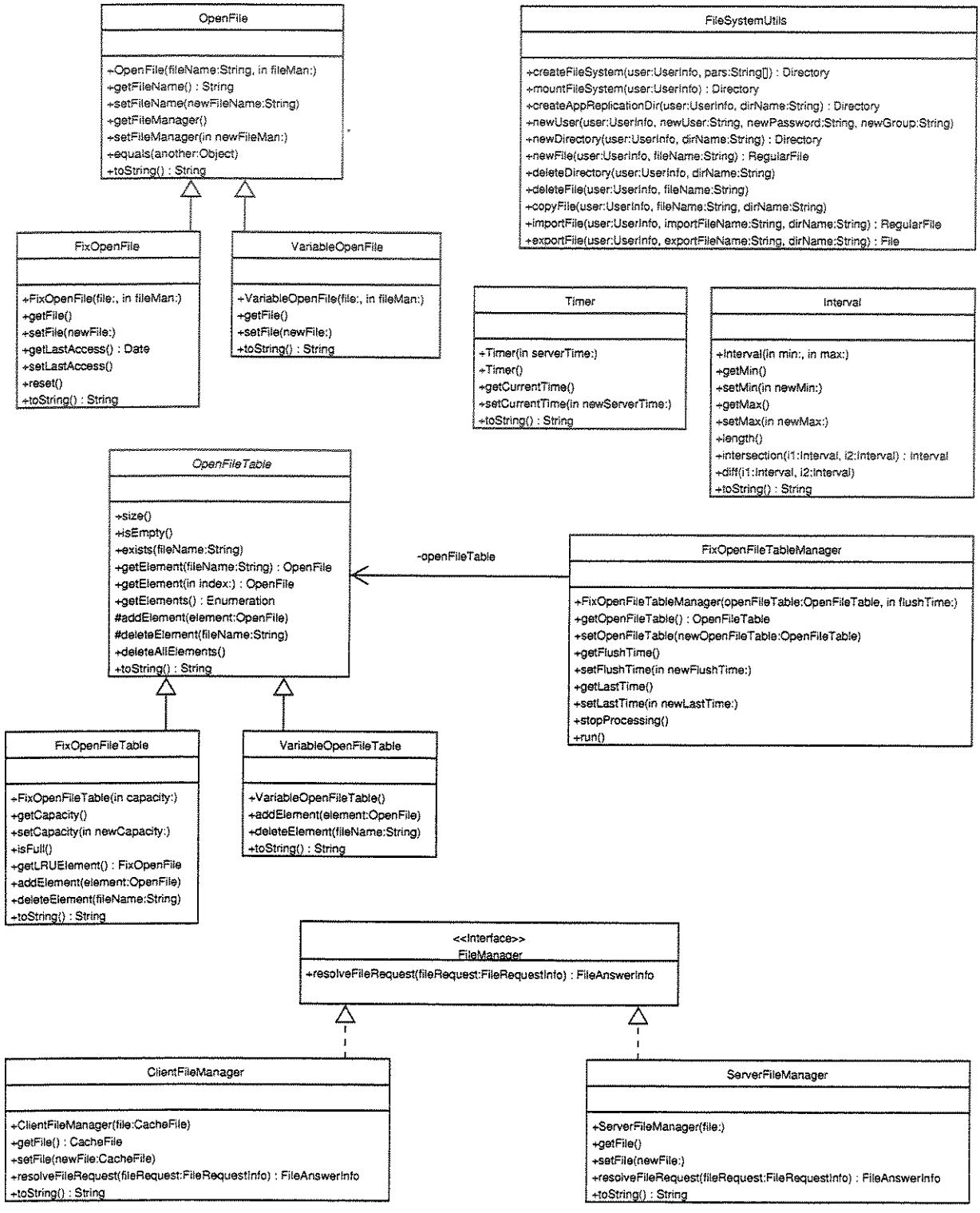

Figura B.4: Classes do pacote filesystem.utils.fsutils

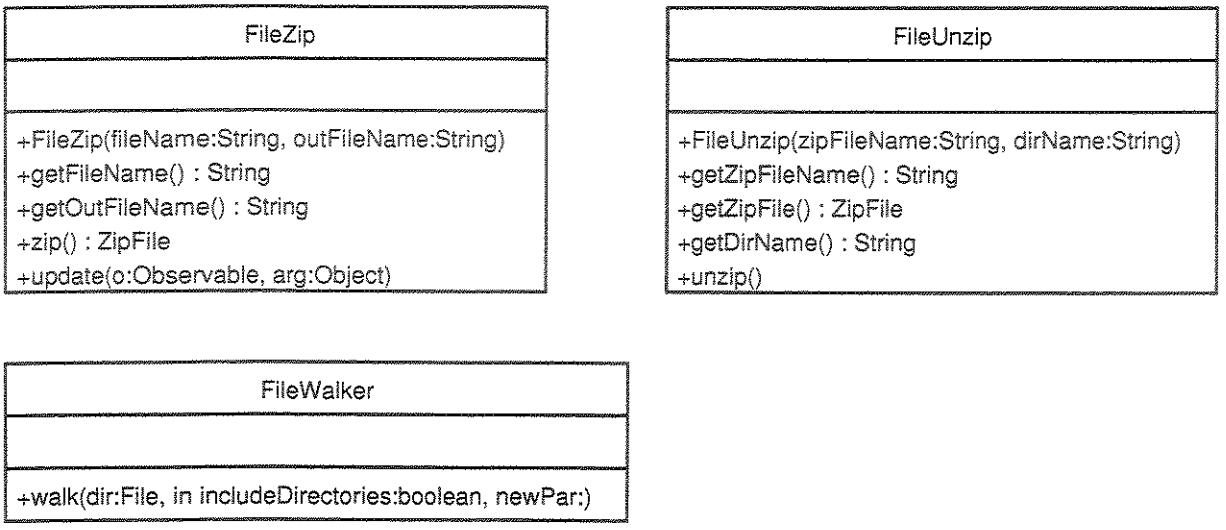

Figura B.5: Classes do pacote filesystem.utils.ziputils

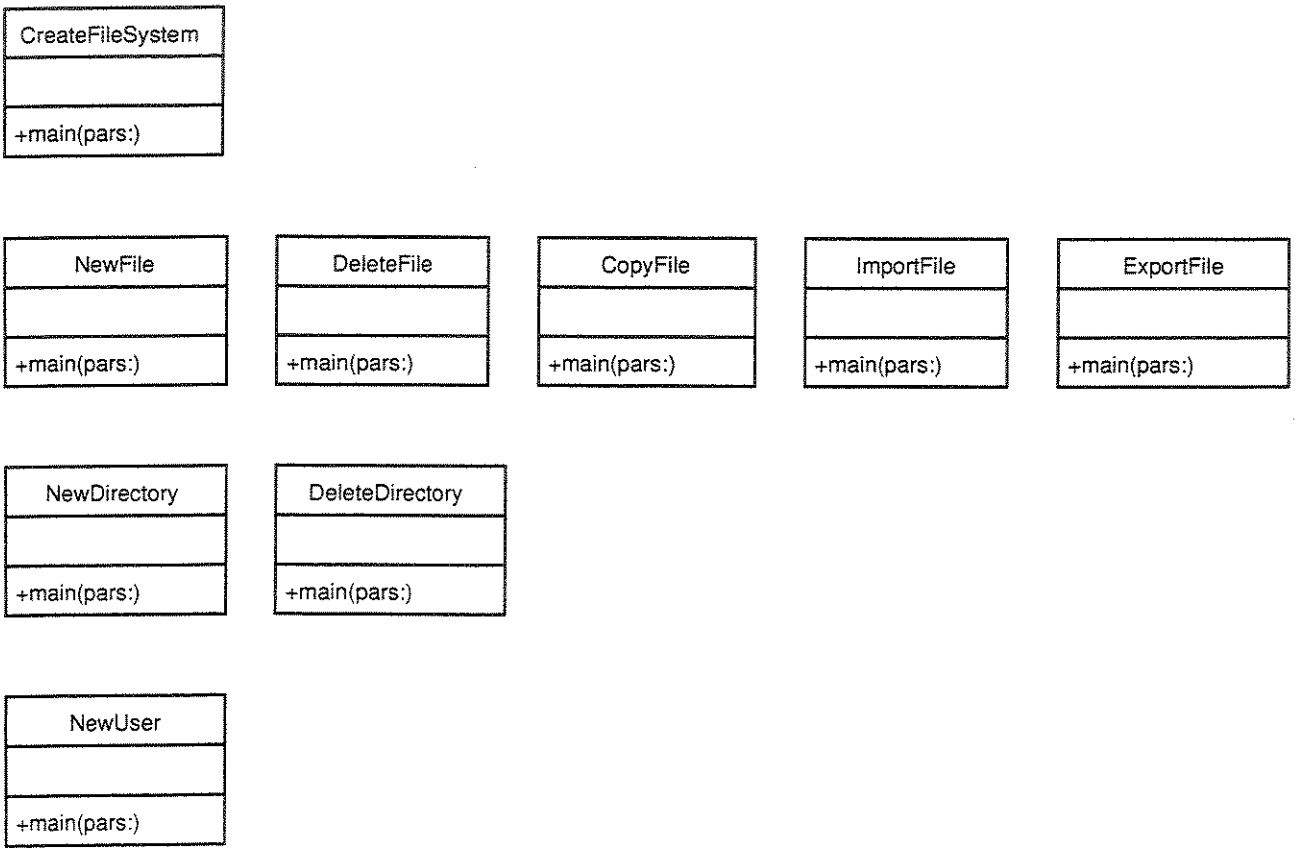

Figura B.6: Classes do pacote filesystem.commands

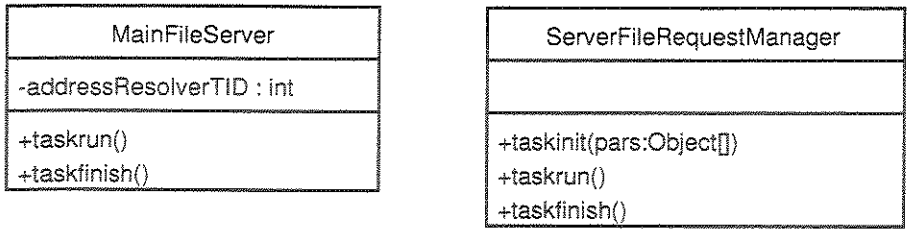

Figura B.7: Classes do pacote filesystem.distributed.server

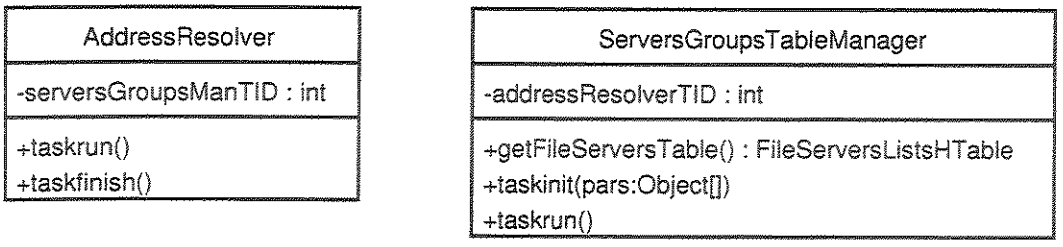

Figura B.8: Classes do pacote filesystem.distributed.coordinator

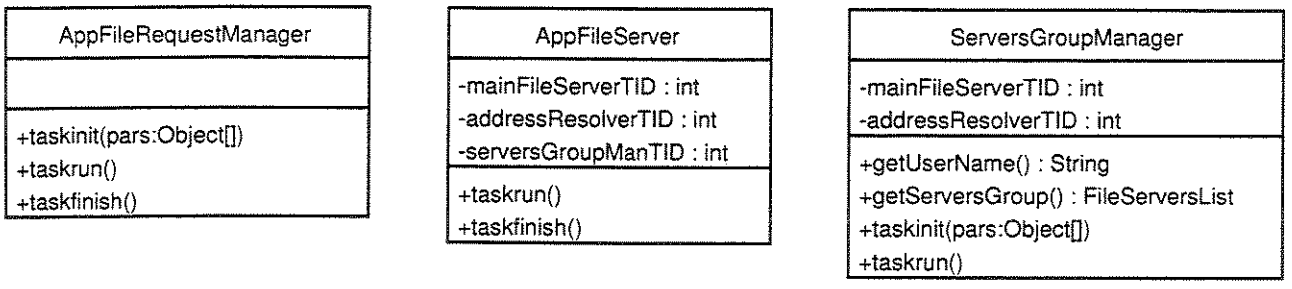

Figura B.9: Classes do pacote filesystem.distributed.graphical

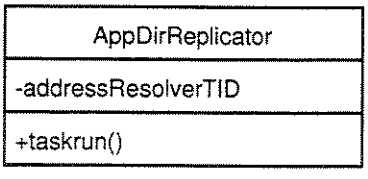

Figura B.10: Classes do pacote filesystem.distributed.application

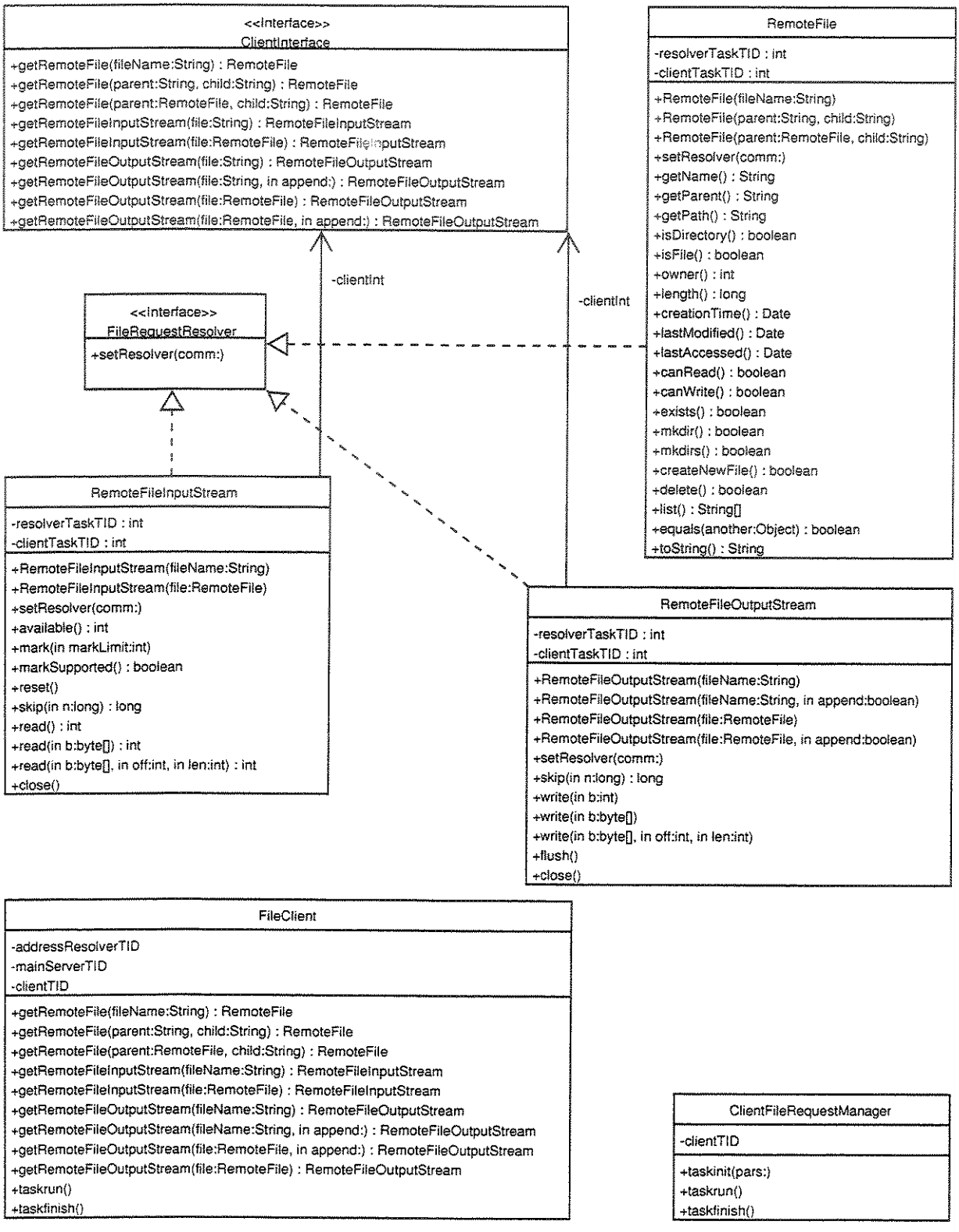

Figura B.11: Classes do pacote filesystem.distributed.worker

Apêndice C

Interface do Sistema de Arquivos

A seguir é mostrada a interface do SAD_GSA. Esta interface é usada pelas tarefas das aplicações paralelas para obter acesso aos serviços oferecidos pelo sistema de arquivos. O Componente dos Trabalhadores do serviço de arquivos do JOIN implementa esta interface.

```
public interface ClientInterface{

  public RemoteFile getRemoteFile(String fileName);
  public RemoteFile getRemoteFile(String parent, String child);
  public RemoteFile getRemoteFile(RemoteFile parent, String child);

  public RemoteFileInputStream getRemoteFileInputStream(String file)
                               throws java.io.IOException;
  public RemoteFileInputStream getRemoteFileInputStream(RemoteFile file)
                               throws java.io.IOException;

  public RemoteFileOutputStream getRemoteFileOutputStream(String file)
                                throws java.io.IOException;
  public RemoteFileOutputStream getRemoteFileOutputStream(String file,
                                boolean append) throws java.io.IOException;
  public RemoteFileOutputStream getRemoteFileOutputStream(RemoteFile file)
                                throws java.io.IOException;
} // interface ClientInterface
```

Além desta interface, o sistema de arquivo oferece três classes principais para manipular os arquivos. Estas classes oferecem a funcionalidade necessária para manipular arquivos e *streams*. As mesmas são mostradas a seguir.

```
public class RemoteFile{

    public RemoteFile(String fileName);
    public RemoteFile(String parent, String child);
    public RemoteFile(RemoteFile parent, String child);

    public String getFileName();
    public String getParent();
    public String getPath();

    public boolean isDirectory();
    public boolean isFile();
    public int owner() throws java.io.IOException;
    public long length() throws java.io.IOException;
    public Date creationTime() throws java.io.IOException;
    public Date lastModified() throws java.io.IOException;
    public Date lastAccessed() throws java.io.IOException;
    public boolean canRead() throws java.io.IOException;
    public boolean canWrite() throws java.io.IOException;
    public boolean setActualDirectory() throws java.io.IOException;

    public boolean exists();
    public void createNewFile() throws java.io.IOException;
    public void mkdir() throws java.io.IOException;
    public void mkdirs() throws java.io.IOException;
    public void delete() throws java.io.IOException;
    public String[] list() throws java.io.IOException;

    public boolean equals(Object another);
    public String toString();
} // class RemoteFile
```

```
public class RemoteFileInputStream extends InputStream{

  public RemoteFileInputStream(String file) throws java.io.IOException;
  public RemoteFileInputStream(RemoteFile file)
         throws java.io.IOException;

  public int available() throws IOException;
  public boolean markSupported();
  public void mark(int markLimit);
  public void reset() throws IOException;
  public long skip(long n) throws IOException;

  public int read() throws java.io.IOException;
  public int read(byte[] b) throws java.io.IOException;
  public int read(byte[] b, int off, int len)
             throws java.io.IOException;

  public void close() throws IOException;
} // class RemoteFileInputStream


public class RemoteFileOutputStream extends OutputStream{

  public RemoteFileOutputStream(String file) throws java.io.IOException;
  public RemoteFileOutputStream(String file, boolean append)
         throws java.io.IOException;
  public RemoteFileOutputStream(RemoteFile file)
         throws java.io.IOException;

  public long skip(long n) throws IOException;

  public void write(int b) throws java.io.IOException;
  public void write(byte[] b) throws java.io.IOException;
  public void write(byte[] b, int off, int len) throws
  java.io.IOException;

  public void flush() throws IOException;
  public void close() throws IOException;
} // class RemoteFileOutputStream
```

Apêndice D

Publicação Derivada deste Trabalho

Evento: II Simpósio de Informática do Planalto Médio
Universidade de Passo Fundo. Rio Grande do Sul
Maio de 2000

Título: Um Sistema de Arquivos Distribuído baseado na Internet

Autores: Juan Carlos Hidalgo Costa [jcarlos@dca.fee.unicamp.br]
Marco Aurélio Amaral Henriques [marco@dca.fee.unicamp.br]

Instituição: Departamento de Engenharia de Computação e Automação Industrial (DCA)
Faculdade de Engenharia Elétrica e de Computação (FEEC)
UNICAMP

Endereço Postal: DCA/FEEC/UNICAMP
Caixa Postal 6101
CEP: 13083-970
Campinas, SP

Resumo:

Os computadores conectados pela Internet, a maior rede de computadores do mundo, oferecem em conjunto um poder de cômputo enorme. Eles podem ser vistos como um Computador Maciçamente Paralelo Virtual com memória distribuída, que pode ser usado na resolução de problemas de grande porte. Existem várias propostas que visam tirar proveito da Internet como um computador virtual, utilizando Java como linguagem independente de plataforma e baseadas na ampla disponibilidade de navegadores para WWW. Entretanto, a maior parte destes projetos não trata, ou trata de forma superficial, a necessidade de se ter um sistema de arquivos que garanta a viabilidade e eficiência dos mesmos. Este trabalho propõe um Sistema de Arquivos Distribuído baseado em Grupos de Servidores para poder ser utilizado por plataformas deste tipo. São propostos mecanismos para

atender os requisitos fundamentais de sistemas deste tipo, tentando eliminar as principais deficiências dos sistemas de arquivos atuais. Também são mostrados os testes realizados com um protótipo, que refletem os resultados obtidos nas implementações realizadas. Como plataforma de referência e de testes do sistema de arquivos é utilizado JoiN, um sistema paralelo virtual baseado em máquinas heterogêneas conectadas pela Internet.

Referências Bibliográficas

[AG94] George S. Almasi e Allan Gotlieb. *Highly Parallel Computing.* The Benjamin/Cummings Publishing Company, Inc., segunda edição, 1994.

[AG02] GentleWare AG. Poseidon for uml, community edition 1.2, 2000, 2001, 2002. Disponível em: http://www.gentleware.com.

[AISS97a] A. D. Alexandrov, M. Ibel, K. E. Schauser, e C. J. Scheiman. Superweb: Towards a global web-based parallel computing infrastructure. In *11th International Parallel Processing Symposium*, Abril de 1997.

[AISS97b] Albert D. Alexandrov, Maximilian Ibel, Klaus E. Schauser, e Chris J. Scheiman. Ufo: A personal global file system based on user-level extensions to the operating system. In *In Proceedings of the USENIX Technical Conference*, Anaheim, California, Janeiro de 1997.

[AL94] Paul Albitz e Cricket Liu. *DNS and BIND.* O'Reilly and Associates, 1994.

[AML+93] F. Anklesaria, M. McCahill, P. Lindner, D. Johnson, D. Torrey, e B. Alberti. The internet protocol: A distributed document search and retrieval protocol. Request For Comment 1436, Março de 1993. University of Minnesota.

[BBB96] Eric Baldeschwieler, Robert Blumofe, e Eric Brewer. Atlas: An infrastructure for global computing. In *Proceedings of the Sevent ACM SIGOPS European Workshop: Systems Support for Worldwide Applications*, Setembro de 1996.

[BJK+95] Robert D. Blumofe, C. F. Joerg, B. C. Kuszmaul, C. E. Leiserson, K. H. Randall, e Y. Zhou. Cilk: An efficient multithreaded runtime system. In *Proceedings of the Fifth Symposium on Principles and Practice of Parallel Programming*, 1995.

[BKKMK98] Arash Baratloo, Mehmet Karaul, Holger Karl, e Zvi M. Kedem. An infrastructure for network computing with java applets. In *Proceedings of the ACM 1998 Workshop on Java for High-Performance Network Computing*, Standford University, Palo Alto, California, Fevereiro de 1998.

[BLCL+94] T. Berners-Lee, R. Calliau, A. Luotonen, H. Frystyk Nielsen, e A. Secret. The world-wide web. *Communications of the ACM*, 37(8), Agosto de 1994.

[BLFF96] T. Berners-Lee, R. Fielding, e H. Frystyk. Hypertext transfer protocol - http/1.0. Internet Engineering Task Force (IETF), Fevereiro de 1996. Disponível em: htpp://www.w3.org/pub/WWW/Protocols/HTTP/spec.ps.

[BLMM94] T. Berners-Lee, L. Masinter, e M. McCahill. Uniform resource locators (url). Request For Comment 1738, Dezembro de 1994. Network Working Group. Disponível em: http://www.w3.org/pub/WWW/Addressing/rfc1738.txt.

[BN84] A. D. Birrell e B. J. Nelson. Implementating remote procedure calls. *ACM Transactions on Computer System*, 2(1):39–59, Fevereiro de 1984. Association for Computing Machinery.

[BO91] M. Baker e J. Ousterhout. A viability in the sprite distributed file system. *ACM Operating System Review*, 25(2):95–98, Abril de 1991.

[BP88] B. N. Bershad e C. B. Pinkerton. Watchdogs - extending the unix file system. *Computing Systems*, 1(2):169–188, Agosto de 1988.

[BSST96] Tim Brecht, Harjinder Sandhu, Meijuan Shan, e Jimmy Talbot. Paraweb: Towards world-wide supercomputing. In *Proceedings of the Setima ACM SIGOPS European Workshop on System Support for Worldwide Applications*, 1996.

[BVAD98] E. Belani, A. Vahdat, T. Anderson, e M. Dahlin. The crisis wide area security architecture. In *In Proceedings of the USENIX Security Symposium*, San Antonio, Texas, Janeiro de 1998.

[Cala] B. Callaghan. Webnfs client specification. Request For Comments 2054. Disponível em: ftp://ftp.isi.edu/in-notes/rfc2054.txt.

[Calb] B. Callaghan. Webnfs server specification. Disponível em: ftp://ftp.isi.edu/in-notes/rfc2054.txt.

[Car99] Alan R. Carter. *Windows NT 4.0 Workstation MCSE Study System.* IDG Books Worldwide, Setembro de 1999.

[CCI+97] P. Cappello, B. O. Christiansen, M. F. Ionescu, M. O. Neary, K. E. Schauser, e D. Wu. Javelin: Internet-based parallel computing using java. In Geoffrey C. Fox e Wei Li, editores, *ACM Workshop on Java for Science and Engineering Computation*, Junho de 1997.

[CD93] J. Cohen e L. David. Afs: Nfs on steroids. *LAN Technology*, Março de 1993.

[CDK94] G. Colouris, J. Dollimore, e T. Kindberg. *Distributed Systems - Concepts and Desing*. Addison-Wesley, segunda edição, 1994.

[CFK+98] Karl Czajkowski, Ian Foster, Carl Kesselman, Nicholas Karonis, Stuart Martin, Warren Smith, e Steven Tuecke. A resource management architecture for metacomputing systems. In *Proceedings of the IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing*, 1998.

[DPS+95] Alan Demers, Karin Petersen, Mike Spreitzer, Douglas Terry, Marvin Theimer, e Brent Welch. The bayou architecture: Support for data sharing among mobile users. In *In Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, Dezembro de 1995.

[DWAP94] M. Dahlin, R. Wang, T. Anderson, e D. Patterson. Cooperative caching: Using remote client to improve file system performance. In *In Proceedings of the 1st USENIX Symposium on Operating System Desing and Implementation*, pp. 267–280, Novembro de 1994.

[ELD+96] D. H. J. Epenema, M. Livny, R. Van Dantzig, X. Evers, e J. Pruyne. A worldwide flock of condors: Load sharing among workstations clusters. *Journal of Future Generation Computer Systems*, (12):53–65, 1996.

[Eve90] C. F. Everhart. Conventions for names in the service directory in the afs distributed files system. Technical report, Transarc Corporation, Março de 1990.

[Fel97] Data Fellows. F-secure shh user's and administrator's guide. Data Fellows, Julho de 1997. Disponível em: http://www.DataFellows.com.

[Fer98] Adam J. Ferrari. Jpvm: Network parallel computing in java. In *Proceedings of the ACM 1998 Workshop on Java for High-Performance Network Computing*, Standford University, Palo Alto, California, Fevereiro de 1998.

[FF97] Geoffrey C. Fox e Wojtek Furmanski. Computing on the web. new approache in parallel processing. petaop and exaop performance in the year 2007. *IEEE Internet Computing*, 1(2), Março de 1997.

[FK98] Ian Foster e Carl Kesselman. The globus project: A status report. In *Proceedings of the IPPS/SPDP '98 Heterogeneous Computing Workshop*, pp. 4–18, 1998.

[FKN+95] Glenn S. Fowler, David G. Korn, Stephen North, Herman C. Rao, e K. Phong Vo. *Libraries and File System Architecture*, capítulo 2. Practical Reusable UNIX Software, Outubro de 1995.

[Lam78] L. Lamport. Time, clocks and ordering of events in a distributed system. In *Proceedings of the Communications of the ACM*, volume 21, pp. 558–564–12, Julho de 1978.

[Lam79] L. Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. In *Proceedings of the IEEE Transactions on Computer*, volume C-28, pp. 690–691, Setembro de 1979.

[LG96] Mike Lewis e Andrew Grimshaw. The core legion object model. In *Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing*, Los Alamitos, California, Agosto de 1996. IEEE Computer Society Press.

[Mac94] R. Macklem. Not quite nfs, soft cache consistency for nfs. In *In Proceedings of the USENIX Association Conference*, pp. 261–278, Janeiro de 1994.

[mat02] The matrix and information directory services web site. Disponível em: http://www.mids.org/mmq/601/mid/hosts.html, 2002.

[MBKQ96] Marshall Kirk McKusick, Keith Bostic, Michael Karels, e John Quarterman. *The Desing and Implementation of the 4.4BSD Operating System*, capítulo 9. Addison-Wesley Publishing Company, Inc., 1996.

[MCF92] Maurício F. Magalhães, Eleri Cardozo, e Luis F. Faina. *Introdução aos Sistemas Operacionais.* Faculdade de Engenharia Elétrica e de Computação, 1992. Revisado por Marco A. Amaral Henriques.

[Mic98] Microsoft Corporation. *Microsoft Information Server Training Kit.* Microsoft Press, Março de 1998.

[Min00] Mark Minasi. *Mastering NT Server 4.* Sybex Inc., setima edição, Janeiro de 2000.

[Moh99] James Mohr. *Supporting Windows NT and 2000 Workstations and Server.* Prentice Hall, Dezembro de 1999.

[MSC+86] J. H. Morris, M. Satyanarayanan, M. H. Conner, J. H. Howard, D. S. Rosenthal, e F. D. Smith. Andrew: A distributed personal computing environment. *Communications of the ACM*, 29(3), Março de 1986.

[MSC+90] J. Moran, R. Sandberg, D. Coleman, J. Kepecs, e B. Lyon. Breaking through the nfs performence barrier. In *In Proceedings of the Spring UNIX Users Group*, pp. 190–206, Munich, Germany, Abril de 1990.

[Nau97] Patrick Naughton. *The Java Handbook.* Osborne McGraw-Hill, 1997.

[Nel86] M. Nelson. The sprite operating system. Technical Report 83/301, UCB/CSD, Junho de 1986.

[Net98a] Netscape Communications Corporation. Java and Security, 1998. Developer's edge archived Conference materials. Disponível em: http://developer.netscape.com/developer/conference/proceedings/j4/.

[Net98b] Netscape Communications Corporation. Netscape Object Signing. Establishing Trust for Downloaded Software , 1998. White Paper. DevEdge Online Documentation. Disponível em: http://developer.netscape.com/docs/manuals/signedobj/trust/index.htm.

[NLRC97] Noam Nisan, Shmulik London, Ori Regev, e Noam Camiel. Globally distributed computation over the internet - the popcorn project. In *WWW6 - Sixth International World Wide Web Conference*, Santa-Clara, Abril de 1997.

[Nov98a] Novell, Inc. Directory standards demystified: Ldap unlocks the power of your network, 1998. White Paper. Disponível em: http://www.novell.com/documentation/lg/nw5/docui/index.html.

[Nov98b] Novell, Inc. How does netscape directory server stack up?, 1998. White Paper. Disponível em: http://www.novell.com/documentation/lg/nw5/docui/index.html.

[Nov98c] Novell, Inc. Netware 5, 1998. White Paper. Disponível em: http://www.novell.com/documentation/lg/nw5/docui/index.html.

[Nov98d] Novell, Inc. Novell directory service (nds) in netware 5, 1998. White Paper. Disponível em: http://www.novell.com/documentation/lg/nw5/docui/index.html.

[Nov98e] Novell, Inc. Novell storage service (nss) in netware 5, 1998. White Paper. Disponível em: http://www.novell.com/documentation/lg/nw5/docui/index.html.

[Nov98f] Novell, Inc. Simplifying and securing the network, 1998. White Paper. Disponível em: http://www.novell.com/documentation/lg/nw5/docui/index.html.

[NWO88] M. Nelson, B. Welch, e J. Ousterhout. Caching in the sprite network file system. *ACM Transactions on Computer Systems*, 6(1):134–154, Fevereiro de 1988. Associotion for Computing Machinery.

[ONOCCES99] Michael O. Neary, Bernd O. Christiansen, Peter Capello, e Klaus E. Schauser. Javelin: Parallel computing on the internet. *FGCS Special Issue on Metacomputing*, 1999. Elsevier Science, Amsterdam, Netherlands.

[Par97] J. Parker. Distributed file systems, Março de 1997. Disponível em: http://www.chicago.edu/JasonParker/dfs.html.

[PGJH90] Gerald J. Popok, Richard G. Guy, Thomas W. Page Jr, e John S. Heidemann. Replication in ficus distributed file system. In *In Proceedings of the Workshop on Management of Replicated Data*, pp. 20–25, Novembro de 1990.

[PJS+94] B. Pawlowski, C. Juszczak, P. Staubach, C. Smith, D. Lebel, e D. Hitz. Nfs version 3: Desing and implementation. In *In Proceedings of the USENIX Association Conference*, pp. 137–151, Junho de 1994.

[PR91] J. Postel e J. Reynolds. File tranfer protocol (ftp). Request For Comments 959, Outubro de 1991. Information Sciences Institute, Marina del Ray, CA. Disponível em: ftp://ftp.isi.edu/in-notes/rfc959.txt.

[PZ97] Michael Philippsen e Matthias Zenger. Javaparty - transparent remote objects in java. *Concurrency: Practice and Experience*, 9(11):1225–1242, Novembro de 1997.

[Rao91] H. C. Rao. *The Jade File System.* PhD thesis, University of Arizona, Março de 1991.

[RC95] Herman Chung-Hea Rao e Ming-Feng Chen. An internet file system, Maio de 1995. Disponível em: http://atttaiwan.fareastone.com.tw/ifs.html.

[RHR+94] P. Reiher, J. Heidemann, D. Ratner, G. Skinner, e G. Popek. resolving file conflicts in the ficus file system. In *In Proceedings of the Summer USENIX Conference*, pp. 183–195, Junho de 1994.

[RN98] Ori Regev e Noam Nisan. The popcorn market - an online markets for computational resources. In *Proceedings of the First International Conference On Information and Computation Economies*, Charleston SC, 1998.

[Rob02] Jason Robbins. Argouml: A modelling tool for design using uml, 2000, 2001, 2002. Projeto OpenSource. Disponível em: http://argouml.tigris.org/.

[RP93] H. C. Rao e L. L. Peterson. Accessing files in an internet: The jade file system. *IEEE Transactions on Software Engineering*, 19(6), Junho de 1993.

[RS95] Herman C. Rao e Andrea Skarra. A transparent service for synchronized replication across loosely-connected, heterogeneous file systems. In *In Proceedings of the IEEE 2nd Workshop on Services and Networked Environments*, Junho de 1995.

[RSA78] Ron Rivest, Adi Shamir, e Len Adleman. A Method for Obtaining Digital Signatures and Public Key Criptosystems. *Communications of the ACM*, Fevereiro de 1978.

[Sat89] M. Satyanarayanan. Integrating security in a large distributed system. *ACM Transactions on Computer Systems*, 7(3), Agosto de 1989.

[SC98] David A. Salomon e Helen Custer. *Inside Windows NT.* Microsoft Press, segunda edição, Março de 1998.

[SGK+85] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, e B. Lyon. Desing and implementation of the sun network filesystem. In *In Proceedings of the USENIX Association Conference*, pp. 119–130, Junho de 1985.

[Sha98] Rawn Shah. Cable Network Ins and Outs. *SunWorld*, 12(3), Março de 1998.

[Sid86] R. N. Sidebotham. Volumes: The andrew file system data structuring primitive. In *In Proceedings of the European UNIX User Group Conference*, Agosto de 1986.

[SKK+90] Mahadev Satyanarayanan, James J. Kistler, Puneet Kumar, Maria E. Okasaki, Ellen H. Siegel, e David C. Steere. Coda: A highly available file system for a distributed workstation environment. *IEEE Transactions on Computer*, 39(4):447–459, Abril de 1990.

[SNS88] J. Steiner, C. Neuman, e J. Schiller. Kerberos: An authentication service for open network systems. In *In Proceedings of the USENIX Association Conference*, pp. 191–202, Fevereiro de 1988.

[SOHL+96] Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker, e Jack Dongarra. *MPI: The Complete Reference.* The MIT Press, 1996.

[SS96a] M. Satyanarayanan e M. Spasojevic. Afs and the web: Competitors or collaborators? In *Proceedings of the 7th ACM SIGOPS European Workshop on System Support for Worldwide Applications*, Julho de 1996.

[SS96b] M. Spasojevic e M. Satyanarayanan. An empirical study of a wide-area distributed file system. *ACM Transactions on Computer Systems*, 14(2), Maio de 1996.

[Sta95] William Stallings. *Network and Internetwork Security Principles and Practice.* Prentice Hall, 1995.

[Sun89] Sun Microsystems. Nfs: Network file system protocol specifications. Request For Comments 1094, Março de 1989.

[Sun96] Sun Microsystems. Webnfs: The filesystem for the world wide web. Technical report, Sun Microsystems, 1996. Disponível em: http://www.sun.com/solaris/networking/webnfs/webnfs.html.

[Sun97a] Sun Microsystem, Inc. The filesystem for the internet, Abril de 1997. Disponível em: http://www.sun.com/software/white-papers/wp-webnfs/.

[Sun97b] Sun Microsystems. Secure computing with java: Now and the future. JavaOne Conference, 1997. White Paper.

[Sun97c] Sun Microsystems. Secure Computing with Java: Now and the Future. JavaOne Conference, 1997. White Paper.

[Tan95] Andrew S. Tanenbaum. *Distributed Operating Systems.* 07458. Prentice Hall, Inc., Upper Saddle River, New jersey, quarta edição, 1995.

[Tan96] Andrew S. Tanenbaum. *Computers Networks.* Prentice-Hall, Inc., terceira edição, 1996.

[TEAtNt95] David A. Patterson Thomas E. Anderson, David E. Culler e the NOW team. A case for now (networks of workstations). *IEEE Micro*, 15(1), Fevereiro de 1995.

[TL95] Todd Tannenbaum e Michael Liztkow. The condor distributed processing system. *Dr. Dobb's Journal*, pp. 40–48, Fevereiro de 1995.

[Tob89] R. Tobbicke. Distributed file systems: Focus on andrew file system/distributed file system (afs/dfs). In *In Proceedings of the 13th IEEE Symposium on Mass Storage System*, USA, 1989.

[TTP+95] Douglas B. Terry, Marvin M. Theimer, Karin Petersen, Alan J. Demers, Mike J. Spreitzer, e Carl H. Hauser. Managing update conflicts in bayou, a weakly connected replicated storage system. In *In Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, pp. 172–183, Dezembro de 1995.

[VA98] A Vahdat e T. Anderson. Transparent result caching. In *In proceedings of the USENIX Technical Conference*, New Orleans, Louisiana, Junho de 1998.

[VAD+96] A. Vahdat, T. Anderson, M. Dahlin, D. Culler, E. Belani, P. Eastham, e C. Yoshikawa. Webos: Operating system service for wide area applications. In *In Proceedings of the Setima Symposium on High Performance Distributed Computing*, Julho de 1996. WebOS Project. Disponível em: http://now.cs.berkeley.edu/WebOS/publications.shtml.

[VDA96] Amin Vahdat, Michael Dahlin, e Thomas Anderson. Turning the web into a computer. Technical draft, University of Berkeley, Maio de 1996. WebOS Project. Disponível em: http://now.cs.berkeley.edu/WebOS/publications.shtml.

[VEA96] A. Vahdat, P. Eastham, e T. Anderson. Webfs: A global cache coherent file system, Dezembro de 1996. Disponível em: http://www.cs.berkeley.edu/vahdat/webfs/webfs.html.

[Wel86] B. B. Welch. The sprite remote procedure call system. Technical report, UCB/ Computer Sciences Departament. University of California, Berkeley, Junho de 1986.

[Wel90] B. B. Welch. *Naming, State Management and User-Level Extensions in the Sprite Distributed File System.* PhD thesis, University of California, Berkeley, Agosto de 1990.

[Wel92] Brent Welch. A comparison of the vnode and sprite file system architectures. In *In Proceedings of the USENIX File System Workshop*, pp. 29–44, Maio de 1992.

[WLS+85] D. Walsh, B. Lyon, G. Sager, J. Chang, D. Goldberg, S. Kleiman, T. Lyon, R. Sandberg, e P. Weiss. Overview of the sun network file system. In *In Proceedings of the USENIX Association Conference*, pp. 117–124, Janeiro de 1985.

[WO86] B. B. Welch e J. K. Ousterhout. Prefix tables: A simple mechanism for locating files in a distributed filesystem. In *In Proceedings of the 6th ICDCS*, pp. 184–189, Maio de 1986.

[WO88] B. B. Welch e J. K. Ousterhout. Pseude-devides: User-level extensions to the sprite file system. In *In Proceedings of the Summer USENIX Conference*, pp. 184–189, Junho de 1988.

[YCE+97] Chad Yoshikawa, Brent Chun, Paul Eastham, Amin Vahdat, Thomas Anderson, e David Culler. Using smart client to build scalable services. *In Proceedings of the USENIX Summer Conference*, Janeiro de 1997.

[YH98a] Eduardo Javier Huerta Yero e Marco Aurélio Amaral Henriques. Um sistema para o processamento massivamente paralelo na world wide web. Dissertação. Departamento de Engenharia da Computação e Automação Industrial (DCA), Faculdade de Engenharia Elétrica e Computação (FEEC), Universidade Estadual de Campinas (UNICAMP), Brasil, Agosto de 1998.

[YH98b] Eduardo Javier Huerta Yero e Marco Aurélio Amaral Henriques. Uma introdução a JOIN: Um sistema de processamento paralelo baseado na World Wide Web. Technical Report RT-DCA 02/98, Faculdade de Engenharia Elétrica, Universidade Estadual de Campinas, Março de 1998.

[Ylo95] T. Ylonen. The ssh (secure shell) remote login protocol. Technical report, 1995. Disponível em: http://www.cs.hut.fi/ssh/RFC.