CENTRO UNIVERSITÁRIO FEEVALE

LUCAS GRAEBIN

ESTUDO COMPARATIVO DE SISTEMAS DE ARQUIVOS DISTRIBUÍDOS

Novo Hamburgo, Junho de 2007.

LUCAS GRAEBIN

ESTUDO COMPARATIVO DE SISTEMAS DE ARQUIVOS DISTRIBUÍDOS

Centro Universitário Feevale

Instituto de Ciências Exatas e Tecnológicas

Curso de Ciência da Computação

Trabalho de Conclusão de Curso

Orientador: Vandersilvio da Silva

Co-orientador: Reynaldo Cardoso Novaes

Novo Hamburgo, Junho de 2007.

RESUMO

A constante onipresença das redes propiciou um meio de comunicação comum entre grande parte dos usuários de microcomputadores. Este canal de transmissão facilitou o processo de armazenamento e compartilhamento de arquivos entre usuários através de serviços como o *Peer-to-Peer* (P2P) de forma mais segura em relação aos dispositivos móveis de pouca confiabilidade utilizados até então. Empresas de renome, como a IBM, Microsoft e Google, e grupos de pesquisa em Universidades vêm investindo na pesquisa e no desenvolvimento de ferramentas que objetivam o armazenamento distribuído de arquivos. Estes projetos, individualmente, são direcionados à resolução de problemas específicos, que vão desde o reaproveitamento de espaço ocioso em discos até a possibilidade de o cliente trabalhar com seus arquivos tendo sua estação de trabalho desconectada da rede. Um sistema de arquivos distribuído, quando bem projetado, possibilita o acesso a arquivos armazenados em um servidor com desempenho e confiabilidade semelhantes aos arquivos armazenados em computadores locais. Entretanto, sua implementação traz como obstáculo os principais desafios existentes na área de sistemas distribuídos, a exemplo da replicação e da transparência. Este trabalho aborda um estudo de conceitos, desafios e problemas existentes com a implementação de sistemas de arquivos distribuídos. O objetivo é identificar, com a aplicação de experimentos a serem elaborados neste trabalho, o grau da transparência de replicação, transparência de acesso e transparência de localização existente nas ferramentas a serem selecionadas. Em paralelo, serão realizadas simulações para mensurar a eficiência destas ferramentas em processos de escrita e leitura.

Palavras-chave: Sistemas distribuídos, sistemas de arquivos distribuídos, armazenamento distribuído, *peer-to-peer*.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AFS	*Andrew File System*
AVSG	*Accessible Volume Storage Group*
CORBA	*Common Object Request Broker Architecture*
DNS	*Domain Name System*
ISO	*International Organization for Standardization*
NFS	*Network File System*
P2P	*Peer-to-Peer*
RFC	*Requests For Comments*
RPC	*Remote Procedure Call*
URL	*Universal Resource Locator*
USB	*Universal Serial Bus*
VFS	*Virtual File System*
VSG	*Volume Storage Group*

SUMÁRIO

INTRODUÇÃO

Por muito tempo, os discos flexíveis eram os únicos meios para o compartilhamento e transferência de dados entre usuários de computadores pessoais. Embora seu uso fosse trivial, a disciplina usual era de possuir uma segunda cópia do disco para assegurar a disponibilidade dos dados e evitar perdas. Devido a sua instabilidade, baixa capacidade e taxa de transferência, os discos flexíveis passaram a ser vistos como uma alternativa pouco confiável. Com o passar do tempo, surgiram alternativas de armazenamento mais confiáveis, tais como o CD. Apesar de poder ser lido em grande parte dos microcomputadores, a manipulação dos dados ali armazenados era inconveniente, pois dependia de um dispositivo de gravação não presente em todos os equipamentos. Com o advento das tecnologias *Universal Serial Bus* (USB) e IEEE-1394 (também conhecido como *Firewire*), o mercado de mídia removível foi revitalizado. Entretanto, apesar de sua robustez e elevado grau de confiabilidade em relação aos discos flexíveis, estes dispositivos não garantem a total disponibilidade dos dados, sendo vulneráveis a problemas eletrônicos, furtos e perdas. Devido a estas inseguranças, os usuários tiveram que aderir a serviços disponíveis na Internet para o armazenamento e transferência de arquivos.

Com a crescente oferta de largura de banda e a redução acentuada de seu custo para utilização dos meios de transmissão, os protocolos HTTP, FTP e serviços *Peer-to-Peer* (P2P) (DABEK et al., 2001) tornaram-se alternativas de baixo custo para o armazenamento e transferência de arquivos entre computadores pessoais. Apesar destas técnicas, estes ambientes, em sua maioria, não garantem a disponibilidade dos dados, sendo vulneráveis a problemas de hardware e software. Alguns analistas prevêem que, no futuro, os dados que hoje são armazenados localmente, residirão em servidores existentes na Internet (ROUSH, 2006). Diversas empresas oferecem serviços que exploram esta forma de armazenamento, como por exemplo, o BeInSync (BEINSYNC, 2007) e o MediaMax da Streamload (MEDIAMAX, 2007). Nestes casos, os problemas de disponibilidade são tratados com o uso

de uma infra-estrutura própria da empresa. Além dos exemplos acima, empresas de grande porte, como a Microsoft, oferecem serviços como o FolderShare (FOLDERSHARE, 2007) e possuem pesquisas nesta área (ROWSTRON, 2001) o que demonstra o potencial deste tipo de solução em um futuro próximo.

Apesar de existirem soluções proprietárias para o repositório de arquivos, diversos esforços baseados em Software Livre e com iniciativas em outras direções, como por exemplo, o reaproveitamento de espaço ocioso em discos entre os computadores de uma rede (BECK, 2003), vem sendo direcionados para esta área.

Para ser bem sucedido, o armazenamento distribuído deve tratar um dos grandes desafios dos sistemas distribuídos que é a transparência. De acordo com Coulouris, a transparência é definida como “o encobrimento do usuário e do programador de aplicação da separação dos componentes de um sistema distribuído, de modo que o sistema seja percebido como um todo ao invés de uma coleção de componentes independentes” (COULOURIS, 2005, p. 23). Para o armazenamento distribuído dos arquivos, a transparência é obtida oferecendo ao usuário um modelo de armazenamento que estende o tradicional conceito de armazenamento presente nos computadores domésticos.

Um sistema de arquivos distribuído é a implementação de um sistema de arquivos cujos locais de armazenamento estão dispersos fisicamente, mas provê ao usuário uma visão centralizada, semelhante aos sistemas de arquivos locais (CHOW, 1998). Silberschatz complementa que o sistema de arquivos distribuído é composto de clientes, servidores e dispositivos de armazenamento dispersos na rede, onde não há um único repositório de arquivos, mas sim diversos, sendo eles independentes (SILBERSCHATZ, 2000). Dessa forma, pode-se resumir que os sistemas de armazenamento distribuído constituem de uma camada de abstração para a realização de tarefas como escrita e leitura em discos rígidos espalhados fisicamente na rede.

Os mais importantes desafios na construção de sistemas distribuídos estão presentes quando se implementa sistemas de arquivos distribuídos e devem ser levados em conta de acordo com o ambiente a que se destinam. Estes aspectos são a já citada transparência, a resolução de nomes, o gerenciamento de replicações e a segurança (CHOW, 1998). O grau de importância destes conceitos na implementação dos sistemas de arquivos distribuídos, além de depender do ambiente, recebem tratamentos diferentes entre os autores da área de sistemas

distribuídos. De acordo com Singhal, os dois serviços de maior importância presentes nos sistemas de arquivos distribuído são o servidor de nomes e o gerenciador de *cache* (SINGHAL, 1994). Estes serviços são, respectivamente, responsáveis pelo mapeamento de arquivos e diretórios, e pelo armazenamento de cópias dos dados de usuários em seus computadores (SINGHAL, 1994). Chow, por sua vez, ressalta a importância da dispersão e da multiplicidade, onde múltiplos clientes dispersos fisicamente podem acessar diversos arquivos residentes em servidores também espalhados em diversas localizações (CHOW, 1998).

A diversidade de abordagens em trabalhos teóricos, como citado acima, também tem reflexos nas implementações existentes atualmente. O resultado disso pode ser observado na implementação de diversos sistemas para armazenamento distribuídos que vêm sendo desenvolvidos e mantidos nos últimos anos, cada um deles apresentando a resolução de um problema específico.

Este trabalho tem por objetivo fazer uma revisão bibliográfica dos conceitos, desafios e problemas gerados com a implementação de sistemas distribuídos e sistemas de arquivos distribuídos. Na segunda parte deste trabalho, serão realizados estudos e experimentos em sistemas de arquivos distribuídos para identificar o grau da transparência de replicação, transparência de acesso e transparência de localização existente nestas ferramentas, além de mensurar a eficiência destas soluções em processos de escrita e leitura de arquivos, fazendo assim um comparativo com um sistema de arquivos local.

Além desta introdução, o trabalho esta organizado em mais quatro capítulos. No primeiro, será feita uma revisão nos conceitos, desafios e problemas existentes no desenvolvimento de sistemas distribuídos. No segundo capítulo, será feita uma revisão na área de armazenamento distribuído, abrangendo seus conceitos, desafios e problemas. No capítulo três, será apresentado a arquitetura de algumas das principais ferramentas de sistemas de arquivos distribuídos. No quarto capítulo, será apresentado o ambiente construído, os cenários elaborados e os resultados obtidos com os experimentos e as simulações. Por fim, serão apresentadas as considerações finais.

1 SISTEMAS DISTRIBUÍDOS

As redes estão presentes em todos os lugares. As primeiras evidências de redes de computadores apareceram no início da década de 60 com o projeto e implementação de sistemas centralizados, contendo um grande número de terminais espalhados (KIRNER, 1988). A partir de então, a evolução tecnológica, possibilitando uma redução nos custos de comunicação e de *hardware*, aliada ao avanço no estudo de redes, propiciou as condições favoráveis ao surgimento das redes de comunicação de computadores, proporcionando o compartilhamento de recursos entre máquinas e o aumento da confiabilidade e desempenho em redes locais (KIRNER, 1988). Inicialmente, foi dada ênfase a problemas simples, como a troca de mensagens entre usuários de diferentes máquinas e o compartilhamento de impressoras ou de discos rígidos apenas para leitura, entretanto, com o passar do tempo, foi-se desenvolvendo sistemas mais complexos que oferecessem ao usuário um maior aproveitamento dos recursos distribuídos pela rede (KON, 1994).

O compartilhamento de recursos é um forte motivo para a construção de sistemas distribuídos (COULOURIS, 2005). Os recursos podem ser gerenciados por servidores e acessados por clientes, ou podem ser encapsulados como objetos e acessados por outros objetos clientes (COULOURIS, 2005). O compartilhamento de recursos computacionais pode existir de forma explícita ou implícita (KIRNER, 1988). No compartilhamento explícito, a rede oferece a possibilidade dos usuários acessarem e manipularem diretamente os recursos remotos, exigindo que os clientes conheçam os detalhes dos recursos, enquanto que no compartilhamento implícito, o acesso é feito automaticamente pelo sistema de forma transparente para o usuário. Sistemas com compartilhamento de recursos de comunicação e compartilhamento implícito de recursos acabaram, em grande parte, constituindo-se nos sistemas distribuídos (KIRNER, 1988).

Várias definições para sistemas distribuídos podem ser encontradas na literatura. De acordo com Coulouris, um sistema distribuído é um “sistema cujos componentes, *software* ou *hardware*, localizado em computadores conectados a uma rede, comunicam-se e coordenam suas ações unicamente através da transmissão de mensagens” (COULOURIS, 2005, p. 2). Tanenbaum define sistemas distribuídos como sendo “[...] uma coleção de computadores independentes que aparecem para os usuários do sistema como um único computador” (TANENBAUM, 2002, p. 2). Ribeiro complementa apresentando uma definição mais detalhada:

> “Os sistemas distribuídos foram criados para distribuir as tarefas e aumentar o poder computacional através do uso de vários processadores como também promover o compartilhamento de recursos. Cada processador possui sua própria memória e a comunicação entre os processadores é feita através de linhas de comunicação. Esses sistemas objetivam melhorar a comunicação entre os computadores, propiciando a integração destes num sentido amplo, que pode envolver a facilidade de mudanças futuras, rapidez nas trocas de informações e confiabilidade na execução dos processos. Eles permitem que uma aplicação seja dividida em diferentes partes que podem ser processadas em sistemas independentes que se comunicam através das linhas de comunicação. Ele cria a ilusão de que a rede de computadores seja vista como um único sistema de tempo compartilhado, em vez de um conjunto de máquinas distintas.” (RIBEIRO, 2005, p. 32).

As diferenciações na definição de sistemas distribuídos e a conseqüente falta de clareza de algumas têm gerado controvérsias no que diz respeito à relação entre sistemas distribuídos e as redes de computadores (TANENBAUM, 2003). “A distinção fundamental é que em um sistema distribuído, uma coleção de computadores independentes mostra-se aos usuários como sendo um sistema único” (TANENBAUM, 2003, p. 2). De acordo com Verissimo, uma rede de computadores é uma infra-estrutura que conecta máquinas e adota um conjunto de protocolos de comunicação para permitir a interação entre elas, enquanto que os sistemas distribuídos são construídos sobre as redes de computadores, que hospeda serviços nas máquinas da rede e faz uso de protocolos de comunicação para suportar a execução coerente das aplicações (VERISSIMO, 2001).

Os sistemas distribuídos devem possuir, obrigatoriamente, características de multiplicidade de nós de processamento, para permitir o aumento no desempenho, tolerância a falhas e disponibilidade do sistema, e um mecanismo de comunicação para suportar a conversação entre os componentes do sistema distribuído (SPECTOR, 1981). Excepcionalmente, características como, a possibilidade de expansão, para permitir o crescimento incremental do sistema, mecanismos para detecção de erros e redundância de dispositivos, para obter disponibilidade e tolerância a falhas, e dispersão geográfica, para

permitir que os recursos sejam separados geograficamente, devem ser agregadas em seu desenvolvimento (SPECTOR, 1981).

Quando bem implementado, um sistema distribuído pode oferecer diversas vantagens em relação aos sistemas centralizados e sobre um conjunto de máquinas isoladas, tais como o compartilhamento de recursos, aumento da disponibilidade dos serviços, possibilidade de expansão da arquitetura etc (KON, 1994; COULOURIS, 2005).

1.1 Desafios de Sistemas Distribuídos

O projeto de um sistema distribuído é geralmente mais complexo em relação aos sistemas centralizados, pois seus recursos estão, muitas vezes, fisicamente dispersos, não havendo nenhum relógio comum entre os múltiplos processadores, podendo ocorrer atrasos na entrega das mensagens ou estas podem, até mesmo, serem perdidas. Apesar destas complexidades e dificuldades, os usuários devem ver um ambiente de sistema distribuído como sendo um sistema centralizado virtual, que oferece características como flexibilidade, eficiência, segurança e facilidade em seu uso (SINHA, 1996). Esta seção tem por objetivo apresentar os principais conceitos e desafios existentes na implementação de sistemas distribuídos.

1.1.1 Heterogeneidade

A heterogeneidade (isto é, variedade e diferença) pode ocorrer de várias formas nos sistemas distribuídos, variando desde a diversidade de *hardware* e diferenças em protocolos de interligação de redes até variações em gerenciadores de dados. Devido a estas distinções, a integração entre sistemas em ambientes heterogêneos é mais complexa em relação a ambientes homogêneos, onde cada sistema é baseado em uma arquitetura similar ou equivalente de *hardware* e *software* (NOTKIN, 1987). A Internet é um exemplo prático de ambiente heterogêneo, pois concede aos usuários o acesso a serviços e a execução de aplicações sobre um conjunto de computadores e redes de comunicação com características diferenciadas (COULOURIS, 2005).

A heterogeneidade é freqüentemente inevitável, podendo ser originada quando surgem necessidades de aquisição e desenvolvimento de sistemas ou equipamentos de

hardware (NOTKIN, 1987). Ela pode fazer-se presente de diferentes formas em um sistema distribuído (COULOURIS, 2005):

- *Rede:* Embora a Internet seja composta de muitos tipos de rede, suas diferenças são mascaradas pelo fato de que todos os computadores ligados a elas utilizam protocolos comuns para se comunicar. Por exemplo, um computador que possui uma placa *Ethernet* tem uma implementação dos protocolos Internet e outro computador, em um tipo diferente de rede, também possuirá uma implementação dos protocolos Internet.

- *Hardware:* Os tipos de dados, como os inteiros, podem ser representados de diversas formas em diferentes tipos de *hardware*. Existem, por exemplo, duas alternativas para a ordem em que os *bytes* de valores inteiros são armazenados: uma iniciando a partir do *byte* mais significativo e outra a partir do *byte* menos significativo. Essas diferenças na representação devem ser consideradas caso mensagens devam ser trocadas entre programas sendo executados em diferentes *hardwares*.

- *Sistemas Operacionais:* Embora os sistemas operacionais de todos os computadores na Internet precisem incluir uma implementação dos protocolos Internet, nem todos fornecem necessariamente a mesma interface de programação de aplicativos para esses protocolos. Por exemplo, as chamadas para troca de mensagens no UNIX, são diferentes das chamados no Windows.

- *Linguagens de Programação:* Diferentes linguagens de programação utilizam diferentes representações para caracteres e estruturas de dados, como *arrays* e registros. Essas diferenças devem ser consideradas, caso programas escritos em diferentes linguagens precisem se comunicar.

- *Implementação por diferentes programadores:* Os programas escritos por diferentes desenvolvedores não podem se comunicar a menos que eles utilizem padrões comuns. Para realizar a comunicação via rede entre duas aplicações é preciso que sejam estabelecidos e adotados padrões, como a exemplo dos protocolos Internet.

Não há uma única solução correta para se resolver os problemas de ambientes heterogêneos (COULOURIS, 2005). Para resolver este desafio, os sistemas distribuídos são compostos geralmente de uma camada de *software* denominada *middleware*, que é inserida logicamente entre os usuários e o nível mais elevado da aplicação para fornecer uma

abstração de programação, assim como o mascaramento da heterogeneidade das redes, do *hardware*, de sistemas operacionais e linguagens de programação subjacentes (TANENBAUM, 2002; COULOURIS, 2005), conforme ilustrado na Figura 1.1.

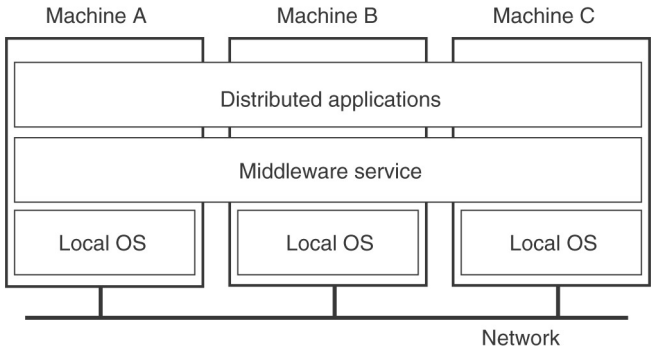

Figura 1.1 Exemplo de um sistema distribuído composto por uma camada de middleware
Fonte: Tanenbaum, 2002, p. 2

Além de resolver os problemas de heterogeneidade, o *middleware* fornece um modelo computacional uniforme para ser usado pelos programadores de serviços e aplicativos distribuídos (COULOURIS, 2005). Os modelos possíveis incluem a invocação remota de objetos, a notificação remota de eventos, o acesso remoto a banco de dados e o processamento de transações distribuídas (COULOURIS, 2005).

1.1.2 Sistemas Abertos

Diz-se que um sistema computacional é aberto quando ele pode ser estendido e reimplementado de várias maneiras (COULOURIS, 2005). O fato de um sistema distribuído ser ou não um sistema aberto é determinado pelo grau com que os novos serviços podem ser adicionados e disponibilizados para uso por uma variedade de programas clientes (COULOURIS, 2005).

A característica de sistema aberto é obtida a partir do momento em que a especificação e a documentação das principais interfaces de software dos componentes de um sistema estão disponíveis para os desenvolvedores de software, entretanto, a publicação de interfaces é apenas o ponto de partida para adicionar e estender serviços em um sistema distribuído, uma vez que o maior desafio para os projetistas é a complexidade de sistemas distribuidos compostos por muitos componentes e elaborados por diferentes pessoas (COULOURIS, 2005).

A publicação dos protocolos de comunicação Internet, os chamados *Requests For Comments* (RFC), permitiu a construção de uma variedade de sistemas e aplicativos para a Internet, incluindo a web (COULOURIS, 2005). Os sistemas projetados a partir de padrões públicos são chamados de sistemas distribuídos abertos, para reforçar o fato de que eles são extensíveis (COULOURIS, 2005). Eles podem ser ampliados em nível de *hardware*, pela adição de computadores em uma rede, e em nível de *software*, pela introdução de novos serviços ou pela reimplementação dos antigos, permitindo aos aplicativos compartilharem recursos (COULOURIS, 2005). Uma vantagem adicional, freqüentemente mencionada para sistemas aberto, é sua independência de fornecedores individuais (COULOURIS, 2005).

1.1.3 Tolerância a Falhas

Sistemas computacionais não estão livres de falhas. Quando elas ocorrem no nível de *hardware* ou *software*, os programas podem produzir resultados incorretos, ou podem ser interrompidos antes da conclusão de suas operações (COULOURIS, 2005). Um sistema é considerado tolerante a falhas quando este consegue mascarar a presença de falhas através das técnicas de redundância (JALOTE, 1994).

Para que um sistema seja confiável, ele deverá continuar funcionando corretamente mesmo com a presença de certos tipos de erros ou interferências indevidas (KIRNER, 1988). Confiança é um termo que abrange várias exigências úteis para sistemas distribuídos, como disponibilidade, confiança, segurança e sustentabilidade (MULLENDER, 1993). A disponibilidade corresponde a probabilidade de o sistema manter-se operacional dentro de certo período de tempo. A confiabilidade é o grau de sucesso que um sistema consegue atingir dentro de um período de tempo, mantendo seu comportamento dentro das especificações normais. Em contraste com a disponibilidade, a confiança é definida em termos de intervalo de tempo ao invés de um determinado momento, que é quando ocorreria a chamada do sistema pelo usuário. A segurança refere-se a situação em que nenhum efeito catastrófico venha a ocorrer originário de uma falha temporária do sistema, e a sustentabilidade, por sua vez, refere-se a possibilidade do sistema detectar fracassos e consertá-los automaticamente (MULLENDER, 1993).

Uma das maneiras para se classificar as falhas em sistemas distribuídos é baseada no comportamento do componente após o ocorrido (JALOTE, 1994). As falhas podem ocorrer a

nível de processo ou canal de comunicação e são classificadas em (COULOURIS, 2005; JALOTE, 1994):

- *Omissão:* Recorrem a casos em que um processo ou um canal de comunicação não responde a pedidos entrantes.

- *Arbitrária (ou Bizantinas):* Nestes casos, o componente comporta-se de maneira totalmente arbitrária durante a falha. Por exemplo, um processo pode ajustar os valores errados a uma variável, ou pode retornar um valor errado como resposta a uma requisição. As falhas arbitrárias não podem ser descobertas apenas vendo se o processo responde as requisições, pois se poderia arbitrariamente omitir a resposta correta retornando um valor errado como resposta a uma solicitação.

- *Temporização:* Esta falha ocorre quando um componente responde demasiadamente cedo ou demasiadamente tarde. A temporização é bastante relevante em sistemas multimídia, onde há a transferência de áudio e vídeo. Em sistemas assíncronos, pode haver uma alta latência entre o pedido do cliente e a resposta do servidor caso este esteja sobrecarregado, entretanto não se pode afirmar que ocorreram fracassos na solicitação, pois este modelo de sistema não oferece garantia de tempo de resposta.

Em sistemas tolerantes a falhas, a existência de defeitos e interferências indevidas pode ser superada através de redundância temporal, redundância de *hardware* ou redundância de *software*, técnicas geralmente utilizadas na construção de sistemas distribuídos (JALOTE, 1994). A redundância temporal corresponde a determinação de um resultado através de execuções repetidas da mesma operação pelo mesmo elemento, usando eventualmente métodos diferentes. A redundância de *hardware* compreende do acréscimo de elementos repetitivos capazes de executarem a mesma operação. A redundância de *software* inclui todos os programas e instruções que são empregadas para dar suporte à tolerância a falhas (JALOTE, 1994).

As partes redundantes do sistema podem ser utilizadas como elementos de votação ou como elementos de reserva. Atuando como elementos de votação, cada parte redundante permanecerá ativa no sistema, fornecendo seu resultado que, comparado com os outros, contribuirá para a escolha do resultado correto por maioria. Como elemento de reserva, cada parte redundante permanecerá latente até que seja acionada pelo sistema para substituir outro elemento defeituoso que estiver sendo desativado. Neste último caso, o sistema deverá possuir

um bom mecanismo para a realização de testes e maneiras eficientes de identificação de defeitos para poder localizá-los e superá-los (KIRNER, 1988).

Os sistemas tolerantes a falhas situam-se em duas classes: sistemas de alta disponibilidade e sistemas de alta confiabilidade. Enquanto que sistemas de alta disponibilidade podem tolerar pequenos atrasos decorrentes de manutenção, os sistemas de alta confiabilidade exigem que o funcionamento correto seja mantido, apesar da presença de faltas (KIRNER, 1988).

O objetivo principal da abordagem tolerante a falhas consiste em inibir o efeito das faltas, ou seja, uma interferência indevida (KIRNER, 1988), através das redundâncias do sistema. Quando estes sistemas forem submetidos à ocorrência de uma falha, eles deverão reagir executando alguns, ou todos os procedimentos a seguir (KIRNER, 1988; SIEWIOREK, 1984; COULOURIS, 2005):

- *Confinar a falta:* Consiste em limitar o alcance de seus efeitos través do uso de circuitos de detecção de faltas, verificação de consistência e precedendo certas operações, protocolos múltiplos de comunicação etc., evitando dessa forma a propagação das falhas.

- *Detectar a falta:* Corresponde a tomar conhecimento de sua existência através de *bits* de paridade, verificação de consistência, violação de protocolo etc. Essa detecção poderá ser feita com o dispositivo fora de operação ou em operação.

- *Mascarar a falta:* Tem por objetivo eliminar os seus efeitos, fazendo com que as informações redundantes se imponham sobre as informações incorretas.

- *Tentar novamente:* Este é um procedimento que dará bons resultados quando se tratar de faltas transientes, que não provoquem nenhum dano físico ao sistema, podendo ser bem sucedida em uma segunda tentativa.

- *Diagnosticar:* Esta etapa tem por objetivo identificar a localização da falta e suas propriedades, uma vez que a detecção da falta não trata desse aspecto.

- *Reconfigurar:* Consiste em substituir ou isolar as partes defeituosas. A alternativa por isolar as partes defeituosas poderá levar o sistema a uma degradação gradual. A

substituição das partes defeituosas só será possível se houverem disponíveis partes correspondentes.

- *Recuperar:* Tem a função de colocar a operação do sistema, ou de alguma de suas partes, numa situação anterior a ocorrência das falhas com o objetivo de eliminar os erros que aparecem. Isto exigirá do sistema, conhecimento de pontos de operação que possam ser restaurados. Algumas técnicas utilizadas para recuperar pontos de operação anteriores consistem em manter arquivos de *backup*, usar pontos de verificação, manipular listas de intenção etc.

- *Reiniciar:* Consiste em recolocar o sistema em funcionamento. Este processo poderá não ser realizado totalmente com sucesso caso o sistema tenha sido danificado gravemente pelo erro, ou este não possua suporte para este procedimento. O reinício "quente" corresponde a retomada de todas as operações a partir do ponto de onde ocorreu a detecção da falta. O reinício "morno" ocorrerá quando somente uma parcela do sistema estiver em condições de ser retomada. O reinício "frio" consistirá no reinício total do sistema, precedido pela recarga, se for necessária, para superar perdas de grande amplitude.

- *Reparar:* Constitui-se na atividade de consertar o componente defeituoso. Isto poderá ser realizado com o sistema fora de operação ou em operação. No caso de reparo com o sistema fora de operação, o elemento defeituoso deverá ser consertado com o sistema desligado. No caso de reparo com o sistema em operação, o elemento defeituoso deverá ser substituído por um elemento sobressalente nos moldes da reconfiguração, ou deverá ser retirado, ocasionando eventualmente uma degradação gradual no sistema.

- *Reintegrar:* Após a reparação do elemento, este deverá ser reintegrado no sistema. Para sistemas em operação, a reintegração deverá ser efetuada sem a interrupção das operações.

Os sistemas distribuídos fornecem um alto grau de disponibilidade perante a falhas de *hardware* ou *software* (COULOURIS, 2005). A disponibilidade de um sistema é a medida da proporção de tempo em que ele está pronto para uso. Quando um dos componentes de um sistema distribuído falha, apenas o trabalho que estava usando o componente defeituoso é

afetado. Um usuário pode passar para outro computador, caso aquele que estava usando falhe - um processo do servidor pode ser iniciado em outro computador (COULOURIS, 2005).

1.1.4 Segurança

Muitos recursos de informação que se tornam disponíveis e são mantidos em sistemas distribuídos têm um alto valor intrínseco para seus usuários, portanto, sua segurança é de considerável importância (COULOURIS, 2005). A segurança de recursos de informação tem três componentes: confidencialidade (proteção contra exposição para pessoas não autorizadas), integridade (proteção contra alteração ou dano) e disponibilidade (proteção contra interferências com os meios de acesso aos recursos) (COULOURIS, 2005).

Implementar técnicas de segurança em sistemas distribuídos é mais complexo em relação aos sistemas centralizados devido a falta de um único ponto de controle e do uso de redes inseguras para a transmissão de dados (SINHA, 1996). Tanenbaum (2002) diz que a segurança em sistemas computacionais esta relacionada fortemente com a noção de confiança dos usuários, que usufruirão dos serviços.

Embora a Internet possibilite que um programa em um computador se comunique com um programa em outro computador, independentemente de sua localização, existem riscos de segurança associados ao livre acesso a todos os recursos em uma intranet. Mesmo que um *firewall* possa ser utilizado para formar uma barreira em torno de uma intranet, restringindo o tráfego de entrada e saída, isso não garante o uso apropriado dos recursos pelos usuários de dentro da intranet, nem o uso apropriado dos recursos na Internet, que não são protegidos por *firewalls*.

Em um sistema distribuído, os clientes enviam pedidos para acessar dados gerenciados por servidores, o que envolve o envio de informações em mensagens por uma rede. Nesta situação, o desafio é enviar informações sigilosas em uma ou mais mensagens por uma rede de maneira segura, que não pode ser resolvida apenas ocultando o conteúdo de mensagens, e identificar corretamente um usuário ou outro agente remoto. Esses desafios podem ser resolvidos com o uso de técnicas de criptografia desenvolvidas para esse propósito.

1.1.5 Concorrência

Tanto os serviços como os aplicativos fornecem recursos que podem ser compartilhados pelos clientes em um sistema distribuído (COULOURIS, 2005). Portanto, existe a possibilidade de que vários clientes tentem acessar um recurso compartilhado ao mesmo tempo, como por exemplo, uma estrutura de dados que registre lances de um leilão, que pode ser acessada com muita freqüência, quando o prazo final se aproximar (COULOURIS, 2005).

O processo que gerencia um recurso compartilhado poderia aceitar e tratar um pedido de cliente por vez, mas essa estratégia limita o desempenho do tratamento de pedidos (COULOURIS, 2005). Portanto, os serviços e aplicativos geralmente permitem que vários pedidos de cliente sejam processados concorrentemente (COULOURIS, 2005).

Sempre que existir compartilhamento e acesso simultâneo, deve-se assegurar que as requisições sejam processadas em sua ordem de chegada para evitar que resultados inexatos venham a ocorrer (GALLI, 2000). Entretanto, em um sistema distribuído, às vezes é impossível determinar a ordem exata em que dois eventos ocorrerem, uma vez que estes ambientes não possuem memória e nem *clock* em comum (SILBERSCHATZ, 2000).

Uma das soluções para evitar os problemas de concorrência é impedindo que dois ou mais processos acessem um mesmo recurso simultaneamente. A exclusão mútua garante que, enquanto um processo estiver acessando determinado recurso, todos os demais processos interessados no uso deste recurso deverão aguardar pelo término de sua utilização. A parte do código do programa que acessa o recurso compartilhado é denominada de região crítica. Somente tal região necessita proibir o acesso concorrente. O controle da concorrência nesta região pode ser gerenciado com a implementação e uso de semáforos e monitores (GALLI, 2000).

O semáforo é uma variável do tipo *integer* cujo seu valor indica o estado do recurso protegido, que só pode ser manipulada por duas únicas instruções: P e V (GALLI, 2000). Estas denominações são as iniciais das palavras holandesas *Proberen* e *Verhogen*, que significam testar e incrementar, respectivamente (MACHADO, 2002). O valor do semáforo igual a 1, indica que nenhum processo está utilizando o recurso, enquanto que o valor igual a 0, indica que o recurso está em uso. Sempre que um processo desejar entrar em sua região

crítica, a instrução P é executada, e caso o semáforo seja igual a 1, este valor será decrementado e o processo solicitante poderá executar as instruções de sua região crítica. Entretanto, caso uma instrução P seja executada em um semáforo com valor igual a 0, o processo ficará impedido de possuir o acesso, permanecendo em estado de espera (TANENBAM, 2006; MACHADO, 2002; GALLI 2000).

O monitor representa um conjunto de variáveis, de procedimentos e de estruturas de dados que são agrupados em um tipo especial de módulo ou de pacote. Os processos podem invocar os procedimentos de um monitor sempre que quiserem, mas eles não podem acessar diretamente as estruturas de dados internas do monitor a partir de procedimentos declarados fora dele. Em um monitor, apenas um processo pode estar ativo em qualquer momento, o que torna esta técnica útil para obter a exclusão mútua. Quando um processo chama um procedimento do monitor, as primeiras instruções do procedimento verificarão se existem processos ativos. Caso exista, o processo de chamada será suspenso até que o outro processo tenha deixado o monitor. Se nenhum outro processo estiver utilizando o monitor, o processo de chamada pode entrar (TANENBAUM, 2006; GALLI, 2000).

1.1.6 Transparência

A transparência é definida como sendo “o encobrimento do usuário e do programador da aplicação da separação dos componentes de um sistema distribuído, de modo que o sistema seja percebido como um todo ao invés de uma coleção de componentes independentes” (COULOURIS, 2005, p. 23). Em outras palavras, a transparência tem por objetivo esconder, sempre que possível, todos os detalhes do sistema que são irrelevantes do usuário (CHOW, 1998). Alcançar a transparência completa é uma tarefa complexa e requer que diversos aspectos, identificados pela *International Organization for Standardization* (ISO), sejam levados em consideração durante o desenvolvimento (COULOURIS, 2005; TANENBAUM, 2002):

- *Transparência de acesso:* Permite que recursos locais e remotos sejam acessados com o uso de operações idênticas.

- *Transparência de localização:* Permite que os recursos sejam acessados sem conhecimento de sua localização física ou na rede. A transparência de localização

permite que recursos sejam movidos entre nodos de um sistema distribuído, como por exemplo, um arquivo, sem ter seu nome e caminho afetado.

- *Transparência de concorrência:* Permite que vários processos operem concorrentemente, usando recursos compartilhados sem interferência entre eles.

- *Transparência de replicação:* Permite que várias instâncias dos recursos sejam usadas para aumentar a confiabilidade e o desempenho, sem conhecimento das réplicas por parte dos usuários ou dos programadores de aplicativos.

- *Transparência de falhas:* Permite a ocultação de falhas, possibilitando que usuários e processos concluam suas tarefas, a despeito da falha de componentes de *hardware* ou *software*.

- *Transparência de mobilidade:* Permite a movimentação de recursos e clientes dentro de um sistema, sem afetar a operação de usuários ou aplicativos.

- *Transparência de desempenho:* Permite que o sistema seja reconfigurado para melhorar o desempenho à medida que as cargas variam.

- *Transparência de escalabilidade:* Permite que o sistema e os aplicativos se expandam em escala, sem alterar a estrutura do sistema ou os algoritmos de aplicativos.

As duas transparências mais importantes são a de acesso e a de localização, pois sua presença ou ausência afeta mais fortemente a utilização de recursos distribuídos. Às vezes, elas são referidas em conjunto como transparência de rede (COULOURIS, 2005).

Como exemplo de transparência de acesso, pode-se considerar uma interface gráfica em um sistema de arquivos que organiza diretórios e subdiretórios em pastas, onde o usuário possa acessar seus arquivos locais e remotos, enquanto que um exemplo da falta de transparência de acesso seria um sistema distribuído que não permitisse o acesso aos arquivos em um computador remoto a menos que o cliente faça uso, por exemplo, de um aplicativo de FTP (COULOURIS, 2005).

A transparência de localização pode ser exemplificada através dos nomes de recursos da web, isto é, os URLs, pois a parte do URL que identifica o nome de um servidor web se refere a um nome de computador em um domínio, em vez de seu endereço IP (COULOURIS,

2005). Entretanto, os URLs não são transparentes à mobilidade, pois se a página web mudar para um domínio diferente, todas as referencias (*links*) existentes nas outras páginas ainda apontarão para a página original (COULOURIS, 2005).

O endereço de correio eletrônico pode ser ilustrado como transparência de rede. Ele consiste em um nome de usuário e um nome de domínio. O envio de correspondências para tal usuário não envolve o conhecimento de sua localização física ou na rede, nem o procedimento de envio de uma mensagem de correio depende da localização do destinatário. Assim, o correio eletrônico, dentro da Internet, oferece tanto transparência de localização como de acesso (COULOURIS, 2005).

A transparência de falhas também pode ser vista no contexto de correio eletrônico, que é entregue ao destinatário mesmo quando os servidores ou os enlaces de comunicação falham. As falhas são mascaradas pela tentativa de retransmitir as mensagens até que sejam enviadas com êxito, mesmo que esse processo demore. Geralmente, o *middleware* converte as falhas de redes e processos em exceções em nível de programação (COULOURIS, 2005).

A telefonia móvel pode ser ilustrada como exemplo da transparência de mobilidade, pois em um cenário onde quem realiza a chamada e quem é chamado estejam viajando de trem em diferentes partes de um país, nenhum dos dois possui conhecimento da mobilidade dos telefones (COULOURIS, 2005).

O uso dos diferentes tipos de transparência oculta e transforma em anônimos os recursos que não têm relevância direta para a execução de uma tarefa por parte de usuários e de programadores de aplicativos (COULOURIS, 2005).

1.1.7 Escalabilidade

Os sistemas distribuídos funcionam de forma efetiva e eficaz em muitas escalas diferentes, variando desde uma pequena intranet até a Internet (COULOURIS, 2005). Um sistema é descrito como escalável se permanece eficiente quando há um aumento significativo no número de recursos e no número de usuários (GALLI, 2000; COULOURIS, 2005). A Internet é um exemplo de um sistema distribuído no qual o número de computadores e serviços vem aumentando substancialmente (COULOURIS, 2005).

O projeto de um sistema distribuído escalável apresenta os seguintes desafios (COULOURIS, 2005; GALLI, 2000):

- *Controlar o custo dos recursos físicos:* À medida que a demanda por um recurso aumenta, deve ser possível, a um custo razoável, ampliar o sistema para atendê-la. Por exemplo, a freqüência com que os arquivos são acessados em uma intranet provavelmente vai aumentar a medida com que o número de usuários e de computadores for aumentando. Deve ser possível adicionar servidores de arquivos de forma a evitar o gargalo de desempenho que haveria caso um único servidor tivesse que tratar todos os pedidos.

- *Controlar a perda de desempenho:* Considerando o gerenciamento de um conjunto de dados, cujo tamanho é proporcional ao número de usuários ou recursos presentes no sistema, como por exemplo, a tabela de correspondência entre os nomes de domínios dos computadores e seus endereços IP mantidos no *Domain Name System* (DNS), seu crescente aumento, mesmo possuindo uma estrutura hierárquica, pode resultar em alguma perda de desempenho.

- *Impedir que os recursos de software se esgotem:* Um exemplo prático da falta de escalabilidade é o esgotamento de endereços IPs na Internet. Por isso, uma nova versão do protocolo, com endereços IP em 128 bits, está sendo adotada e exigirá modificações em muitos componentes de *software*.

- *Evitar gargalos de desempenho:* Em geral, os algoritmos devem ser descentralizados para evitar a existência de gargalos de desempenho. Esse item pode ser ilustrado referenciando o predecessor do DNS, no qual a tabela de correspondência entre endereços IP e nomes era mantida em um único arquivo central, cujo *download* podia ser feito em qualquer computador que precisasse dele. Esse modelo funcionava perfeitamente quando havia apenas algumas centenas de computadores na Internet. O DNS eliminou esse gargalo particionando a tabela de correspondência de nomes entre diversos servidores localizados em toda a Internet e administrados de forma local.

De preferência, o software de sistema e de aplicativo não deve mudar quando a escala do sistema aumentar, mas isso é difícil de conseguir (COULOURIS, 2005). O problema da escala é um tema central no desenvolvimento de sistemas distribuídos

(COULOURIS, 2005). Sinha (1996) menciona alguns dos princípios utilizados para projetar sistemas escaláveis:

- *Evitar sistemas centralizados:* O uso de sistemas centralizados, como um único servidor de arquivo ou um único banco de dados, não é considerado um sistema escalável, podendo apresentar perda de desempenho e tornar-se um gargalo quando houver aumentos demasiados de requisições para ele.

- *Evitar algoritmos centralizados:* Um algoritmo centralizado é aquele que opera em um único servidor, coletando informações dos diversos nodos da rede e redistribuindo-as a eles. O uso de tais algoritmos no projeto de sistemas distribuídos também não é considerado como escalável, pelo mesmo motivo dos sistemas centralizados.

- *Execute a grande parte das operações em estações clientes:* Quando possível, determinadas operações deveriam ser executadas no cliente ao invés do servidor, pois ele é um recurso comum para os demais membros da rede e conseqüentemente, os ciclos de um servidor são mais custosos em relação aos ciclos de uma estação de trabalho. Este princípio realça a escalabilidade do sistema e reduz a disputa por recursos compartilhados.

2 SISTEMAS DE ARQUIVOS DISTRIBUÍDOS

Os sistemas de arquivos foram originalmente desenvolvidos para sistemas computacionais centralizados e computadores *desktop* como um recurso do sistema operacional que fornece uma interface de programação conveniente para o armazenamento de arquivos em disco (COULOURIS, 2005). Subseqüentemente, eles adquiriram características como controle de acesso e mecanismos de proteção de arquivos, que os tornaram úteis para o compartilhamento de dados e programas. Os sistemas de arquivos distribuídos suportam o compartilhamento de informações através de arquivos e recursos de *hardware* com armazenamento persistente em toda uma intranet. Um serviço de arquivo bem projetado dá acesso a arquivos armazenados em um servidor com desempenho e confiabilidade semelhantes (e, em alguns casos, melhores) aos arquivos armazenados em discos locais. Seu projeto é adaptado para as características de desempenho e confiabilidade das redes locais e, portanto, eles são mais eficazes no fornecimento de armazenamento persistente compartilhado para uso em intranets (COULOURIS, 2005).

Um serviço de arquivos permite que os programas armazenem e acessem arquivos remotos exatamente como se fossem locais, possibilitando que os usuários acessem seus arquivos a partir de qualquer computador em uma rede (COULOURIS, 2005). A concentração do armazenamento persistente em alguns poucos servidores reduz a necessidade de armazenamento em disco local e (o mais importante) permite que sejam feitas economias no gerenciamento e no arquivamento (*backup*) dos dados persistentes pertencentes a uma organização (COULOURIS, 2005).

Um sistema de arquivos distribuído é uma implementação de um sistema de arquivos que consiste em locais de armazenamento dispersos fisicamente em uma rede, entretanto, ele provê uma visão de sistema de arquivos tradicional para o usuário (CHOW, 1998). Esses sistemas possuem duas características principais, que devem ser tratadas de forma

transparente para os usuários, que é a dispersão e a multiplicidade de usuários e arquivos, ou seja, os múltiplos clientes acessam em diversos locais os arquivos residentes em servidores espalhados pela rede (CHOW, 1998).

A principal característica de um sistema de arquivos distribuído é o fato de ele gerenciar um conjunto de dispositivos de armazenamento dispersos. Seu espaço de armazenamento global é composto por diferentes espaços de armazenamento menores localizado remotamente. Existem configurações de serviços de sistemas de arquivos distribuídos nas quais os servidores são executados em máquinas dedicadas e configurações na qual um computador pode desempenhar o papel de um servidor e de um cliente (SILBERSCHATZ, 2000).

2.1 Conceitos Básicos

No decorrer desta seção, serão apresentados conceitos amplos relacionados a sistemas de arquivos distribuídos que servirão de base para a análise das seções subseqüentes.

2.1.1 Sistemas de Arquivos

Os sistemas de arquivos são responsáveis pela organização, armazenamento, recuperação, atribuição de nomes, compartilhamento e proteção de arquivos (COULOURIS, 2005). Eles fornecem uma interface de programação que caracteriza a abstração de arquivo, liberando os usuários da preocupação com os detalhes da alocação e do armazenamento físico no disco (GALLI, 2000; TANENBAUM, 2000; COULOURIS, 2005).

Os arquivos contêm dados e atributos (COULOURIS, 2005). Os dados consistem em uma seqüência de elementos (normalmente, *bytes* de 8 *bits*), acessíveis pelas operações de leitura e escrita de qualquer parte dessa seqüência (COULOURIS, 2005; TANENBAUM, 2000; SILBERSCHATZ, 2000). Os atributos são mantidos como um único registro, contendo informações como o tamanho do arquivo, indicações de tempo, tipo de arquivo, identidade do proprietário e listas de controle de acesso (COULOURIS, 2005; TANENBAUM, 2000; SILBERSCHATZ, 2000). Uma estrutura de registro de atributo típica esta ilustrada na Figura 2.1.

Os sistemas de arquivos são projetados para armazenar e gerenciar um grande número de arquivos, com recursos para criação, atribuição de nomes e exclusão de arquivos (TANENBAUM, 2000). Os sistemas de arquivos também assumem a responsabilidade pelo controle de acesso aos arquivos, restringindo o acesso de acordo com as autorizações dos usuários e com o tipo de acesso solicitado (leitura, atualização, execução etc.) (COULOURIS, 2005).

Tamanho do arquivo
Horário de criação
Horário de acesso (leitura)
Horário de modificação (escrita)
Horário de alteração de atributo
Contagem de referência
Proprietário
Tipo de arquivo
Lista de controle de acesso

Figura 2.1 Estrutura básica de um registro de atributo de arquivo
Fonte: Coulouris, 2005, p. 327

Um sistema de arquivos fornece serviços de acesso a arquivos para os clientes (SILBERSCHATZ, 2000). Uma interface cliente para este serviço é formada por um conjunto de operações de arquivos primitivas, como criar um arquivo, excluir um arquivo, ler a partir de um arquivo e gravar em um arquivo (SILBERSCHATZ, 2000). O principal componente de *hardware* que um servidor de arquivos controla é um conjunto de dispositivos de armazenamento secundário local (geralmente discos magnéticos), nos quais os arquivos são armazenados e a partir dos quais eles são recuperados de acordo com os pedidos dos clientes (SILBERSCHATZ, 2000).

2.1.2 Nomeação e Transparência

A nomeação é um mapeamento entre objetos lógicos e físicos. Por exemplo, os usuários trabalham com objetos lógicos de dados representados por nomes de arquivos, enquanto que o sistema manipula blocos físicos de dados, armazenados em trilhas de discos,

fornecendo ao usuário uma abstração de um arquivo que oculta os detalhes de como e onde ele está armazenado de fato no disco físico (SILBERSCHATZ, 2000). Dada a localização lógica de um arquivo, ou seja, o caminho até ele, é necessário analisar os componentes deste caminho a fim de encontrar sua localização física. É necessário descobrir em quais blocos de quais discos e de quais servidores se encontra o arquivo em questão (KON, 1994).

Quando os arquivos de um sistema estão distribuídos entre vários servidores localizados em diferentes máquinas, é desejável que a localização destes dados seja transparente aos usuários do sistema (FLOYD, apud KON, 1994). Em um sistema transparente, uma nova dimensão é adicionada a abstração, a de ocultar o local na rede onde o arquivo esta armazenado:

> "Idealmente, um serviço de sistema de arquivo distribuído deve aparecer aos seus clientes como um sistema de arquivos centralizado-convencional. A multiplicidade e a dispersão de seus servidores e dispositivos de armazenamento devem ser transparentes. Ou seja, a interface cliente desse serviço não deve fazer distinção entre arquivos locais e remotos. Cabe ao serviço localizar os arquivos e organizá-lo para o transporte dos dados. Um sistema de arquivos distribuído transparente facilita a mobilidade do usuário trazendo o ambiente do usuário (ou seja, seu diretório inicial) para onde quer que o usuário efetue login" (SILBERSCHATZ, 2000, p. 542).

Diversas técnicas de transparência, adotadas em sistemas distribuídos e visto na seção 1.1.6, são empregadas parcialmente ou diretamente na implementação de serviços de arquivos (COULOURIS, 2005).

2.1.3 Cache

Para aumentar o desempenho no acesso a informação fornecida por um sistema, procura-se armazenar os dados com maior número de solicitação em memória, evitando desta forma uma possível sobrecarga para obter a informação do sistema novamente (SINGHAL, 1994; CARVALHO, 2003). Esta técnica auxilia na economia do tempo de processamento, pois para acessar informações remotas, por exemplo, o sistema estará limitado a velocidade da rede, que, mesmo rápida, estará limitado a velocidade do meio físico do servidor remoto, pois este ainda necessitará procurar e processar os dados solicitados, carregando-os na memória e enviando-os para o cliente (SINGHAL, 1994; CARVALHO, 2003).

Mesmo no acesso a informações locais, a velocidade de acesso à memória é superior a velocidade de acesso ao meio de armazenamento, como um disco rígido, por exemplo, que

necessitaria mover o braço do leitor até a trilha em que se encontram os dados e esperar até que a rotação do disco traga-os à cabeça de leitura (CARVALHO, 2003).

Em sistemas de arquivos distribuídos, o *cache* pode estar presente tanto no cliente, como no servidor, evitando assim que o usuário faça uso desnecessário da rede para obter informações antes já solicitadas, enquanto que o servidor diminui o acesso ao meio físico de armazenamento dos dados para enviá-los ao cliente (CARVALHO, 2003).

2.2 Serviços Oferecidos

Com a finalidade de proporcionar um ambiente de fácil uso para os usuários, escondendo toda a complexidade existente na nomeação global, no acesso aos arquivos dispersos pela rede e na organização destes, os sistemas de arquivos distribuídos fazem uso de três serviços que têm por objetivo controlar e atender estas funcionalidades, que são o serviço de nomes, o serviço de arquivos e o serviço de diretórios, respectivamente (GALLI, 2000).

2.2.1 Serviço de Nomes

O serviço de nomes tem por objetivo indicar a localização de um determinado arquivo no conjunto de computadores do sistema de arquivos distribuído. Caso o nome do arquivo contiver o nome do computador aonde ele esta localizado, como por exemplo “apolo:/tmp/arquivo”, então este serviço não provê transparência de localização. Para prover esta transparência, o nome de um arquivo não deve possuir indícios de sua localização física, pois caso ele mude de lugar ou possua várias cópias, o seu nome ou caminho não precisarão ser alterados (GALLI, 2000).

2.2.2 Serviço de Arquivos

O serviço de arquivos é responsável por fornecer os mesmos serviços e recursos de um sistema de arquivos tradicional, com a diferença que um sistema de arquivos distribuído pode ser acessado de qualquer estação de trabalho de dentro da rede. Esse serviço também cuida das propriedades dos arquivos, como data de criação, data de alteração, tamanho, proprietário, permissões de acesso e outras informações relevantes, além de manter a integridade das operações realizadas nos arquivos (GALLI, 2000).

2.2.3 Serviço de Diretórios

O objetivo do serviço de diretórios é manter a organização dos arquivos armazenados no sistema, fornecendo ao usuário uma interface para que eles possam organizar seus arquivos em uma estrutura hierárquica, através de diretórios e subdiretórios (GALLI, 2000).

2.3 Características Desejadas

Muitos dos conceitos encontrados na implementação de sistemas distribuídos devem ser levados em consideração no desenvolvimento de sistemas de arquivos distribuídos. Inicialmente, os primeiros sistemas de arquivos distribuídos ofereciam recursos de transparência de acesso e transparência de localização, emergindo subseqüentemente à preocupação no desenvolvimento de recursos como desempenho, escalabilidade, controle de concorrência, tolerância a falhas e segurança (COULOURIS, 2005).

2.3.1 Atualização Concorrente de Arquivos

As alterações feitas em um arquivo por um único cliente não devem interferir na operação de outros clientes que estejam acessando, ou alterando, o mesmo arquivo simultaneamente (COULOURIS, 2005). Em muitos aplicativos, a necessidade de controle de concorrência para acesso a dados compartilhados é amplamente aceita, e as técnicas de implementação muitas vezes são dispendiosas (COULOURIS, 2005).

O maior problema encontrado nas implementações desse tipo de solução é quanto a sincronização dos arquivos, o que inclui leitura e escrita concorrente (CARVALHO, 2003). A leitura concorrente pode ser implementada facilmente caso não haja escrita concorrente, pois quando um arquivo estiver sendo lido, certamente ninguém poderá alterá-lo (CARVALHO, 2003). Para implementar a escrita concorrente, deve-se levar em conta que, quando um cliente escreve em um arquivo, todos os leitores devem ser avisados que o documento foi alterado, tendo-se que tomar cuidado também para que estas alterações não desfaçam as alterações realizadas por outros usuários (COULOURIS, 2005; CARVALHO, 2003).

2.3.2 Replicação de Arquivos

Em um serviço de arquivos que suporta replicação, um arquivo pode ser representado por várias cópias de seu conteúdo em diferentes locais, trazendo como benefício a possibilidade dos servidores, que hospedam um mesmo conjunto de arquivos, compartilharem a carga do fornecimento de um serviço para clientes que acessam esses arquivos melhorando assim a escalabilidade do serviço, e melhora a tolerância a falhas, permitindo que, em caso de falhas, os clientes localizem outro servidor que contenha uma cópia do arquivo (COULOURIS, 2005).

A exigência básica de um esquema de replicação é que diferentes réplicas do mesmo arquivo residam em servidores independentes, ou seja, a disponibilidade de uma réplica não é afetada pela disponibilidade das demais (SILBERSCHATZ, 2000). Kon (1994) menciona algumas importantes vantagens que podem existir com a replicação de arquivos:

Caso um disco seja danificado, as informações nele contidas não serão perdidas, podendo ser recuperadas de outros discos em outros servidores;

Se um servidor está momentaneamente inoperante ou inacessível, os seus arquivos podem ser acessados em servidores alternativos, pois haverá uma maior disponibilidade do serviço de arquivos;

Arquivos ou diretórios muito lidos podem ser oferecidos por vários servidores, distribuindo-se a demanda eqüitativamente entre os diversos servidores e aumentando o desempenho global do sistema.

O principal desafio associado a replicação é a sua atualização e manutenção da consistência entre as réplicas dos arquivos nos diversos servidores (SILBERSCHATZ, 2000). Para o usuário, as réplicas de um arquivo denotam a mesma entidade lógica e, dessa forma, uma atualização a qualquer réplica deve refletir nas demais (KON, 1994). Os detalhes da replicação devem ser transparentes para os usuários, sendo tarefa do esquema de nomeação mapear um nome de arquivo replicado em um computador específico sem o conhecimento de sua localização pelo cliente (SILBERSCHATZ, 2000; KON, 1994).

Poucos serviços de arquivos suportam replicação completa, mas a maioria suporta o armazenamento de arquivos, ou de porções de arquivos em *caches* locais, que é uma forma limitada de replicação (COULOURIS, 2005).

2.3.3 Heterogeneidade do Hardware e do Sistema Operacional

As interfaces de serviço devem ser definidas de modo que o software cliente e servidor possa ser implementado para diferentes sistemas operacionais e computadores (COULOURIS, 2005). Uma das grandes deficiências dos sistemas computacionais tem sido a incompatibilidade tanto de *hardware* quanto de *software* fabricado por diferentes companhias (KON, 1994). Cada vez mais, buscam-se padronizações que permitam que máquinas de diferentes fabricantes se comuniquem sem grandes dificuldades (KON, 1994). O sistema de arquivos distribuído deve ser implementado de forma que o servidor e o cliente não necessitem da mesma arquitetura de *hardware* e da mesma solução de *software* para a correta execução do serviço (COULOURIS, 2005).

2.3.4 Tolerância a Falhas

Por ser parte essencial nos sistemas distribuídos, é essencial que o serviço de arquivos distribuídos continue a funcionar diante de falhas de clientes e servidores (COULOURIS, 2005). Falhas de *hardware* ou de *software* podem ocasionar uma interrupção momentânea no funcionamento de uma máquina (KON, 1994). A queda de um nodo, ou uma falha em um canal de comunicação pode levar a uma partição na rede, ou seja, a conversação entre dois pontos da rede é interrompida durante certo intervalo de tempo (KON, 1994). A grande maioria dos sistemas de arquivos distribuídos é sensível a partições na rede quando estão em operação, e caso o cliente não consiga estabelecer contato com alguns dos servidores, os processos que fizeram solicitações aos serviços de arquivos serão notificados de que as informações solicitadas não estão disponíveis ou simplesmente são bloqueados até que a conexão se estabeleça (KON, 1994).

O método mais utilizado para aumentar a disponibilidade de um serviço de arquivos tem sido a replicação de dados, onde os arquivos são armazenados em dois ou mais servidores e caso um deles não esteja disponível, outro servidor poderá fornecer os serviços solicitados (KON, 1994).

2.3.5 Consistência

Um sistema de arquivos distribuídos deve garantir a consistência dos arquivos de seus usuários (SILBERSCHATZ, 2000). Quando um arquivo é replicado ou recuperado do

cache de uma estação cliente, por exemplo, podem ocorrer demoras inevitáveis na propagação das modificações devido a latências de rede, podendo resultar na geração de inconsistências ou até perda das informações caso o sistema seja frágil a esse tipo de cenário (COULOURIS, 2005; SILBERSCHATZ, 2000).

2.3.6 Segurança

Compartilhar arquivos entre vários ambientes e usuários é uma das vantagens que os sistemas de arquivos distribuídos oferecem (CARVALHO, 2003). Entretanto, deixar que outros usuários possuam acesso a arquivos confidenciais é um grande problema, sendo necessário adotar mecanismos que garantam que pessoas não autorizadas não tenham acesso a tais informações (COULOURIS, 2005; CARVALHO, 2003).

Praticamente todos os sistemas de arquivos fornecem mecanismos de controle de acesso baseados no uso de listas de controle de acesso (COULOURIS, 2005). Nos sistemas de arquivos distribuídos, há a necessidade de autenticar as requisições dos clientes para que o controle de acesso no servidor seja baseado nas identidades corretas de usuário e para proteger o conteúdo das mensagens de requisição e resposta com assinaturas digitais e, opcionalmente, criptografia de dados secretos (COULOURIS, 2005).

2.3.7 Eficiência

As técnicas usadas para a implementação de serviços de arquivos representam uma parte importante do projeto de sistemas distribuídos (COULOURIS, 2005). Um sistema de arquivos distribuído deve fornecer um serviço que seja comparável (ou melhor) aos sistemas de arquivos locais, em termos de desempenho e confiabilidade (COULOURIS, 2005). Ele deve ser conveniente para administrar, com operações e ferramentas que permitam aos administradores de sistema instalá-lo e operá-lo adequadamente (COULOURIS, 2005).

3 FERRAMENTAS DE SISTEMAS DE ARQUIVOS DISTRIBUÍDOS

No decorrer deste capítulo, serão apresentadas algumas das principais ferramentas de sistemas de arquivos distribuídos.

3.1 Network File System

O Network File System (NFS) foi originalmente desenvolvido pela Sun para ser utilizado em ambientes Unix, mas atualmente é possível encontrar implementações para quase todas as arquiteturas de computadores e sistemas operacionais (TANENBAUM, 2002; COULOURIS, 2005). A idéia principal do NFS é possibilitar que um conjunto de clientes e servidores possam compartilhar um sistema de arquivos em comum, permitindo que um nodo seja ao mesmo tempo um cliente acessando arquivos remotos ou um servidor exportando arquivos (TANENBAUM, 2002).

O NFS é uma coleção de protocolos que provê ao cliente a visão de um sistema de arquivos distribuído, similar ao funcionamento do CORBA (TANENBAUM, 2002). Nesta seção, serão apresentados diversos aspectos do NFS, concentrando-se nas versões 3 (especificado na RFC 1813) e 4 (especificado na RFC 3530).

3.1.1 Arquitetura

Nos sistemas operacionais, as chamadas de sistema são utilizadas para fazer a interface de comunicação entre o cliente e o sistema de arquivos local (TANENBAUM, 2002). Nos sistemas operacionais baseados em Unix, esta interface é substituída por outra interface denominada de *Virtual File System* (VFS), que realiza a comunicação e oculta as diferenças entre os diversos sistemas de arquivos (TANENBAUM, 2002). As operações

encaminhadas à interface VFS são direcionadas ao sistema de arquivos local ou algum componente previamente instalado, como o cliente do NFS, que é responsável pelo acesso aos arquivos armazenados nos servidores através das chamadas de procedimentos remotos (TANENBAUM, 2002; COULOURIS, 2005). A Figura 3.1 ilustra a arquitetura do NFS.

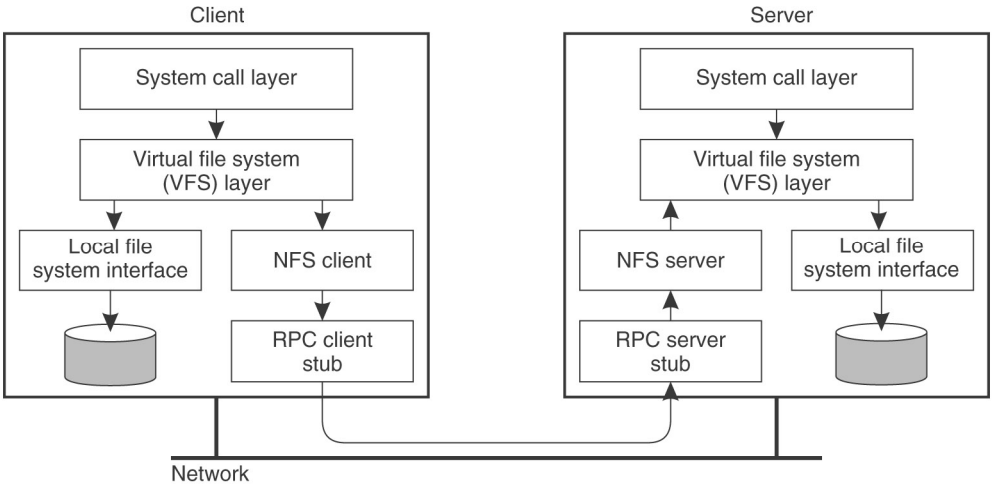

Figura 3.1 Arquitetura do Network File System em ambientes Unix
Fonte: Tanenbaum, 2002, p. 578

Quando o cliente efetua uma requisição ao serviço de arquivos, esta solicitação é encaminhada para a interface do VFS, que de acordo com a localização física do arquivo solicitado, remete a requisição para o sistema de arquivos local ou para um componente (KON, 1996).

O VFS mantém uma tabela para cada sistema de arquivos montado, com uma entrada para cada um dos arquivos abertos, denominada de *v-node*, que é utilizado para informar se o arquivo é local ou remoto (KON, 1996). Caso o arquivo esteja armazenado localmente, o *v-node* possuirá uma referência para o *i-node* do arquivo local e caso o arquivo esteja armazenado remotamente, ele conterá o manipulador do arquivo que fornece todas as informações necessárias para acessá-lo (KON, 1996).

Uma importante vantagem na arquitetura do NFS é quanto a independência do sistema de arquivos local, possibilitando que sejam compartilhados arquivos em ambientes heterogêneos (TANENBAUM, 2002).

3.1.2 Comunicação

Uma das características importantes na arquitetura do NFS é sua independência quanto a sistema operacional, rede e protocolo de comunicação, possibilitando que, por exemplo, clientes Windows acessem servidores Unix (TANENBAUM, 2002). Essa independência é atingida devido ao fato do NFS possuir em sua arquitetura uma camada de RPC que oculta as diferenças citadas, sendo esta responsável pela comunicação entre clientes e servidores (TANENBAUM, 2002).

Uma RPC é similar a uma invocação a método remoto, pois um processo cliente chama um procedimento que está sendo executado em um processo servidor, sendo que os servidores podem ser clientes de outros servidores para permitir encadeamentos de RPCs (COULOURIS, 2005). Um processo servidor define em sua interface de serviço os procedimentos que estão disponíveis para serem chamados de forma remota (COULOURIS, 2005).

Todas as operação da versão 4 do NFS são implementadas com uma única chamada composta de procedimento remoto a um servidor de arquivos, ou seja, uma mesma chamada RPC para a leitura de um arquivo, por exemplo, pode conter uma operação complexa envolvendo bloqueio, abertura, leitura etc., a exemplo da Figura 3.2 (b), diferente da versão 3, onde este processo era resolvido com duas chamadas de procedimento remoto, ilustrado na Figura 3.2 (a) (TANENBAUM, 2002).

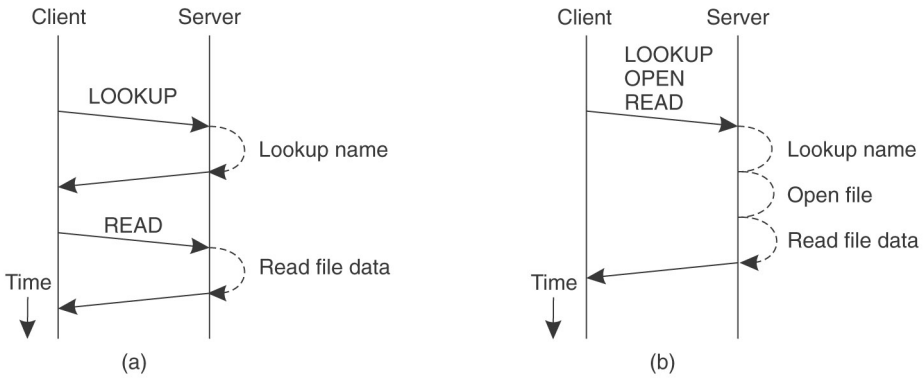

Figura 3.2 Exemplo do processo de leitura de um arquivo no NFSv3 (a) e no NFSv4 (b)
Fonte: Tanenbaum, 2002, p. 582

3.1.3 Processos

O NFS é um sistema cliente-servidor no qual usuários solicitam ao servidor que realize operações em determinado arquivo (TANENBAUM, 2002). Uma de suas grandes características, que foi abandonado na versão 4, é o fato do servidor não guardar o estado das transações realizados. Esta característica traz algumas vantagens imediatas, sendo que a mais significativa é que, quando ocorre uma falha no servidor, ele não perde nenhuma informação sobre o estado da comunicação com os clientes, não precisando, desta forma, gastar tempo na tentativa de reconstituir este estado quando ele se recuperar da falha. Para o cliente, tudo que ele necessita fazer quando a comunicação com o servidor falhar é repetir as chamadas ao serviço até que obtenha resposta. Entretanto, esta característica impede o servidor de gerenciar o acesso concorrente de seus arquivos e desta forma, o NFS não assume a consistência de seu sistema de arquivos, podendo haver casos em que, diferentes clientes possuam diferentes cópias do mesmo arquivo ou diretório no *cache* de seus computadores (KON, 1996; TANENBAUM, 2002).

Em sua versão 4, o NFS guarda o estado das transações realizadas, mantendo uma tabela de todos os arquivos abertos pelos usuários, assemelhando-se a um sistema de arquivos centralizado (TANENBAUM, 2002).

3.1.4 Serviço de Nomes

Assim como em todos os sistemas de arquivos distribuídos, o serviço de nomes é uma característica importante do NFS (TANENBAUM, 2002). A idéia principal do modelo de serviço de nomes do NFS é prover aos usuários um acesso transparente ao sistema de arquivos dos servidores, que é alcançada quando o cliente monta em seu sistema de arquivos local o sistema de arquivos distribuído, conforme apresentado na Figura 3.3. Esta ilustração demonstra que os usuários não compartilham o mesmo espaço de nomes, pois em ambos os clientes, o mesmo diretório do servidor esta compartilhado em pontos de montagem diferentes. A desvantagem deste modelo reflete no processo de comunicação entre os clientes, que terão dificuldades em referenciar um mesmo arquivo, uma vez que o caminho de acesso a ele será diferente entre os usuários (TANENBAUM, 2002).

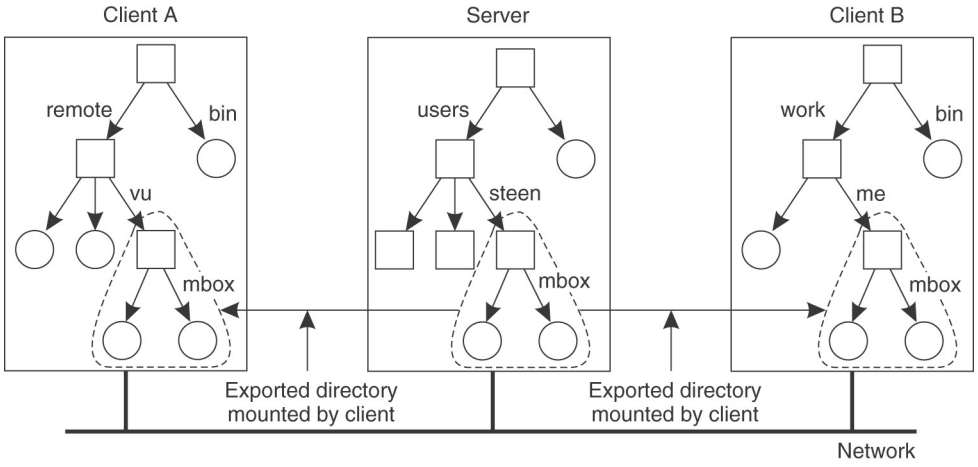

Figura 3.3 Montagem de um sistema de arquivos distribuído no NFS
Fonte: Tanenbaum, 2002, p. 583

3.1.5 Cache e Replicação

O módulo cliente do NFS armazena em *cache* os resultados obtidos em determinadas operações para reduzir o número de requisições feitas aos servidores (TANENBAUM, 2002). O uso do *cache* no cliente implica na possibilidade de existirem diversas versões de arquivos, ou porções deles, em diferentes nós clientes, pois as gravações feitas por um cliente não resultam na atualização imediata do arquivo em outros computadores. Em vez disso, os clientes são responsáveis por fazerem uma consulta seqüencial no servidor para verificar se os dados armazenados em *cache* que eles contêm são atuais. A Figura 3.4 ilustra a integração do *cache* do computador local com o módulo cliente do NFS. Cada cliente pode possuir *cache* em memória ou no disco rígido que contém informações recentemente acessadas do servidor, tais como o conteúdo de um arquivo, seus atributos e diretórios (TANENBAUM, 2002).

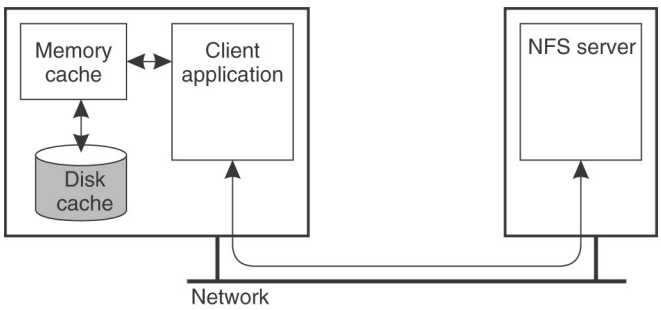

Figura 3.4 Representação do cache ao lado do cliente no Network File System
Fonte: Tanenbaum, 2002, p. 595

A versão 4 do NFS disponibiliza um suporte ainda limitado a replicação de arquivos, permitindo que os dados sejam copiados entre diversos servidores, sendo que estas cópias estarão disponíveis somente para consulta dos usuários (TANENBAUM, 2002). Quando o cliente estabelecer acesso ao sistema de arquivos remoto do servidor, ele obterá a relação dos locais que disponibilizam uma réplica do ponto de montagem e caso ocorram falhas de comunicação, o cliente tentará estabelecer uma nova conexão com os servidores informados na lista (TANENBAUM, 2002).

3.1.6 Tolerância a Falhas

Até a versão 4 do NFS, a tolerância a falhas não era um assunto muito mencionado, uma vez que seu protocolo não exigia que o servidor guardasse informações sobre o estado da comunicação com os seus clientes para oferecer o seu serviço satisfatoriamente, entretanto, com a mudança de arquitetura entre a versão 3 e 4 do NFS, a tolerância e a recuperação de falhas passam a ser preocupações desta ferramenta (TANENBAUM, 2002).

Um dos problemas existentes com o RPC diz respeito ao fato dele não proporcionar nenhuma garantia em relação a sua confiabilidade, sendo que o principal problema do NFS é a falta de uma implementação para detecção de solicitações duplicadas (TANENBAUM, 2002). Entretanto, essas situações são amenizados com o uso de um *cache* implementado no servidor que gerencia as requisições (que possuem em seu cabeçalho um código identificador único) vindas do cliente (TANENBAUM, 2002).

A Figura 3.5 ilustra esta implementação com a exemplificação da negociação de mensagens entre o cliente e o servidor. Na primeira situação, apresentada na Figura 3.5 (a), o cliente envia uma requisição e inicia um *timer* que, se o cronômetro encerrar antes da resposta chegar, o cliente retransmite a mensagem original com seu código identificador original, que será ignorada pelo servidor, que ainda não havia concluído o processamento da primeira requisição (TANENBAUM, 2002).

No segundo caso, apresentado na Figura 3.5 (b), o servidor recebe uma requisição retransmitida logo após ter enviado o processamento da solicitação original ao cliente, sendo esta nova requisição ignorada, uma vez que seus códigos identificadores são idênticos e a resposta à mensagem original foi encaminhada (TANENBAUM, 2002).

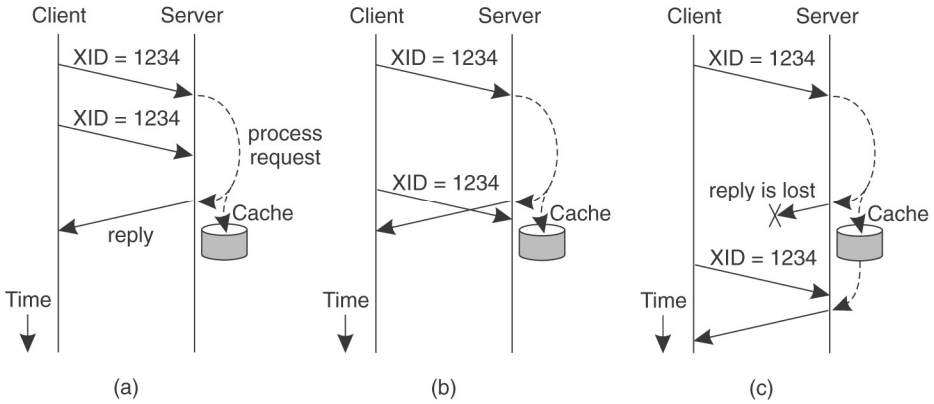

Figura 3.5 Exemplo de situações para controle de mensagens retransmitidas
Fonte: Tanenbaum, 2002, p. 598

Na terceira situação, apresentada na Figura 3.5 (c), a resposta enviada ao cliente se perdeu, sendo então retransmitido o conteúdo existente em *cache*, evitando que o servidor tivesse que processar o pedido novamente (TANENBAUM, 2002).

3.2 Coda File System

O Coda File System começou a ser desenvolvido em 1987 pela Universidade de Carnegie Mellon, nos EUA, sendo descendente da versão 2 do Andrew File System (AFS) (TANENBAUM, 2002). Seu principal objetivo é fornecer suporte a realização de operações desconectadas ao sistema de arquivos para computadores portáteis que costumam ficar grande parte do tempo fora da rede, provendo assim uma alta disponibilidade dos arquivos aos seus usuários. Para tanto, uma cópia do arquivo que o usuário esta manipulando é armazenada na máquina local, evitando assim que falhas de comunicação entre o cliente e o servidor prejudiquem o usuário (TANENBAUM, 2002).

O suporte a operações desconectadas surgiu com a proposta de criar um sistema de arquivos tolerante a falhas de rede entre clientes e servidores (KISTLER, 1991). O esforço empregado no Coda File System para lidar com este problema mostrou-se conveniente com o advento de clientes móveis (KISTLER, 1991). Quando um usuário atualiza um arquivo, as alterações precisam ser propagadas aos servidores que o armazenam e caso o cliente esteja conectado ao servidor, esta atualização é feita de forma síncrona, ou seja, a alteração é propagada no momento de sua gravação no cliente (BRAAM, 1998). Caso haja problemas nesta comunicação, ao invés de reportar o erro ao usuário, as informações são gravadas localmente em um registro de alterações pendentes e assim que a comunicação com os

servidores for restabelecida, estas alterações serão propagadas automaticamente (BRAAM, 1998).

3.2.1 Arquitetura

Por ser descendente do Andrew File System, o Coda File System herdou muitas de suas características arquitetônicas (TANENBAUM, 2002). O AFS foi projetado para disponibilizar a cada aluno e professor da Universidade de Carnegie Mellon uma estação de trabalho com um sistema compatível com o Unix, onde os usuários entrariam em qualquer computador da rede e a sua visão do sistema de arquivos deveria ser a mesma. Sendo assim, era essencial tomar o máximo de cuidado quanto a escalabilidade da ferramenta, uma vez que ela deveria atender a aproximadamente 10 mil estações de trabalho (TANENBAUM, 2002, KON, 1994).

Para satisfazer essa exigência, o AFS é implementado em dois grupos de computadores, conforme ilustrado na Figura 3.6 (TANENBAUM, 2002). Um grupo consiste em um número relativamente pequenos de servidores de arquivos dedicados que são administrados de forma centralizada, denominados de *Vice*. O segundo grupo consiste em uma coleção maior de estações de trabalho, denominadas de *Virtue*, que concedem aos usuários acesso ao sistema de arquivos distribuído (TANENBAUM, 2002).

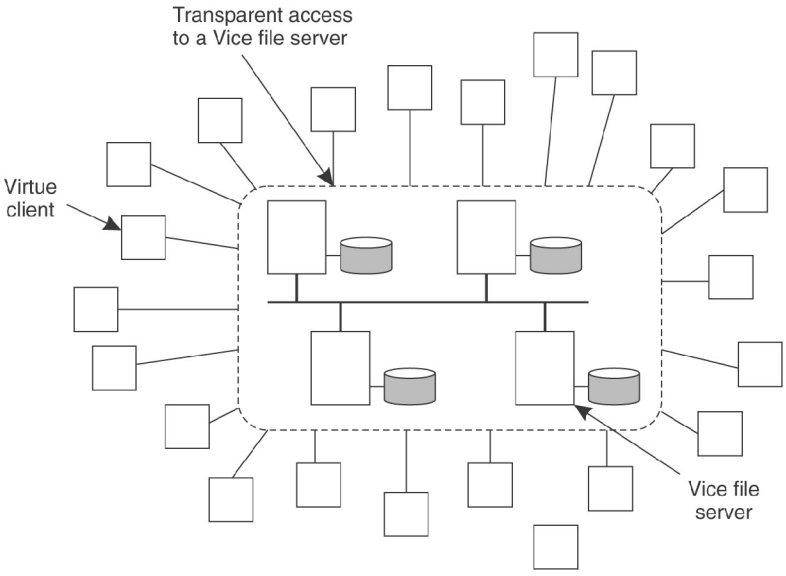

Figura 3.6 Arquitetura do Andrew File System
Fonte: Tanenbaum, 2002, p. 605

O Coda File System segue esta mesma organização. Todas as estações de trabalho possuem um processo denominado de *Venus* que é executado em nível de usuário no sistema operacional e tem como objetivo fornecer ao cliente o acesso ao sistema de arquivos distribuído (TANENBAUM, 2002). O processo *Venus* também é responsável por permitir que o cliente continue manipulando seus arquivos mesmo que a estação de trabalho não esteja conseguindo estabelecer conexão com o servidor, sendo esta sua principal característica em relação ao NFS (TANENBAUM, 2002).

A arquitetura interna de uma estação de trabalho do AFS é ilustrada na Figura 3.7. Assim como no NFS, ele faz uso da interface VFS que intercepta todas as chamadas do cliente e as encaminha ao sistema de arquivos local ou ao processo *Venus*, que por sua vez, estabelece conexão com o servidor *Vice* através das chamadas de procedimentos remotos (TANENBAUM, 2002).

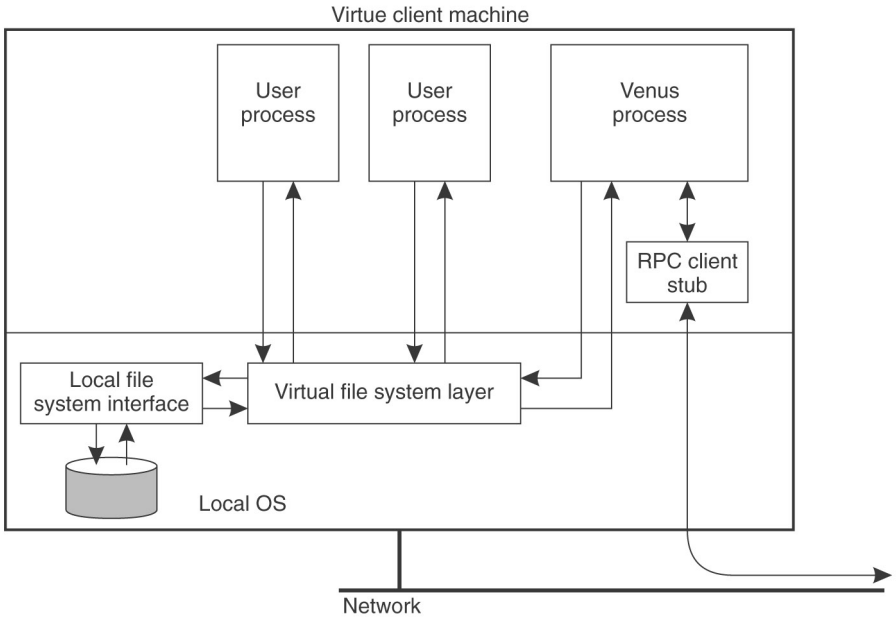

Figura 3.7 Arquitetura interna de uma estação de trabalho do AFS
Fonte: Tanenbaum, 2002, p. 606

O Coda File System disponibiliza aos usuários uma visão de um sistema de arquivos local e suporta todas as operações do sistema operacional desde que sigam as especificações da interface VFS (KLEIMAN, apud TANENBAUM, 2002), que é semelhante à ilustrada na Figura 3.7 (TANENBAUM, 2002).

3.2.2 Comunicação

A comunicação entre os processos do Coda File System é realizada através de chamadas de procedimentos remotos (TANENBAUM, 2002). Entretanto, o Coda File System possui uma implementação mais sofisticada do RPC, denominada de RPC2, que oferece chamadas de procedimentos remotos seguras sobre um protocolo UDP (TANENBAUM, 2002).

A cada chamada de procedimento remoto, o cliente do RPC2 inicia um processo que envia um pedido de requisição do servidor e subseqüentemente os blocos da solicitação até receber uma resposta (TANENBAUM, 2002). Como o processamento do pedido pode levar um tempo arbitrário para ser concluído, o servidor envia regularmente mensagens ao cliente para que este saiba que seu pedido esta sendo processado. Caso ocorra uma falha de comunicação entre o cliente e o servidor, o processo iniciado no computador cliente perceberá que as mensagens cessaram e notificará a aplicação que o invocou do fracasso da operação (TANENBAUM, 2002).

3.2.3 Processos

O Coda File System mantém uma distinção entre os processos do cliente, representados através do *Venus* e processos do servidor, representados através do *Vice*, sendo que ambos utilizam uma biblioteca de *threads* não-preemptiva para permitir que as requisições sejam processadas concorrentemente no cliente (onde vários processos de usuário podem ter requisições de acesso a arquivo em andamento concorrentemente) e no servidor (TANENBAUM, 2002).

3.2.4 Serviço de Nomes

Os arquivos no Coda File System agrupam-se em unidades denominadas de volumes (TANENBAUM, 2002). Os volumes se assemelham a uma partição do sistema operacional, mas geralmente possuem uma granularidade muito menor e correspondem a uma coleção de arquivos associados a um usuário. Os volumes são importantes por duas razões. Primeiro porque eles formam a unidade básica pela qual o espaço de nomes é constituído e esta construção ocorre quando eles são montados na estação cliente. A segunda razão é que eles formam a unidade que será replicada entre os servidores da rede (TANENBAUM, 2002).

É importante notar que, diferente do NFS, o Coda File System possui um espaço de nomes compartilhado (TANENBAUM, 2002). A Figura 3.8 representa esse compartilhamento exemplificando um ambiente com o AFS.

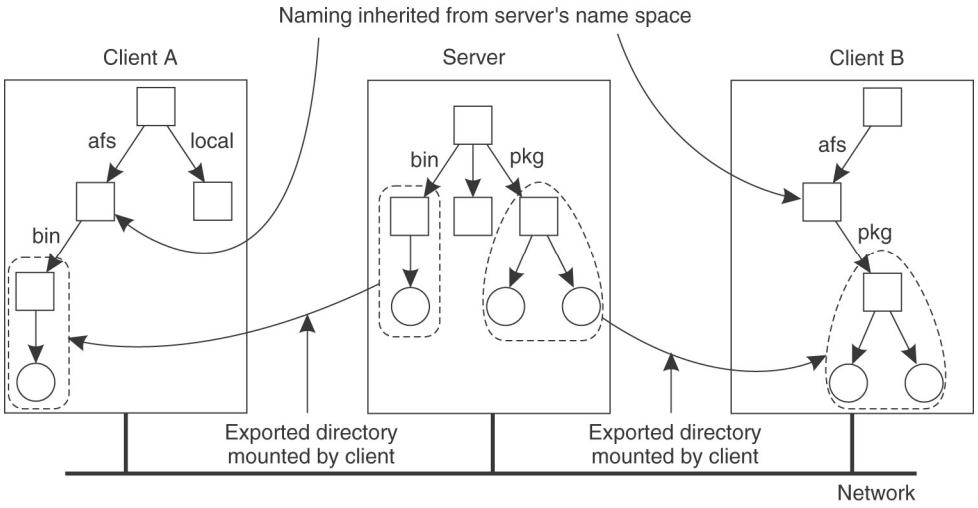

Figura 3.8 Exemplo do espaço de nomes compartilhado oferecido pelo Coda File System
Fonte: Tanenbaum, 2002, p. 610

No Coda File System, os clientes tem seus volumes do sistema de arquivos distribuído montado abaixo do diretório */coda*, similar ao seu antecessor (BRAAM, 1994). Qualquer arquivo compartilhado por algum servidor do Coda File System estará disponível neste ponto de montagem para todos os clientes. Diferente do NFS, os usuários se conectam ao serviço fornecido pelo Coda File System e não individualmente nos servidores, que são adicionados como membros do serviço de forma transparente (BRAAM, 1994).

3.2.5 Cache e Replicação

O *cache* e a replicação são importantes implementações do Coda File System, sendo elas fundamentais para o suporte a alta disponibilidade da qual a ferramenta se propõe (TANENBAUM, 2002).

Quando um arquivo é aberto pelo cliente do Coda File System para leitura ou escrita, este é transferido e armazenado integralmente no computador local com o intuito de aumentar o grau de tolerância a falhas fazendo com que o cliente não fique dependente da disponibilidade do servidor (TANENBAUM, 2002). Quando o processo *Vice* fornece uma cópia de um arquivo para um processo *Venus*, ele também fornece uma promessa de *callback*, ou seja, um *token* emitido pelo servidor garantindo que ele notificará o processo cliente

quando qualquer outro usuário modificar o arquivo. Quando um servidor realiza uma requisição de atualização de um arquivo, ele notifica todos os processos *Venus* para os quais tem promessas de *callback* emitidas, enviando um *callback* (uma chamada de procedimento remoto de um servidor para um processo *Venus*) notificando-os que o arquivo armazenado em *cache* esta desatualizado (TANENBAUM, 2002).

Caso o processo *Venus* receba uma operação para abrir determinado arquivo, ele verificará primeiramente sua existência em *cache* e caso seja encontrado, ele verificará se não recebeu nenhum *callback* de notificação quanto a sua desatualização (TANENBAUM, 2002). Caso o arquivo que esta armazenado em *cache* esteja desatualizado, então uma nova cópia é resgatada do servidor, mas caso o arquivo não tenha sofrido alterações, a cópia armazenada em *cache* é aberta e utilizada sem referencia ao servidor (TANENBAUM, 2002).

No Coda File System, os volumes podem ser replicados entre vários servidores, diferente de seu antecessor (TANENBAUM, 2002). Cada volume é associado a um grupo denominado de *Volume Storage Group* (VSG), que consiste em um conjunto de servidores que armazenam as réplicas do volume. O conjunto de servidores acessíveis de um certo grupo em um certo momento é chamado de *Accessible Volume Storage Group* (AVSG) (TANENBAUM, 2002).

Quando um cliente necessita acessar um arquivo, ele contata um membro do AVSG que contenha em seu volume o arquivo solicitado (TANENBAUM, 2002). Caso este tenha sofrido alterações, após seu fechamento pelo cliente, o arquivo é transferido em paralelo para todos os servidores do AVSG e posteriormente para os demais servidores do VSG (TANENBAUM, 2002; BRAAM, 1994).

Quando um servidor reintegra a rede após uma falha, nenhuma providência é tomada para a atualização dos arquivos (KON, 1994). Os arquivos somente serão atualizados neste servidor caso seja realizada uma requisição de acesso a uma versão mais recente. Se o cliente descobre que o seu servidor está com uma versão antiga do arquivo, o cliente solicita a versão mais recente de quem a possuir e emite uma mensagem ao AVSG informando da existência de uma versão desatualizada (KON, 1994).

3.2.6 Tolerância a Falhas

A arquitetura do Coda File System foi projetada para proporcionar que o sistema não seja sensível a partições na rede, garantindo ao usuário uma alta disponibilidade do serviço (TANENBAUM, 2002). Este recurso é alcançado com a implementação de *cache* no cliente, possibilitando que o usuário manipule arquivos mesmo com a presença de falhas ou instabilidades na rede, e com a implementação da replicação de volumes em diversos servidores, garantindo uma maior disponibilidade dos computadores fornecedores do serviço (TANENBAUM, 2002). A implementação e o uso desses recursos na ferramenta é mencionado na Seção 3.2.5.

3.3 Outras Soluções

Esta seção tem como objetivo apresentar soluções de armazenamento distribuído que vem sendo desenvolvidas e mantidas por empresas de renome, como a IBM, a Microsoft e o Google.

3.3.1 Andrew File System

O Andrew File System é um ambiente de computação distribuída desenvolvido na Universidade de Carnegie Mellon com o apoio da IBM para uso como sistema de computação e informação do campus (COULOURIS, 2005). O projeto, cuja arquitetura é similar ao Coda File System, citado na Seção 3.2, reflete a intenção de suportar o compartilhamento de informações em larga escala, minimizando a comunicação entre cliente e servidor (COULOURIS, 2005). Esta funcionalidade foi alcançada através da adoção da semântica de sessão e pela transferência de arquivos inteiros entre computadores clientes e servidores, armazenando-os em *cache* nos computadores locais até que o servidor receba uma versão mais atualizada (COULOURIS, 2005).

A fim de limitar a possível falta de segurança no sistema, o AFS adota uma série de mecanismos como, por exemplo, o *Kerberos Authentication Server*, que permite a autenticação mútua de servidores e clientes (COULOURIS, 2005).

Inicialmente, o projeto foi implementado na Universidade de Carnegie Mellon em uma rede com estações de trabalho e servidores executando Unix BSD e o sistema

operacional Mach e, subseqüentemente, se tornou disponível em versões comerciais e de domínio público (COULOURIS, 2005).

3.3.2 Farsite

O Farsite é um sistema de arquivos distribuído que vem sendo desenvolvido nos laboratórios de pesquisa da Microsoft (ADYA, 2002). Ele disponibiliza aos usuários a visão de um sistema de arquivos centralizado enquanto que sua estrutura física encontra-se dispersa na rede em servidores e estações de trabalho (ADYA, 2002). O Farsite fornece os mesmos recursos de um sistema de arquivos centralizado (armazenamento confiável, espaço de nomes compartilhado, transparência de acesso e transparência de localização) e os benefícios de um sistema de arquivos local (baixo custo e segurança no acesso aos arquivos) (ADYA, 2002).

Em termos de arquitetura, o Farsite agrupa os computadores no que ele denomina de grupos de diretórios (ADYA, 2002). Todos os integrantes dos grupos de diretórios têm como função armazenar réplicas dos arquivos que o grupo disponibiliza a rede, atender requisições de forma determinística, atualizar as réplicas e enviar respostas aos usuários (ADYA, 2002). Quando o cliente solicita acesso a um arquivo, ele envia uma mensagem a um determinado grupo de diretórios que responde com o conteúdo do arquivo pedido (ADYA, 2002). Caso o usuário atualize esse arquivo, ele envia a atualização ao grupo, que internamente fará o trabalho de replicação (ADYA, 2002).

O Farsite possibilita que os grupos de diretórios deleguem porções de arquivos aos demais grupos da rede, possibilitando um balanceamento de carga e aumentando a performance do serviço (ADYA, 2002). Além desses recursos, o Farsite implementa suporte a *cache* no cliente para o aumento de performance e a criptografia de arquivos armazenados na rede como meio de segurança (ADYA, 2002).

3.3.3 Google File System

O Google File System é um sistema de arquivos distribuído desenvolvido e estendido extensamente dentro da empresa Google como plataforma de armazenamento, onde o maior agrupamento de servidores existente para dados provê centenas de *Terabytes* através de milhares de discos rígidos em mais de mil máquinas que são acessadas simultaneamente por centenas de clientes (GHEMAWAT, 2003). O sistema compartilha de muitos dos mesmos

objetivos dos demais sistemas de arquivos distribuídos, tais como escalabilidade, desempenho, confiança e disponibilidade, que foram implementados levando em consideração as necessidades da empresa e o ambiente tecnológico utilizado por ela (GHEMAWAT, 2003).

Sua arquitetura é composta por um único servidor principal e diversos computadores que armazenam os arquivos acessados pelos usuário (GHEMAWAT, 2003). Assim como outras soluções de armazenamento distribuído, o Google File System possui um relacionamento cliente-servidor simétrico, sendo possível que uma estação de trabalho atue na rede tanto como cliente, quanto como servidor (GHEMAWAT, 2003).

Quando um cliente necessita de um arquivo, ele entra em contato com o servidor principal que o orienta informando em qual computador esta localizado o arquivo solicitado (GHEMAWAT, 2003). A partir dai, todas as alterações realizadas pelo cliente no arquivo solicitado são encaminhadas diretamente ao computador indicado pelo servidor principal (GHEMAWAT, 2003). Além deste apontamento, o servidor principal é responsável por gerenciar a replicação dos arquivos e coordenar os processos de inclusão e remoção de computadores na rede (GHEMAWAT, 2003).

4 EXPERIMENTOS

A experimentação é uma disciplina muito importante para diversas áreas de pesquisa (WOHLIN et. al., 2000). Seu objetivo é construir uma base de conhecimento confiável e reduzir assim incertezas sobre quais teorias, ferramentas e metodologias são mais adequadas para certos contextos (WOHLIN et. al., 2000).

O objetivo dos experimentos neste trabalho é de possibilitar a identificação do grau de transparência, classificado em alto e baixo, existente nos sistemas de arquivos distribuídos. Para tanto, serão elaborados três cenários distintos, não exaustivos, para cada uma das transparências a serem estudadas (replicação, acesso e localização) onde, o critério para indicar o grau de transparência alto em determinada técnica é a obtenção do sucesso no respectivo cenário e o critério para indicar o grau de transparência baixo em determinada técnica é a não obtenção do sucesso no respectivo cenário. Em paralelo a estes ensaios, serão realizadas simulações para mensurar a eficiência destas ferramentas em processos de escrita e leitura de arquivos.

Este capítulo apresentará o ambiente construído para a realização dos experimentos, os cenários elaborados e os resultados obtidos com os experimentos.

4.1 Definição do Cenário

Os experimentos serão realizados em um ambiente em que os usuários usufruirão de serviços de armazenamento distribuído onde, através de qualquer computador cliente localizado na rede, conseguirão gerenciar seus arquivos de forma transparente, sem precisarem se preocupar com a localização física dos dados e com a utilização de programas específicos para a manipulação dessas informações. Fazendo-se uso de uma analogia, a proposta é similar às redes de energia elétrica, onde toda complexidade referente a geração,

transmissão e distribuição é oculta para o usuário, que simplesmente usufrui deste recurso para ligar equipamentos eletrônicos, sem precisar preocupar-se de onde a energia foi gerada e que caminhos ela percorreu para chegar até ele. Além da transparência de acesso e localização, o ambiente garantirá a replicação dos arquivos dos clientes com o intuito de fortalecer a disponibilidade do serviço.

4.2 Configuração do Cenário

Para a realização dos experimentos será utilizado um *notebook* e três computadores *desktop* com as configurações de *hardware* descritas nas tabelas abaixo. Estes computadores serão interligados por um *Hub* de 10 Mbps e todos possuirão a instalação padrão do Sistema Operacional Debian GNU/Linux 4.0 com Kernel 2.6.18-4.

Tabela 4.1 Configuração de *hardware* do computador A

Processador	Móbile AMD Sempron 2800+ 1600 MHz
Placa Mãe	SIS 760 GX
Memória RAM	512 MB (2 x 256 DDR-SDRAM)
Memória Cache L1	128 KB
Memória Cache L2	256 KB
Disco Rígido	40 GB, IDE, 4200 RPM
Placa de Rede	SIS 900 10/100 Mbps Ethernet PHY

Fonte: Tabela do Autor

Tabela 4.2 Configuração de *hardware* do computador B

Processador	AMD Athlon XP 2000 MHz
Placa Mãe	Asus A7V8X-X
Memória RAM	512 MB (1 x 512 DDR-SDRAM)
Memória Cache L1	128 KB
Memória Cache L2	256 KB
Disco Rígido	80 GB, IDE, 7200 RPM
Placa de Rede	Realtek 10/100 Mbps Ethernet PHY (Onboard)

Fonte: Tabela do Autor

Tabela 4.3 Configuração de *hardware* do computador C

Processador	Intel Celeron 2400 MHz
Placa Mãe	Intel Desktop Board D845PEMY
Memória RAM	256 MB (1 x 256 DDR-SDRAM)
Memória Cache L1	128 KB
Memória Cache L2	1024 KB

Disco Rígido	40 GB, IDE, 7200 RPM
Placa de Rede	Intel PRO 10/100 LAN Network Connection (Onboard)

Fonte: Tabela do Autor

Tabela 4.4 Configuração de *hardware* do computador D

Processador	Intel Celeron 2400 MHz
Placa Mãe	Intel Desktop Board D845PEMY
Memória RAM	256 MB (1 x 256 DDR-SDRAM)
Memória Cache L1	128 KB
Memória Cache L2	1024 KB
Disco Rígido	40 GB, IDE, 7200 RPM
Placa de Rede	Intel PRO 10/100 LAN Network Connection (Onboard)

Fonte: Tabela do Autor

Devido às semelhanças de *hardware*, os computadores C e D atuarão como servidores do serviço de armazenamento distribuído nos cenários propostos, enquanto que, os demais computadores, desempenharão o papel de clientes.

A Figura 4.1 ilustra o laboratório a ser construído para a realização dos experimentos. Nos ensaios da transparência de replicação e da transparência de acesso, o volume dos servidores será replicado entre os dois computadores, proporcionando aos clientes um espaço único de 5 GB tolerante a falhas para o armazenamento de arquivos. No ensaio da

transparência de localização, pretende-se agrupar o volume de ambos os servidores, oferecendo ao ambiente um espaço de armazenamento de 10 GB, sem tolerância a falhas. Em ambos os casos o espaço em disco definido pode variar dependendo da limitação das ferramentas.

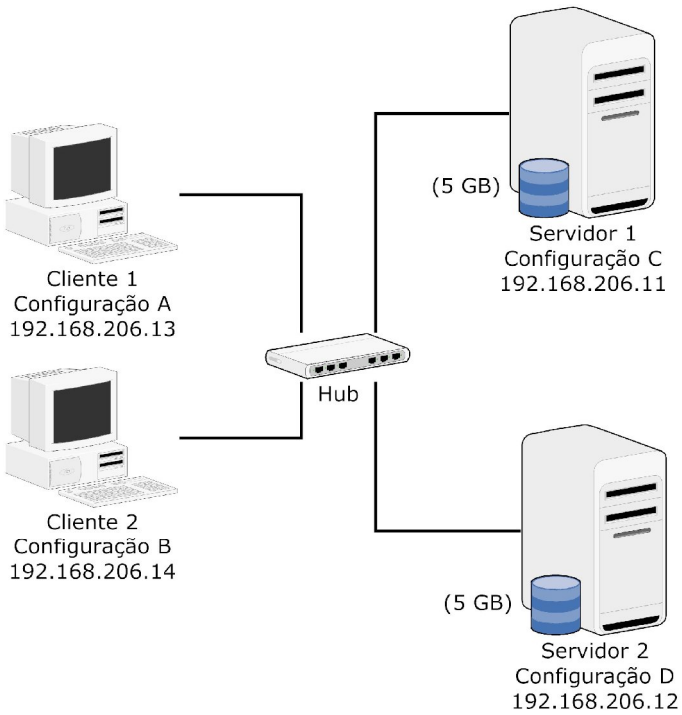

Figura 4.1 Laboratório utilizado para a realização dos experimentos
Fonte: Imagem do Autor

4.3 Cenários Elaborados

A seguir serão apresentados os cenários elaborados para identificar o grau da transparência de replicação, transparência de acesso e transparência de localização existente nos sistemas de arquivos distribuídos.

4.3.1 Transparência de Replicação

O cenário da transparência de replicação simulará a cópia de um arquivo local para o ponto de montagem criado no computador cliente que concede acesso ao sistema de arquivos distribuído e em seguida a falha de um dos servidores. Entretanto, caso o computador cliente continue tendo acesso ao arquivo recentemente copiado, mesmo com a ausência de um dos nodos, não representa conclusivamente a presença da técnica de replicação na ferramenta, pois o nodo que falhou não necessariamente é o nodo em que o cliente esta conectado. A Figura 4.2 ilustra este problema. Na representação, o Cliente 1 esta conectado com o Servidor

2 durante a manipulação de um determinado arquivo. Caso ocorra uma falha de comunicação entre os servidores antes da replicação do arquivo recentemente manuseado e em seguida o usuário estabeleça novamente uma conexão com o Servidor 2 para ter acesso ao arquivo, este será bem sucedido, uma vez que não ocorreram problemas de comunicação entre esses nodos.

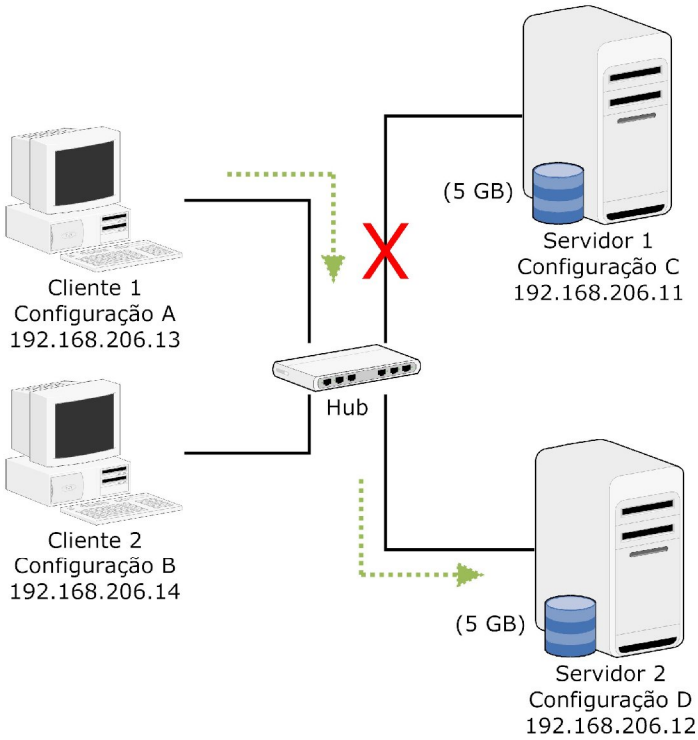

Figura 4.2 Representação da falha de comunicação entre os servidores
Fonte: Imagem do Autor

Para uma correta identificação do grau da transparência de replicação existente nas ferramentas, é necessário garantir que a situação acima não ocorrerá, ou seja, que o servidor em que o cliente estabeleceu a conexão seja o servidor que sofrerá a falha. Para isso, é necessário que, antes de simular a falha de um dos servidores, haja a certeza de em qual computador o arquivo será armazenado antes da replicação. Uma possível solução para esta situação seria a construção de um *cluster* de alta disponibilidade (*failover*) entre os servidores, onde apenas um nodo receberia as requisições enquanto o outro permaneceria ocioso até que o primeiro falhasse. Entretanto, a construção desse cenário dependerá da existência desse recurso nas ferramentas selecionadas.

Obtendo a solução para a situação acima, cabe então a realização do experimento. A proposta deste ensaio é de executar no cliente uma linha de comando similar a apresentada na Figura 4.3, que copia um arquivo no tamanho aproximado de 1 MB para o ponto de montagem que concede acesso ao sistema de arquivos distribuído e após o intervalo de 5

segundos estabelece uma conexão remota com o servidor em que o cliente esta conectado, desabilitando todos os dispositivos de rede e em seguida forçando seu desligamento, simulando assim uma falha.

```
cp /tmp/arquivo.tar.gz /mnt/; sleep 5; ssh servidor1 poweroff –I -f
```

Figura 4.3 Procedimento para ensaio de replicação
Fonte: Imagem do Autor

Para o cenário ser bem sucedido, a replicação entre os servidores deverá ocorrer no intervalo de 5 segundos e em seguida, após o desligamento do computador, o cliente deverá conseguir ter acesso ao arquivo através do ponto de montagem para o qual ele foi copiado.

A latência de 5 segundos para a replicação de um arquivo no tamanho aproximado de 1 MB proposta para este experimento não foi baseada em bibliografias, sendo considerada satisfatória no escopo deste trabalho levando em consideração que a replicação deverá ocorrer em uma rede de 10 Mbps. Outros ensaios poderão ser realizados seguindo este mesmo foco para se obter resultados mais precisos, transferindo arquivos de tamanhos maiores e diminuindo o intervalo entre a cópia e a desativação das interfaces de rede do servidor.

4.3.2 Transparência de Acesso

Como já citado na Seção 1.1.6, a transparência de acesso possibilita que os recursos remotos de um sistema de arquivos distribuído sejam acessados de forma semelhante aos recursos de um sistema local. Sendo assim, para identificar o grau da transparência de acesso existente nas ferramentas, será executado um conjunto de programas, apresentados na Tabela 4.5, presentes na instalação padrão na maioria das distribuições do Sistema Operacional Linux.

Tabela 4.5 Programas utilizados para a experimentação da transparência de acesso

ls	Lista o conteúdo de determinado diretório.
cd	Altera o diretório corrente.

mkdir	Cria um diretório.
rm	Exclui arquivos ou diretórios.
mv	Move ou altera o nome de um arquivo.
touch	Altera o *timestamp* de um arquivo. Para o experimento, este comando será utilizado unicamente para a criação de um arquivo vazio.

Fonte: Tabela do Autor

Para automatizar esta execução e simular acessos seqüenciais de forma simétrica entre as ferramentas, é proposto a execução da linha de comando apresentada na Figura 4.4, que contempla a execução de todos os programas acima citados.

```
for i in $(seq 500); do mkdir $i; cd $i; touch $i.txt; mv $i.txt $i;
ls; cd ..; rm –rf $i; done
```

Figura 4.4 Procedimento para ensaio da transparência de acesso
Fonte: Imagem do Autor

Esta linha de comando será executada no ponto de montagem do sistema de arquivos distribuído e para o experimento ser bem sucedido, a execução dela não deverá gerar nenhuma exceção para o sistema.

4.3.3 Transparência de Localização

A transparência de localização pode existir em diversos aspectos em sistemas distribuídos. No escopo deste trabalho, a transparência de localização pode ser identificada com a realização de pelo menos dois ensaios. O primeiro é realizado durante os experimentos da transparência de replicação, onde o ambiente proposto disponibiliza aos clientes um serviço de armazenamento distribuído tolerante a falhas. Neste cenário, o cliente não precisa se preocupar com a localização física (entre os volumes dos servidores) dos arquivos que ele esta manipulando e caso um nodo falhe, suas requisições serão encaminhadas a um nodo ativo, muitas vezes sem o conhecimento do cliente, que continuará a usufruir dos serviços

desta vez localizados em outro servidor. Pode-se afirmar que, dependendo do resultado obtido com os experimentos da transparência de replicação, a transparência de localização estará presente na ferramenta.

A proposta do segundo ensaio é construir um laboratório similar ao ilustrado na Figura 4.1, onde o volume dos servidores será agrupado ao invés de replicado, fornecendo um espaço de armazenamento de 10 GB, similar a proposta das ferramentas de armazenamento distribuído em *Grid* (TAURION, 2004), que oferece uma alta escalabilidade do ambiente em termos de armazenamento. Caso as ferramentas a serem selecionadas proporcionem a construção deste ambiente, os testes se resumirão na movimentação de arquivos entre os discos dos servidores. Este ensaio será considerado bem sucedido se, após a movimentação de determinado arquivo entre os servidores, o cliente continue tendo acesso a ele fazendo uso do mesmo caminho utilizado até então.

4.4 Testes de Desempenho

Testes de desempenho, quando aplicados a sistemas de arquivos, indicam como o sistema reage a certas operações, como escrita e leitura.

A proposta dos testes de desempenho neste trabalho é de comparar a performance de escrita e leitura entre sistemas de arquivos distribuídos e um sistema de arquivos local, que para este experimento será o Ext3. Todos os testes de performance entre os sistema de arquivos serão realizados no computador de configuração B e para mensurar o desempenho no sistema de arquivos local, será adicionado um segundo disco rígido de 40 GB de espaço, interface IDE e de 4200 RPM para que não ocorram interferências do sistema operacional durante o processo, aumentando assim a exatidão dos resultados.

A ferramenta escolhida para a realização deste *benchmark* foi o IOzone (CILIENDO, 2007), uma vez que não foram encontradas soluções para mensurar desempenho em sistemas de arquivos distribuídos. O IOzone permite a criação de arquivos seqüências de diferentes tamanhos e a simulação de gravação em blocos de tamanhos variados (CILIENDO, 2007). Neste trabalho, serão aplicados os testes apresentados na Tabela 4.6.

Tabela 4.6 Testes de performance a serem realizados

Escrita	Mensura a performance de escrita de um arquivo novo.
Reescrita	Mensura a performance de escrita de um arquivo já existente. Normalmente, o desempenho da reescrita é superior ao desempenho da escrita de um arquivo novo, uma vez que seus metadados já estão criados.
Leitura	Mensura a performance de leitura de um arquivo existente.
Releitura	Mensura a performance de leitura de um arquivo recentemente lido. Normalmente, o desempenho da releitura é superior ao desempenho de leitura, uma vez que o sistema operacional armazena em *cache* os dados de arquivos recentemente lidos.

Fonte: Tabela do Autor

A Tabela 4.7 apresenta os parâmetros a serem utilizados na chamada do IOzone.

Tabela 4.7 Parâmetros utilizados na chamada do IOzone

-c	Mensura o tempo consumido para a execução dos testes até o fechamento do arquivo.
-t	Permite definir quantas *threads* ou processos estarão ativos durante os testes.
-F	Define o caminho e o nome do arquivo de testes a ser gerado. A quantidade de arquivos informados para este parâmetro deve ser igual à quantidade de *threads* ou processos definidos.

-i 0	Especifica a realização de testes de escrita e reescrita.
-i 1	Especifica a realização de testes de leitura e releitura.
-r	Define o tamanho dos blocos de registros a serem gravados.
-s	Define o tamanho do arquivo de testes a ser manipulado.

Fonte: Tabela do Autor

4.5 Escolha das Ferramentas

Alguns critérios foram previamente estabelecidos para a escolha das ferramentas que fariam parte dos experimentos. O primeiro diz respeito a seu licenciamento. Foi imprescindível que as soluções de sistemas de arquivos distribuídos estivessem enquadradas em uma licença que permitisse a liberdade de execução do programa para a realização deste estudo.

O segundo critério estabelecido foi a eliminação de projetos descontinuados, para que o andamento deste trabalho não fosse prejudicado pela falta de suporte. Para tanto, levou-se em consideração a data de lançamento das últimas versões e correções dos projetos, além do índice de movimentação das listas de discussão, caso houvesse.

Esses critérios foram atendidos plenamente pelas seguintes ferramentas: Network File System, Andrew File System e Coda File System. Entre essas, optou-se pela primeira e a terceira solução citada. O Andrew File System foi dispensado uma vez que sua arquitetura é similar a do Coda File System, que durante o processo de escolha aparentou estar mais maduro e evoluído em relação a seu antecessor devido ao suporte a operações desconectadas.

A versão das ferramentas a serem utilizadas é ilustrada na Tabela 4.8.

Tabela 4.8 Sistemas de arquivos distribuídos utilizados nas experimentações

Ferramenta	Versão
Coda File System	6.9.1
Network File System	4

Fonte: Tabela do Autor

4.6 Resultados Obtidos

Esta seção apresentará os resultados obtidos com a aplicabilidade dos cenários elaborados e os resultados dos testes de desempenho entre os sistemas de arquivos.

4.6.1 Transparência de Replicação

Antes de iniciar a aplicação do cenário criado para identificar o grau da transparência de replicação dos sistemas de arquivos distribuídos, foi necessário encontrar uma alternativa para contornar o problema mencionado na Seção 4.3.1, tendo-se que identificar primeiramente em qual servidor o computador cliente estava estabelecendo conexão.

No Coda File System, o programa cliente do sistema de arquivos distribuído possui um arquivo de configuração denominado de *realms*, que permite definir células para o agrupamento de servidores cujos volumes se replicam (HARKES, 2004). O conteúdo do arquivo *realms* utilizado pelas estações clientes neste experimento é apresentado na Figura 4.5. Nele, há uma célula nomeada de “codaserver” que representa o acesso aos servidores “codaserver1” e “codaserver2”.

```
codaclient1:~# cat /etc/coda/realms
codaserver          codaserver1 codaserver2
codaclient1:~# _
```

Figura 4.5 Conteúdo do arquivo *realms* utilizado nas estações clientes do Coda File System
Fonte: Imagem do Autor

Quando o cliente estabelece conexão com o sistema de arquivos distribuído (ilustrado na Figura 4.6), ele informa, além de suas credenciais, o nome da célula definida no

arquivo de configuração. Durante o processo de conexão com o serviço, o cliente tentará estabelecer acesso com o primeiro servidor descrito logo após o nome da célula.

```
codaclient1:~# clog codaroot@codaserver
username: codaroot@codaserver
Password:
codaclient1:~# _
```

Figura 4.6 Processo de conexão com o serviço do Coda File System
Fonte: Imagem do Autor

Caso este não esteja disponível e caso haja outro servidor informado logo em seguida, como é o caso deste ambiente, o cliente tentará restabelecer uma nova conexão com o serviço, dessa vez acessando outro servidor. Partindo desta premissa, é possível definir no Coda File System em qual servidor o cliente estabelecerá acesso para a realização dos experimentos.

No Network File System, a identificação do servidor que o usuário estabelecerá conexão é informada no processo de criação do ponto de montagem, a exemplo da Figura 4.7.

```
nfsclient1:~# mount -t nfs4 nfsserver1:/teste /mnt/
nfsclient1:~# _
```

Figura 4.7 Processo de conexão com o serviço do Network File System
Fonte: Imagem do Autor

Sabendo em quais servidores ambos os clientes estabelecerão acesso, pode-se iniciar a aplicação do experimento. Conforme mencionado na Seção 4.3.1, este cenário se resume na execução de um comando similar ao proposto na Figura 4.3 e na análise do comportamento das ferramentas.

A Figura 4.8 ilustra a chamada deste procedimento no cliente do Coda File System e o resultado obtido após sua execução. Devido ao fato do cliente do Coda File System possuir um *cache* que oferece o suporte a operações desconectadas, o que poderia invalidar este experimento, sentiu-se a necessidade de averiguar no segundo servidor a existência do arquivo criado. É possível perceber na Figura 4.8 que a conexão do cliente foi migrada, de forma transparente, para um segundo servidor pertencente a célula em que se estabeleceu conexão, mascarando assim a falha ocorrida.

O resultado da aplicação deste cenário no Network File System é ilustrado na Figura 4.9. Percebe-se que, diferente do Coda File System, o Network File System não migrou a

conexão do cliente para um servidor que estivesse disponível para atendê-lo, sendo necessário recriar o ponto de montagem apontando-o para um segundo servidor pertencente a política de replicação. Acredita-se que esta limitação é caracterizada pela ausência de um espaço de nomes globais na arquitetura do Network File System, entretanto, algumas pesquisas vem sendo desenvolvidas propondo a implementação deste recurso (ZHANG, 2003).

```
codaclient1:~# cat /etc/coda/realms
codaserver          codaserver1 codaserver2
codaclient1:~# ls -la /tmp/p361-nightingale.pdf
-rw-r--r-- 1 root root 1114824 2007-06-06 11:45 /tmp/p361-nightingale.pdf
codaclient1:~# ls -la /coda/codaserver/
total 4
drwxr-xr-x 1 root nogroup 2048 2007-06-06 12:07 .
dr-xr-xr-x 1 root nogroup 2048 2007-06-06 11:45 ..
codaclient1:~# cp /tmp/p361-nightingale.pdf /coda/codaserver/; sleep 5; ssh coda
server1 poweroff -i -f &
[2] 2280
codaclient1:~# setterm: $TERM is not defined.
[2]+  Stopped                 ssh codaserver1 poweroff -i -f
codaclient1:~# ls -la /coda/codaserver/
total 1093
drwxr-xr-x 1 root nogroup    2048 2007-06-06 13:05 .
dr-xr-xr-x 1 root nogroup    2048 2007-06-06 11:45 ..
-rw-r--r-- 1 root nogroup 1114824 2007-06-06 13:05 p361-nightingale.pdf
codaclient1:~# ssh codaserver2 ls -la /vicepa/0/0
setterm: $TERM is not defined.
total 1104
drwx------ 2 root root    4096 2007-06-06 13:07 .
drwx------ 3 root root    4096 2007-06-06 09:57 ..
-rw------- 1 root root 1114824 2007-06-06 13:05 1
codaclient1:~# _
```

Figura 4.8 Resultado obtido com a experimentação no Coda File System
Fonte: Imagem do Autor

```
nfsclient1:~# mount -t nfs4 nfsserver1:/tmp/ /mnt/
nfsclient1:~# ls -la /mnt/
total 8
drwxrwxrwt  2 root root 4096 2007-06-06 15:54 .
drwxr-xr-x 23 root root 4096 2007-06-06 06:12 ..
nfsclient1:~# ls -la /tmp/p361-nightingale.pdf
-rw-r--r-- 1 root root 1114824 2007-06-06 15:55 /tmp/p361-nightingale.pdf
nfsclient1:~# cp /tmp/p361-nightingale.pdf /mnt/; sleep 5; ssh nfsserver1 powero
ff -i -f &
[1] 2216
nfsclient1:~# setterm: $TERM is not defined.
[1]+  Stopped                 ssh nfsserver1 poweroff -i -f
nfsclient1:~# ls -la /mnt/
umount: /mnt: device is busy
umount: /mnt: device is busy
nfsclient1:~# umount /mnt/
nfsclient1:~# mount -t nfs4 nfsserver2:/tmp/ /mnt/
nfsclient1:~# ls -la /mnt/
total 1104
drwxrwxrwt  2 root   root       4096 2007-06-06 15:33 .
drwxr-xr-x 23 root   root       4096 2007-06-06 06:12 ..
-rw-r--r--  1 nobody nogroup 1114824 2007-06-06 15:57 p361-nightingale.pdf
nfsclient1:~# _
```

Figura 4.9 Resultado obtido com a experimentação no Network File System
Fonte: Imagem do Autor

Na Seção 3.1.5 e na especificação do protocolo da versão 4 do Network File System é mencionado a abordagem a ser utilizada quando ocorrerem falhas de acesso a servidores replicados, entretanto, esta especificação aparenta não estar implementada (FIELDS, 2006).

Com os resultados obtidos neste cenário pode-se concluir que o Network File System possui uma baixa transparência de replicação, pois apesar da ferramenta possibilitar que mais de uma instância do recurso seja utilizada para aumentar a confiabilidade do serviço, esta só pode ser acessada mediante conhecimento prévio da réplica pelo cliente, diferente do Coda File System, que neste cenário obteve uma alta transparência de replicação por ter migrado a conexão do cliente para a segunda instância da réplica do serviço após a ocorrência de falha.

4.6.2 Transparência de Acesso

A aplicação do cenário para identificar o grau da transparência de acesso nos sistemas de arquivos distribuídos analisados foi bem sucedida em ambas as ferramentas, não havendo a necessidade de modificar os programas utilizados nos experimentos para permitir que eles funcionassem corretamente com os arquivos remotos. A Figura 4.10 exemplifica a chamada do comando proposto sendo executado no Network File System para identificar a existência da transparência de acesso. Sua execução exibe na tela números de 1 até 500.

```
nfsclient1:/mnt# for i in $(seq 500); do mkdir $i; cd $i; touch $i.txt; mv $i.tx
t $i; ls; cd ..; rm -rf $i; done
1
2
3
4
5
6
```

Figura 4.10 Chamada do comando proposto para identificar a transparência de acesso
Fonte: Imagem do Autor

Devido ao fato de não terem ocorrido exceções na execução do comando (a Figura 4.11 ilustra o término do processamento) concluiu-se que o grau da transparência de acesso em ambas as ferramentas estudadas é alto.

```
494
495
496
497
498
499
500
nfsclient1:/mnt# _
```

Figura 4.11 Fim do processamento do comando proposto para identificar a transparência de acesso
Fonte: Imagem do Autor

Acredita-se que o resultado no sucesso desse experimento deu-se devido ao fato das ferramentas analisadas fazerem uso em sua arquitetura da interface VFS, possibilitando que chamadas de sistemas genéricas (como *open()* e *read()*) possam ser executadas independente do sistema de arquivos usado ou do meio físico.

4.6.3 Transparência de Localização

Os sistemas de arquivos distribuídos selecionados impossibilitaram a construção do ambiente proposto na Seção 4.3.3. A possibilidade de elaborar um cenário similar ao proposto neste trabalho é questionada na lista de discussão do Coda File System, onde mantenedores do projeto comentam da sua inviabilidade (HARKES, 1999; TROXEL, 2007). Algumas alternativas similares são sugeridas (HARKES, 1999) e poderiam ser implementadas em ambas as ferramentas, mas invalidariam o resultado.

Apesar deste imprevisto e baseando-se na experiência obtida com a aplicação do cenário da transparência de replicação e da transparência de acesso, pode-se concluir que o Network File System possui uma baixa transparência de localização, enquanto que o Coda File System possui uma alta transparência de localização. Esta afirmativa leva em consideração o sucesso obtido nos experimentos da transparência de replicação com o Coda File System. O fato da conexão do cliente ter sido migrada para outro servidor sem a percepção do cliente quando foi simulada a falha de um dos nodos demonstra a transparência que se obteve nesse processo, que se assemelha, de certa forma, ao caso proposto na Seção 4.3.3.

No Network File System, a baixa transparência de localização também é identificada pelo fato do cliente ter que especificar no processo de montagem o nome do servidor que armazena os arquivos, uma vez que a arquitetura da ferramenta não disponibiliza um espaço de nomes globais. Como mencionado na Seção 1.1.6, a transparência de localização deve possibilitar o acesso ao recurso pelo usuário sem que este tenha que se preocupar com sua localização na rede. Assim como para o problema de migração da sessão do cliente ocorrido no experimento da transparência de replicação, propostas para implementar transparência de localização na versão 4 do Network File System vem sendo discutidas e desenvolvidas (ZHANG, 2003).

4.6.4 Testes de Desempenho

Conforme já mencionado na Seção 4.4, todos os testes de desempenho partiram de um mesmo computador que possuía instalado, além da ferramenta de *benchmark*, o cliente do sistema de arquivos distribuído que estava em avaliação.

Objetivando obter resultados exatos para uma comparação eqüitativa entre os sistemas de arquivos, teve-se a preocupação em considerar três fatores antes da realização das simulações, que poderiam contribuir para a apresentação de resultados incorretos: o uso do *cache* na estação cliente, o tamanho dos blocos das partições e o tamanho dos blocos de dados enviados entre o cliente e o servidor através da rede.

O IOzone possibilita a realização de *benchmark* simulando mais de uma *thread* ou processo simultaneamente (CILIENDO, 2007), proporcionando mensurar o desempenho na manipulação de arquivos de forma concorrente. Entretanto, para fazer uso desta funcionalidade, deve-se empregar o parâmetro *-I* na invocação da ferramenta, que invalida a utilização do *cache* no computador cliente, garantindo assim que todas as operações sejam realizadas no disco rígido (CILIENDO, 2007).

Nos ensaios realizados, não foi possível fazer uso deste parâmetro em um ponto de montagem do Coda File System, pois os arquivos criados pelo IOzone durante os testes eram estabelecidos como somente-leitura, permitindo seu acesso apenas pela ferramenta de *benchmark* (HARKES, 2007) e o impossibilitando para o cliente do sistema de arquivos distribuído, que apresentava erros constantes negando seu acesso a ele, o que impedia a conclusão do processo. Este problema não se refletiu durante os ensaios com o Network File System, mas mesmo assim, optou-se por não fazer uso deste parâmetro na avaliação das ferramentas, acarretando a busca por uma alternativa que pudesse ser aplicada em ambas as soluções.

Devido a estes contratempos, definiu-se que o *benchmark* seria realizado com a execução de apenas um processo e entre os testes com os sistemas de arquivos, a estação cliente seria reinicializada, evitando assim a utilização do *cache*.

Outras duas preocupações que surgiram antes da realização das simulações foi quanto ao tamanho dos blocos da partição dos computadores, que neste experimento eram de 4 KB, e o tamanho dos blocos de informações enviados entre o cliente e o servidor. A

proposta inicial era de simular, tanto no sistema de arquivos local quanto nos sistemas de arquivos distribuídos, a gravação de arquivos em blocos de 4 KB. Entretanto, este ensaio não foi bem sucedido no Coda File System, que apresentava *time-out* no cliente quando era solicitado ao IOzone a manipulação de arquivos demasiadamente grandes simulando a utilização de blocos de dados pequenos. Assim, optou-se por simular a gravação em blocos de dados de 10 MB em sistemas de arquivos distribuídos, deixando assim a própria ferramenta gerenciar o tamanho das mensagens a serem enviadas, e blocos de dados de 4 KB para sistema de arquivos local, uma vez que este era o tamanho em que as partições estavam formatadas.

A linha de comando utilizada para mensurar o desempenho do sistema de arquivos local é apresentada na Figura 4.12, e a linha de comando utilizada para mensurar o desempenho dos sistemas de arquivos distribuídos é apresentada na Figura 4.13.

```
iozone -c -t 1 -F /tmp/ -i 0 -i 1 -r 4k -s 1g
```

Figura 4.12 Chamada do IOzone utilizada em sistemas de arquivos locais
Fonte: Imagem do Autor

```
iozone -c -t 1 -F /mnt/ -i 0 -i 1 -r 10m -s 1g
```

Figura 4.13 Chamada do IOzone utilizada em sistemas de arquivos distribuídos
Fonte: Imagem do Autor

A Figura 4.14 apresenta o ambiente utilizado para a realização dos testes de desempenho que possuía, além da estação cliente, apenas um servidor oferecendo o serviço de armazenamento distribuído sem o recurso de replicação. Todos os testes realizados simularam a manipulação de arquivos no tamanho de 1 GB.

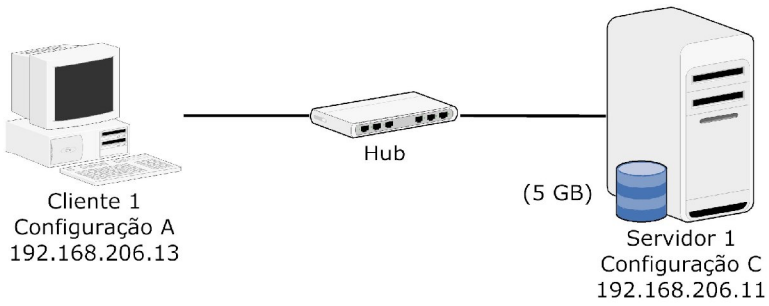

Figura 4.14 Laboratório utilizado para a realização dos benchmarks
Fonte: Imagem do Autor

A fim de identificar possíveis divergências nos resultados, foi mensurado primeiramente a performance do sistema de arquivos local com blocos no tamanho de 4 KB e 10 MB. A Figura 4.15 ilustra o resultado obtido.

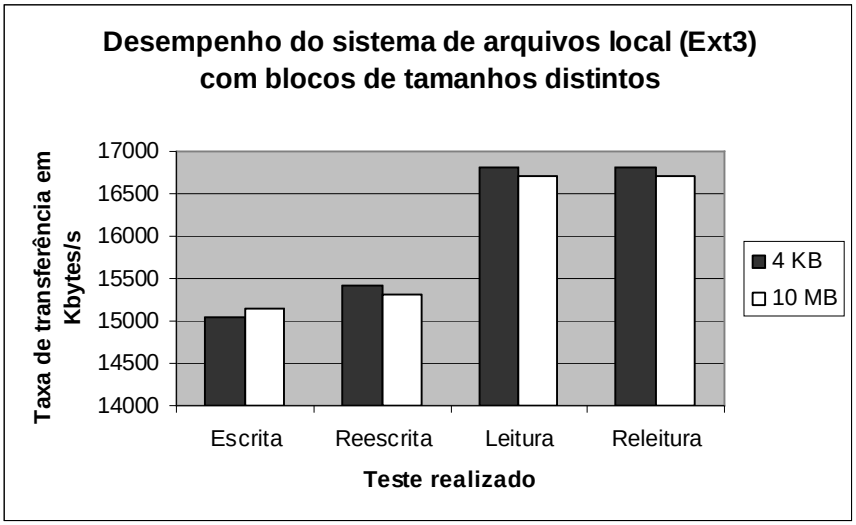

Figura 4.15 Comparativo de desempenho do sistema de arquivos local com blocos de tamanhos distintos
Fonte: Imagem do Autor

Devido a pequena diferença gerada, os gráficos a seguir representarão a performance obtida com o sistema de arquivos Ext3 formatado em blocos no tamanho de 4 KB. A Figura 4.16 ilustra o primeiro comparativo realizado entre os sistemas de arquivos distribuídos e o sistema de arquivos local.

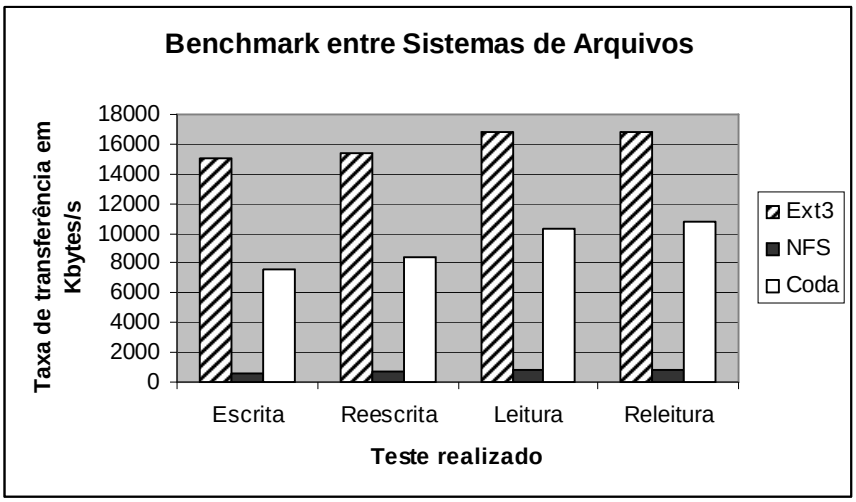

Figura 4.16 Comparativo de desempenho entre os sistemas de arquivos
Fonte: Imagem do Autor

A segunda comparação de performance realizada simulou um ambiente com excessivo tráfego na rede com o auxílio do Netperf (CILIENDO, 2007). O Netperf é uma ferramenta de *benchmark* para análise de tráfego em redes e sua arquitetura é baseada no

modelo cliente-servidor, possibilitando ao utilizador determinar a quantidade de tempo para a transferência de certa quantia de dados entre os computadores (CILIENDO, 2007).

Os parâmetros utilizados na chamada do Netperf são apresentados na Tabela 4.9.

Tabela 4.9 Parâmetros utilizados na chamada do Netperf

-p	Define o número da porta a ser utilizado.
-H	Especifica o nome ou endereço IP do computador remoto.
-t	Define o tipo de teste de tráfego a ser utilizado. Neste trabalho será utilizado o TCP_RR, que simula o tráfego de envio e recebimento medido pelo número de transações trocadas em segundos entre os nodos.
-r	Especifica (em Kbytes) o tamanho do pacote de envio e recebimento.

Fonte: Tabela do Autor

Para a realização deste ensaio, foi incluído um terceiro computador no diagrama apresentado na Figura 4.14. Este computador ficou como responsável pela geração do tráfego excessivo na rede, sendo nele executada a linha de comando apresentada na Figura 4.18. Devido a arquitetura do Netperf ser baseada no modelo cliente-servidor, foi necessário instalar a versão cliente desta ferramenta em outro computador. Assim sendo, optou-se em instalar-la no computador cliente do sistema de arquivos distribuído e executar a linha de comando apresentada na Figura 4.17.

Para garantir sua constante execução, a chamada do Netperf foi incorporada dentro de um laço de execução, conforme ilustrado na Figura 4.18.

```
netserver –p 5001
```

Figura 4.17 Chamada do Netperf no cliente dos sistemas de arquivos distribuídos
Fonte: Imagem do Autor

```
for i in $(seq 100); do netperf -H 192.168.206.13 -p 5001 -t TCP_RR
-- -r 20971520,20971520; done
```

Figura 4.18 Chamada do Netperf no computador responsável pela geração do tráfego
Fonte: Imagem do Autor

Enquanto o Netperf estava em execução, o IOzone foi invocado no computador cliente do sistema de arquivos distribuído fazendo-se uso da linha de comando apresentada na Figura 4.13. Os resultados obtidos com este teste são ilustrados na Figura 4.19.

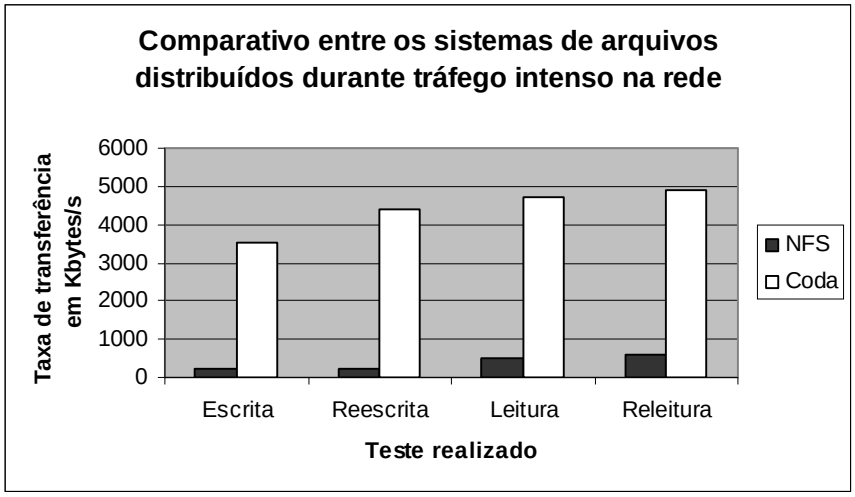

Figura 4.19 Comparativo de desempenho entre os sistemas de arquivos distribuídos durante o tráfego intenso na rede
Fonte: Imagem do Autor

A Figura 4.20 ilustra o percentual de perda de performance entre as ferramentas em relação as duas simulações.

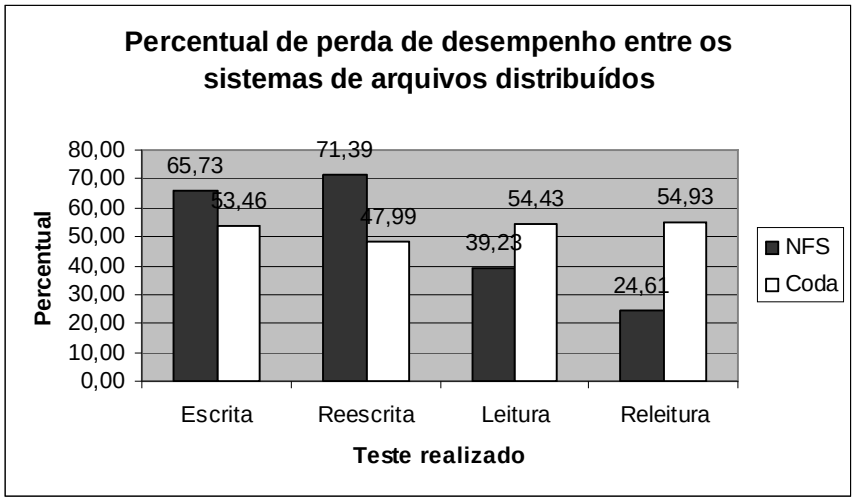

Figura 4.20 Percentual de perda de desempenho entre os sistemas de arquivos distribuídos
Fonte: Imagem do Autor

Percebe-se que o Coda File System obteve uma perda de performance inferior ao Network File System nos testes de escrita e reescrita, e uma perda superior nos testes de leitura e releitura. Acredita-se que esse comportamento é resultado da arquitetura de compartilhamento das ferramentas. Quando o usuário abre um determinado arquivo com sucesso no Coda File System, este é transferido integralmente ao computador cliente para que todas as operações sejam realizadas em *cache*, o que justifica o baixo percentual da perda de performance nos processos de escrita e reescrita em relação ao Network File System, e após seu fechamento, o arquivo é transferido ao servidor novamente, diferente do Network File System, onde todas as operações são realizadas diretamente no servidor (TANENBAUM, 2002).

4.7 Dificuldades Encontradas

Um dos objetivos deste trabalho era identificar o grau da transparência de replicação, transparência de acesso e transparência de localização existente em sistemas de arquivos distribuídos. Para o cumprimento deste objetivo foi necessário a elaboração de cenários em que, dependendo do resultado obtido com sua aplicabilidade, possibilitasse classificar o grau da transparência dos sistemas de arquivos distribuídos em alto ou baixo. Por não existirem pesquisas anteriores com este foco, diversos questionamentos e dúvidas foram levantados durante a elaboração destes cenários quanto a real eficiência deles.

Outro ponto que prejudicou o andamento do trabalho foi a carência de documentação ou a sua desatualização. O Network File System, em sua versão 3, é uma ferramenta amplamente difundida, presente em grande parte das distribuições do Sistema Operacional Linux e com uma vasta documentação em livros e artigos. Entretanto, a versão 3 não contempla o suporte a replicação de arquivos, sendo necessário para este trabalho fazer uso da versão 4, que ainda possui poucas referências de consulta. Por outro lado, o Coda File System possui uma vasta documentação do projeto em seu Site que infelizmente encontrasse desatualizada, tanto em termos de procedimentos para sua instalação quanto em termos de arquitetura e funcionalidades do sistema (HARKES, 2005; JOHNSON, 2007).

A escassez e a descontinuação de projetos desta área de pesquisa, a exemplo do InterMezzo (INTERMEZZO, 2007), também dificultou a escolha das ferramentas a serem estudadas, uma vez que um dos critérios para a seleção era o fato do projeto estar em desenvolvimento.

4.8 Trabalhos Futuros

Durante a realização desta monografia, foram observados vários aspectos que podem estender ou melhorar esse trabalho futuramente.

O primeiro diz respeito à elaboração de métricas que possam identificar o grau da transparência das demais técnicas não citadas neste trabalho, além de uma possível melhoria nas métricas aqui desenvolvidas. Outro aspecto refere-se a realização de testes de desempenho mais precisos e amplos, mensurando o uso de memória e de processamento que os sistemas de arquivos distribuídos consomem, tanto no cliente quanto no servidor, realizando um comparativo com os sistemas de arquivos locais. Como complemento para esta sugestão, a simulação de tráfego intenso na rede durante o processo de manipulação de arquivos grandes pode servir como artefato para identificar protocolos mal formados devido a possíveis problemas de especificação ou implementação.

Também fica como proposta para trabalhos futuros a realização dos experimentos acima em sistemas de arquivos distribuídos não apresentados neste trabalho.

Outras pesquisas, também na área de armazenamento distribuído, podem dar origem a novos trabalhos, como o estudo de Sistemas de Arquivos Paralelos (CARVALHO, 2003) e Data *Grids* (TAURION, 2004).

Todos esses levantamentos não fizeram parte deste trabalho e ficam como inspiração para trabalhos futuros. Alguns desses itens não foram abordados no trabalho para que este não ficasse demasiadamente extenso e outros devido ao desconhecimento do autor nesta área de pesquisa antes da realização deste trabalho.

CONSIDERAÇÕES FINAIS

O objetivo deste trabalho foi o estudo de conceitos, desafios e problemas gerados com a implementação de sistemas distribuídos e sistemas de arquivos distribuídos, abordando uma pesquisa por ferramentas que fornecessem este serviço.

Os grandes benefícios oferecidos pelo armazenamento distribuído vêm motivando e fomentando grupos de pesquisas em Universidades e empresas de grande porte, que investem no desenvolvimento de novas técnicas e melhorias para o aperfeiçoamento destas soluções. Diversas ferramentas experimentais foram desenvolvidas nos últimos anos, cada uma delas com uma finalidade específica e com muitas exigências contraditórias. Entretanto, apesar da existência de inúmeras soluções para proporcionar este serviço, diversos assuntos e desafios existentes na implementação de sistemas distribuídos permanecem problemáticos, como a exemplo da transparência.

Assim sendo, a segunda etapa deste trabalho teve como objetivo identificar o grau da transparência de replicação, transparência de acesso e transparência de localização existente nos sistemas de arquivos distribuídos, uma vez que a presença ou ausência destas técnicas afetam fortemente a utilização de recursos distribuídos. Para tanto, foram propostos cenários que tinham como objetivo identificar o grau da transparência, classificado em alto e baixo, destas técnicas. Nestes cenários, os clientes de uma rede usufruíram do serviço de armazenamento distribuído oferecido pelas ferramentas Coda File System e Network File System.

Como resultado desta experimentação, foi constatado que ambas as ferramentas possuíam um alto grau de transparência de acesso e que o Network File System possuía uma baixa transparência de localização e replicação, enquanto que o Coda File System possuía uma alta transparência de localização e replicação.

Além dos experimentos acima citados, foram realizados *benchmarks* para mensurar o desempenho dos sistemas de arquivos distribuídos nos processos de escrita, reescrita, leitura e releitura de arquivos, sendo que o resultado obtido foi reflexo de suas arquiteturas. Devido ao suporte a *cache* existente no cliente do Coda File System, o que possibilita a manipulação de arquivos no computador local, este obteve uma taxa de transferência superior ao Network File System, que realiza operações diretamente nos arquivos armazenados no servidor, fazendo uso constante da rede. Entretanto, os ensaios demonstraram uma perda de performance maior no Coda File System durante a leitura e releitura de arquivos em uma rede com excessivo tráfego, uma vez que, após o acesso ao arquivo, este é transferido integralmente ao computador local.

Ao final deste trabalho concluiu-se que os sistemas de arquivos distribuídos representam um avanço em relação ao armazenamento de dados. As ferramentas atualmente existentes não abrangem todo o espectro de um sistema gerenciador de arquivos distribuídos mencionado em bibliografias, principalmente devido a dificuldade em se resolver grandes desafios da área de sistemas distribuídos, entretanto, vários avanços foram feitos nos últimos anos, sendo possível encontrar sistemas de arquivos distribuídos sendo utilizados em ambientes de produção.

Esta é uma área de pesquisa bastante promissora, sendo seu potencial demonstrado no investimento realizado por empresas de renome, e as lacunas hoje existentes nas ferramentas de armazenamento distribuídos proporcionam a realização e continuação de diversos trabalhos nesta área.

REFERÊNCIAS BIBLIOGRÁFICAS

ADYA, Atul et. al. Farsite: federated, available, and reliable storage for an incompletely trusted environment. In: ACM SIGOPS OPERATING SYSTEMS REVIEW, 36., 2002, New York. **Anais...** New York: ACM Press, 2002. p. 1-14.

BECK, Micah; MOORE, Terry; PLANK, James S. An end-to-end approach to globally scalable programmable networking. In: ACM SIGCOMM WORKSHOP ON FUTURE DIRECTIONS IN NETWORK ARCHITECTURE, 2003, New York. **Anais...** New York: ACM Press, 2003, p. 328-339.

BEINSYNC. **BeInSync.** Disponível em: <http://www.beinsync.com/>. Acesso em: 11 Jun. 2007.

BRAAM, Peter J. The Coda Distributed File System. **Linux Journal**, Seattle, v. 1998, n. 50es, 6 p., Jun. 1998.

CARVALHO, Roberto Pires de**. Sistemas de Arquivos Paralelos e Distribuídos.** São Paulo: 2003. 75 p. Tese (Mestrado em Ciência da Computação) - Instituto de Matemática e Estatística, Universidade de São Paulo, 2003.

CHOW, Randy; JOHNSON, Theodore. **Distributed operating systems and algorithms.** Reading, Massachusetts: Addison Wesley Longman, 1998. 569 p.

CILIENDO, Eduardo; KUNIMASA, Takechika. **Linux Performance and Tuning Guidelines.** IBM Press, 2007. 170 p.

COULOURIS, George; DOLLIMORE, Jean; KINDBERG, Tim. **Distributed systems: concepts and design.** 4. ed. Harlow: Addison Wesley Longman, 2005. 927 p.

DABEK, Frank et al. Wide-area cooperative storage with CFS. In: ACM SYMPOSIUM ON OPERATING SYSTEMS PRINCIPLES, 18., 2001, New York. **Anais...** New York: ACM Press, 2001. p. 202-215.

FIELDS, Bruce J. Sanity check. **NFS Mailing Lists**, 14 Abr. 2006. Disponível em: <http://linux-nfs.org/pipermail/nfsv4/2006-April/004130.html>. Acesso em: 11 Jun. 2007.

FOLDERSHARE. **FolderShare.** Disponível em: <http://www.foldershare.com/>. Acesso em: 11 Jun. 2007.

GALLI, Doreen L. **Distributed Operating Systems: Concepts and Practice.** New Jersey: Prentice Hall, 2000. 463 p.

GHEMAWAT, Sanjay; GOBIOFF, Howard; LEUNG, Shun-Tak. The Google File System. In: ACM SYMPOSIUM ON OPERATING SYSTEMS PRINCIPLES, 19., 2003, New York. **Anais...** New York: ACM Press, 2003. p. 29-43.

HARKES, Jan. Re: Multiple coda servers. **Coda Mailing Lists**, 11 Set. 1999. Disponível em: <http://www.coda.cs.cmu.edu/maillists/codalist/codalist-1999/1749.html>. Acesso em: 11 Jun. 2007.

HARKES, Jan. Re: what a realm is exactly? **Coda Mailing Lists**, 28 Out. 2004. Disponível em: <http://www.coda.cs.cmu.edu/maillists/codalist/codalist-2004/6966.html>. Acesso em: 11 Jun. 2007.

HARKES, Jan. Re: Q about the VSGDB file and referring to realms. **Coda Mailing Lists**, 26 Set. 2005. Disponível em: <http://www.coda.cs.cmu.edu/maillists/codalist/codalist-2005/7748.html>. Acesso em: 11 Jun. 2007.

HARKES, Jan. Re: Iozone. **Coda Mailing Lists**, 29 Mai. 2007. Disponível em: <http://www.coda.cs.cmu.edu/maillists/codalist/codalist-2007/8548.html>. Acesso em: 11 Jun. 2007.

INTERMEZZO. **InterMezzo File System.** Disponível em <http://www.inter-mezzo.org/>. Acesso em: 11 Jun. 2007.

JALOTE, Pankaj. **Fault Tolerance in Distributed Systems.** New Jersey: Prentice Hall, 1994. 432 p.

JOHNSON, Alastair. Re: discrepancies in coda documentation. **Coda Mailing Lists**, 24 Mai. 2007. Disponível em: <http://coda.cs.cmu.edu/maillists/codalist/codalist-2007/8529.html> Acesso em: 11 Jun. 2007.

KIRNER, Claudio; MENDES, Sueli B. T. **Sistemas operacionais distribuídos: aspectos gerais e análise de sua estrutura.** Rio de Janeiro: Campus, 1988. 184 p.

KISTLER, James J.; SATYANARAYANAN M. Disconnected operation in the Coda file system. In: ACM SYMPOSIUM ON OPERATING SYSTEMS PRINCIPLES, 13., 1991, New York. **Anais...** New York: ACM Press, 1991, p. 213-225.

KON, Fabio**. Distributed File Systems Past, Present and Future.** 1996.

KON, Fabio. **Sistemas de Arquivos Distribuídos.** São Paulo: 1994. 154 p. Tese (Mestrado em Matemática Aplicada) - Instituto de Matemática e Estatística, Universidade de São Paulo, 1994.

MACHADO, Francis Berenger; MAIA, Luiz Paulo. **Arquitetura de sistemas operacionais.** 3. ed. Rio de Janeiro: LTC, 2002. 311 p.

MEDIAMAX. **MediaMax.** Disponível em: <http://www.mediamax.com/>. Acesso em: 11 Jun. 2007.

MULLENDER, Sape. **Distributed systems.** 2. ed. New York: ACM Press/Addison-Wesley Publishing Co., 1993. 595 p.

NOTKIN, David et al. Heterogeneous computing environments: report on the ACM SIGOPS workshop on accommodating heterogeneity. **Commun. ACM**, ACM Press, v. 30, n. 2, p. 132-140, Fev. 1987.

RIBEIRO, Uirá Endy. **Sistemas Distribuídos: Desenvolvendo Aplicações de Alta Performace no Linux.** Rio de Janeiro: Axcel Books do Brasil, 2005. 384 p.

ROUSH, Wade. The Internet Is Your Next Hard Drive**. Technology Review**, 24 Jul. 2006. Disponível em: <http://www.technologyreview.com/read_article.aspx?id=17195&ch=infotech>. Acesso em: 11 Jun. 2007.

ROWSTRON, Antony; DRUSCHEL, Peter. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In: ACM SYMPOSIUM ON OPERATING SYSTEMS PRINCIPLES, 18., 2001, New York. **Anais...** New York: ACM Press, 2001, p. 188-201.

SIEWIOREK, Daniel P. Architecture of Fault-Tolerant Computers. **Computer**, IEEE Computer Society, v. 17, n. 8, p. 9-18, Ago. 1984.

SILBERSCHATZ, Abraham; GALVIN, Peter B.; GAGNE, Greg. **Applied Operating System Concepts.** New York: John Wiley & Sons, 2000. 840 p.

SINGHAL, Mukesh; SHIVARATRI, Niranjan G. **Advanced concepts in operating systems: distributed, database, and multiprocessor operating systems.** Nova York: McGraw-Hill, 1994. 522 p.

SINHA, Pradeep K. **Distributed Operating Systems: Concepts and Design.** New York: Wiley-IEEE Press, 1996. 764 p.

SPECTOR, Alfred Zalmon. **Multiprocessing architectures for local computer networks.** Stanford: 1981. 127 p. Tese (Doutorado em Ciência da Computação) - Departamento de Ciência da Computação, Universidade de Stanford, 1981.

TANENBAUM, Andrew S. **Computer Networks.** New Jersey: Prentice Hall, 2003. 384 p.

TANENBAUM, Andrew S.; WOODHULL, Albert S. **Operating Systems Design and Implementation.** 3. ed. New Jersey: Prentice Hall, 2006. 1080 p.

TANENBAUM, Andrew S.; STEEN, Maarten van. **Distributed Systems: Principles and Paradigms.** New Jersey: Prentice Hall, 2002. 803 p.

TAURION, Cezar. **Grid computing: um novo paradigma computacional.** Rio de Janeiro: Brasport, 2004. 146 p.

TROXEL, Greg. Re: Adding space to a server. **Coda Mailing Lists**, 18 Mai. 2007. Disponível em: <http://www.coda.cs.cmu.edu/maillists/codalist/codalist-2007/8493.html>. Acesso em: 11 Jun. 2007.

VERISSIMO, Paulo; RODRIGUES, Luis. **Distributed Systems for System Architects.** Norwell: Kluwer Academic Publishers, 2001. 623 p.

WOHLIN, Claes et. al. **Experimentation in Software Engineering: An Introduction.** Norwell: Kluwer Academic Publishers, 2000. 204 p.

ZHANG, Jiaying; HONEYMAN, Peter**. Naming, Migration, and Replication in NFSv4.** 2003. Disponível em: <http://citeseer.ist.psu.edu/636612.html>. Acesso em: 11 Jun. 2007.

ANEXOS

ANEXO A

Resultado obtido com o IOzone após a realização dos testes de escrita, reescrita, leitura e releitura no sistema de arquivos Ext3 simulando a manipulação de arquivos no tamanho de 1 GB e blocos no tamanho de 4 KB.

```
Iozone: Performance Test of File I/O
        Version $Revision: 3.263 $
        Compiled for 32 bit mode.
        Build: linux

Contributors:William Norcott, Don Capps, Isom Crawford, Kirby Collins
             Al Slater, Scott Rhine, Mike Wisner, Ken Goss
             Steve Landherr, Brad Smith, Mark Kelly, Dr. Alain CYR,
             Randy Dunlap, Mark Montague, Dan Million,
             Jean-Marc Zucconi, Jeff Blomberg,
             Erik Habbinga, Kris Strecker, Walter Wong.

Run began: Wed May 30 15:29:29 2007

Include close in write timing
Record Size 4 KB
File size set to 1048576 KB
O_DIRECT feature enabled
Command line used: iozone -c -t 1 -F /tmp/ -i 0 -i 1 -r 4k -s 1g
Output is in Kbytes/sec
Time Resolution = 0.000001 seconds.
Processor cache size set to 1024 Kbytes.
Processor cache line size set to 32 bytes.
File stride size set to 17 * record size.
Throughput test with 1 process
Each process writes a 1048576 Kbyte file in 4 Kbyte records

Children see throughput for  1 initial writers      =   15031.65 KB/sec
Parent sees throughput for  1 initial writers       =   15031.31 KB/sec
Min throughput per process                          =   15031.65 KB/sec
Max throughput per process                          =   15031.65 KB/sec
Avg throughput per process                          =   15031.65 KB/sec
Min xfer                                            = 1048576.00 KB

Children see throughput for  1 rewriters            =   15420.41 KB/sec
Parent sees throughput for  1 rewriters             =   15419.55 KB/sec
Min throughput per process                          =   15420.41 KB/sec
Max throughput per process                          =   15420.41 KB/sec
Avg throughput per process                          =   15420.41 KB/sec
Min xfer                                            = 1048576.00 KB

Children see throughput for  1 readers              =   16820.97 KB/sec
Parent sees throughput for  1 readers               =   16819.94 KB/sec
Min throughput per process                          =   16820.97 KB/sec
Max throughput per process                          =   16820.97 KB/sec
Avg throughput per process                          =   16820.97 KB/sec
Min xfer                                            = 1048576.00 KB

Children see throughput for 1 re-readers            =   16810.84 KB/sec
Parent sees throughput for 1 re-readers             =   16809.80 KB/sec
Min throughput per process                          =   16810.84 KB/sec
Max throughput per process                          =   16810.84 KB/sec
Avg throughput per process                          =   16810.84 KB/sec
Min xfer                                            = 1048576.00 KB

iozone test complete.
```

ANEXO B

Resultado obtido com o IOzone após a realização dos testes de escrita, reescrita, leitura e releitura no Network File System simulando a manipulação de arquivos no tamanho de 1 GB e blocos no tamanho de 10 MB.

```
Iozone: Performance Test of File I/O
        Version $Revision: 3.263 $
        Compiled for 32 bit mode.
        Build: linux

Contributors:William Norcott, Don Capps, Isom Crawford, Kirby Collins
             Al Slater, Scott Rhine, Mike Wisner, Ken Goss
             Steve Landherr, Brad Smith, Mark Kelly, Dr. Alain CYR,
             Randy Dunlap, Mark Montague, Dan Million,
             Jean-Marc Zucconi, Jeff Blomberg,
             Erik Habbinga, Kris Strecker, Walter Wong.

Run began: Wed May 30 07:26:42 2007

Include close in write timing
Record Size 10240 KB
File size set to 1048576 KB
Command line used: iozone -c -t 1 -F /mnt/ -i 0 -i 1 -r 10m -s 1g
Output is in Kbytes/sec
Time Resolution = 0.000001 seconds.
Processor cache size set to 1024 Kbytes.
Processor cache line size set to 32 bytes.
File stride size set to 17 * record size.
Throughput test with 1 process
Each process writes a 1048576 Kbyte file in 10240 Kbyte records

Children see throughput for  1 initial writers      =     625.01 KB/sec
Parent sees throughput for  1 initial writers       =     623.43 KB/sec
Min throughput per process                          =     625.01 KB/sec
Max throughput per process                          =     625.01 KB/sec
Avg throughput per process                          =     625.01 KB/sec
Min xfer                                            = 1048576.00 KB

Children see throughput for  1 rewriters            =     763.43 KB/sec
Parent sees throughput for  1 rewriters             =     761.34 KB/sec
Min throughput per process                          =     763.43 KB/sec
Max throughput per process                          =     763.43 KB/sec
Avg throughput per process                          =     763.43 KB/sec
Min xfer                                            = 1048576.00 KB

Children see throughput for  1 readers              =     808.58 KB/sec
Parent sees throughput for  1 readers               =     806.99 KB/sec
Min throughput per process                          =     808.58 KB/sec
Max throughput per process                          =     808.58 KB/sec
Avg throughput per process                          =     808.58 KB/sec
Min xfer                                            = 1048576.00 KB

Children see throughput for 1 re-readers            =     800.26 KB/sec
Parent sees throughput for 1 re-readers             =     798.74 KB/sec
Min throughput per process                          =     800.26 KB/sec
Max throughput per process                          =     800.26 KB/sec
Avg throughput per process                          =     800.26 KB/sec
Min xfer                                            = 1048576.00 KB

iozone test complete.
```

ANEXO C

Resultado obtido com o IOzone após a realização dos testes de escrita, reescrita, leitura e releitura no Coda File System simulando a manipulação de arquivos no tamanho de 1 GB e blocos no tamanho de 10 MB.

```
Iozone: Performance Test of File I/O
        Version $Revision: 3.263 $
        Compiled for 32 bit mode.
        Build: linux

Contributors:William Norcott, Don Capps, Isom Crawford, Kirby Collins
             Al Slater, Scott Rhine, Mike Wisner, Ken Goss
             Steve Landherr, Brad Smith, Mark Kelly, Dr. Alain CYR,
             Randy Dunlap, Mark Montague, Dan Million,
             Jean-Marc Zucconi, Jeff Blomberg,
             Erik Habbinga, Kris Strecker, Walter Wong.

Run began: Wed May 30 18:36:30 2007

Include close in write timing
Record Size 10240 KB
File size set to 1048576 KB
Command line used: iozone -c -t 1 -F /coda/codaserver/ -i 0 -i 1 -r 10m -s 1g
Output is in Kbytes/sec
Time Resolution = 0.000001 seconds.
Processor cache size set to 1024 Kbytes.
Processor cache line size set to 32 bytes.
File stride size set to 17 * record size.
Throughput test with 1 process
Each process writes a 1048576 Kbyte file in 10240 Kbyte records

Children see throughput for  1 initial writers      =    7589.45 KB/sec
Parent sees throughput for  1 initial writers       =    6973.74 KB/sec
Min throughput per process                          =    7589.45 KB/sec
Max throughput per process                          =    7589.45 KB/sec
Avg throughput per process                          =    7589.45 KB/sec
Min xfer                                            = 1048576.00 KB

Children see throughput for  1 rewriters            =    8451.69 KB/sec
Parent sees throughput for  1 rewriters             =    7498.81 KB/sec
Min throughput per process                          =    8451.69 KB/sec
Max throughput per process                          =    8451.69 KB/sec
Avg throughput per process                          =    8451.69 KB/sec
Min xfer                                            = 1048576.00 KB

Children see throughput for  1 readers              =   10358.03 KB/sec
Parent sees throughput for  1 readers               =    9797.49 KB/sec
Min throughput per process                          =   10358.03 KB/sec
Max throughput per process                          =   10358.03 KB/sec
Avg throughput per process                          =   10358.03 KB/sec
Min xfer                                            = 1048576.00 KB

Children see throughput for 1 re-readers            =   10826.63 KB/sec
Parent sees throughput for 1 re-readers             =   10651.36 KB/sec
Min throughput per process                          =   10826.63 KB/sec
Max throughput per process                          =   10826.63 KB/sec
Avg throughput per process                          =   10826.63 KB/sec
Min xfer                                            = 1048576.00 KB

iozone test complete.
```

ANEXO D

Resultado obtido com o IOzone após a realização dos testes de escrita, reescrita, leitura e releitura no sistema de arquivos Ext3 simulando a manipulação de arquivos no tamanho de 1 GB e blocos no tamanho de 10 MB.

```
Iozone: Performance Test of File I/O
        Version $Revision: 3.263 $
        Compiled for 32 bit mode.
        Build: linux

Contributors:William Norcott, Don Capps, Isom Crawford, Kirby Collins
             Al Slater, Scott Rhine, Mike Wisner, Ken Goss
             Steve Landherr, Brad Smith, Mark Kelly, Dr. Alain CYR,
             Randy Dunlap, Mark Montague, Dan Million,
             Jean-Marc Zucconi, Jeff Blomberg,
             Erik Habbinga, Kris Strecker, Walter Wong.

Run began: Sun Jun  3 15:34:39 2007

Include close in write timing
Record Size 10240 KB
File size set to 1048576 KB
Command line used: iozone -c -t 1 -F /tmp/ -i 0 -i 1 -r 10m -s 1g
Output is in Kbytes/sec
Time Resolution = 0.000001 seconds.
Processor cache size set to 1024 Kbytes.
Processor cache line size set to 32 bytes.
File stride size set to 17 * record size.
Throughput test with 1 process
Each process writes a 1048576 Kbyte file in 10240 Kbyte records

Children see throughput for  1 initial writers      =   15143.53 KB/sec
Parent sees throughput for  1 initial writers       =   15142.85 KB/sec
Min throughput per process                          =   15143.53 KB/sec
Max throughput per process                          =   15143.53 KB/sec
Avg throughput per process                          =   15143.53 KB/sec
Min xfer                                            = 1048576.00 KB

Children see throughput for  1 rewriters            =   15311.27 KB/sec
Parent sees throughput for  1 rewriters             =   15311.11 KB/sec
Min throughput per process                          =   15311.27 KB/sec
Max throughput per process                          =   15311.27 KB/sec
Avg throughput per process                          =   15311.27 KB/sec
Min xfer                                            = 1048576.00 KB

Children see throughput for  1 readers              =   16706.31 KB/sec
Parent sees throughput for  1 readers               =   16705.65 KB/sec
Min throughput per process                          =   16706.31 KB/sec
Max throughput per process                          =   16706.31 KB/sec
Avg throughput per process                          =   16706.31 KB/sec
Min xfer                                            = 1048576.00 KB

Children see throughput for 1 re-readers            =   16700.94 KB/sec
Parent sees throughput for 1 re-readers             =   16700.28 KB/sec
Min throughput per process                          =   16700.94 KB/sec
Max throughput per process                          =   16700.94 KB/sec
Avg throughput per process                          =   16700.94 KB/sec
Min xfer                                            = 1048576.00 KB

iozone test complete.
```

ANEXO E

Resultado obtido com o IOzone após a realização dos testes de escrita, reescrita, leitura e releitura no Coda File System simulando a manipulação de arquivos no tamanho de 1 GB e blocos no tamanho de 10 MB com excessivo tráfego na rede.

```
Iozone: Performance Test of File I/O
        Version $Revision: 3.263 $
        Compiled for 32 bit mode.
        Build: linux

Contributors:William Norcott, Don Capps, Isom Crawford, Kirby Collins
             Al Slater, Scott Rhine, Mike Wisner, Ken Goss
             Steve Landherr, Brad Smith, Mark Kelly, Dr. Alain CYR,
             Randy Dunlap, Mark Montague, Dan Million,
             Jean-Marc Zucconi, Jeff Blomberg,
             Erik Habbinga, Kris Strecker, Walter Wong.

Run began: Wed May 31 17:32:12 2007

Include close in write timing
Record Size 10240 KB
File size set to 1048576 KB
Command line used: iozone -c -t 1 -F /coda/codaserver/ -i 0 -i 1 -r 10m -s 1g
Output is in Kbytes/sec
Time Resolution = 0.000001 seconds.
Processor cache size set to 1024 Kbytes.
Processor cache line size set to 32 bytes.
File stride size set to 17 * record size.
Throughput test with 1 process
Each process writes a 1048576 Kbyte file in 10240 Kbyte records

Children see throughput for  1 initial writers      =    3532.49 KB/sec
Parent sees throughput for  1 initial writers       =    3389.86 KB/sec
Min throughput per process                          =    3532.49 KB/sec
Max throughput per process                          =    3532.49 KB/sec
Avg throughput per process                          =    3532.49 KB/sec
Min xfer                                            = 1048576.00 KB

Children see throughput for  1 rewriters            =    4395.91 KB/sec
Parent sees throughput for  1 rewriters             =    4217.34 KB/sec
Min throughput per process                          =    4395.91 KB/sec
Max throughput per process                          =    4395.91 KB/sec
Avg throughput per process                          =    4395.91 KB/sec
Min xfer                                            = 1048576.00 KB

Children see throughput for  1 readers              =    4720.49 KB/sec
Parent sees throughput for  1 readers               =    4495.18 KB/sec
Min throughput per process                          =    4720.49 KB/sec
Max throughput per process                          =    4720.49 KB/sec
Avg throughput per process                          =    4720.49 KB/sec
Min xfer                                            = 1048576.00 KB

Children see throughput for 1 re-readers            =    4879.97 KB/sec
Parent sees throughput for 1 re-readers             =    4697.01 KB/sec
Min throughput per process                          =    4879.97 KB/sec
Max throughput per process                          =    4879.97 KB/sec
Avg throughput per process                          =    4879.97 KB/sec
Min xfer                                            = 1048576.00 KB

iozone test complete.
```

ANEXO F

Resultado obtido com o IOzone após a realização dos testes de escrita, reescrita, leitura e releitura no Network File System simulando a manipulação de arquivos no tamanho de 1 GB e blocos no tamanho de 10 MB com excessivo tráfego na rede.

```
Iozone: Performance Test of File I/O
        Version $Revision: 3.263 $
        Compiled for 32 bit mode.
        Build: linux

Contributors:William Norcott, Don Capps, Isom Crawford, Kirby Collins
             Al Slater, Scott Rhine, Mike Wisner, Ken Goss
             Steve Landherr, Brad Smith, Mark Kelly, Dr. Alain CYR,
             Randy Dunlap, Mark Montague, Dan Million,
             Jean-Marc Zucconi, Jeff Blomberg,
             Erik Habbinga, Kris Strecker, Walter Wong.

Run began: Wed May 31 07:52:33 2007

Include close in write timing
Record Size 10240 KB
File size set to 1048576 KB
Command line used: iozone -c -t 1 -F /mnt/ -i 0 -i 1 -r 10m -s 1g
Output is in Kbytes/sec
Time Resolution = 0.000001 seconds.
Processor cache size set to 1024 Kbytes.
Processor cache line size set to 32 bytes.
File stride size set to 17 * record size.
Throughput test with 1 process
Each process writes a 1048576 Kbyte file in 10240 Kbyte records

Children see throughput for  1 initial writers      =     214.21 KB/sec
Parent sees throughput for  1 initial writers       =     213.88 KB/sec
Min throughput per process                          =     214.21 KB/sec
Max throughput per process                          =     214.21 KB/sec
Avg throughput per process                          =     214.21 KB/sec
Min xfer                                            = 1048576.00 KB

Children see throughput for  1 rewriters            =     218.41 KB/sec
Parent sees throughput for  1 rewriters             =     217.92 KB/sec
Min throughput per process                          =     218.41 KB/sec
Max throughput per process                          =     218.41 KB/sec
Avg throughput per process                          =     218.41 KB/sec
Min xfer                                            = 1048576.00 KB

Children see throughput for  1 readers              =     491.41 KB/sec
Parent sees throughput for  1 readers               =     489.71 KB/sec
Min throughput per process                          =     491.41 KB/sec
Max throughput per process                          =     491.41 KB/sec
Avg throughput per process                          =     491.41 KB/sec
Min xfer                                            = 1048576.00 KB

Children see throughput for 1 re-readers            =     603.30 KB/sec
Parent sees throughput for 1 re-readers             =     587.73 KB/sec
Min throughput per process                          =     603.30 KB/sec
Max throughput per process                          =     603.30 KB/sec
Avg throughput per process                          =     603.30 KB/sec
Min xfer                                            = 1048576.00 KB

iozone test complete.
```