

**Sistemas de Arquivos Paralelos  
Alternativas para a redução do gargalo  
no acesso ao sistema de arquivos**

Roberto Pires de Carvalho

DISSERTAÇÃO APRESENTADA  
AO  
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA  
DA  
UNIVERSIDADE DE SÃO PAULO  
PARA  
OBTENÇÃO DO GRAU DE MESTRE  
EM  
CIÊNCIA DA COMPUTAÇÃO

Área de Concentração: Ciência da Computação  
Orientador: Prof. Dr. Alfredo Goldman vel Lejbman

São Paulo  
2005

**Sistemas de Arquivos Paralelos  
Alternativas para a redução do gargalo  
no acesso ao sistema de arquivos**

Este exemplar corresponde à redação  
final da dissertação devidamente corrigida  
e defendida por Roberto Pires de Carvalho  
e aprovada pela comissão julgadora.

São Paulo, 23 de setembro de 2005.

Banca examinadora:

- Prof. Dr. Alfredo Goldman vel Lejbman (orientador) (IME-USP)
- Prof. Dr. Arnaldo Mandel (IME-USP)
- Prof. Dr. Marcos José Santana (ICMC-USP)

# Agradecimentos

Agradeço ao meu orientador Prof. Dr. Alfredo Goldman pelo apoio, paciência e insistência, sem os quais não teríamos atingido tal qualidade em nosso estudo. Aos meus colegas Rodrigo, Alex e Ala Oeste em geral, por oferecerem apoio e suporte sempre que precisei. Ao Prof. Dr. Fábio Kon, cujo trabalho de mestrado me ajudou muito como ponto de partida. Ao Prof. Dr. Roberto Hirata e aos administradores da rede IME, que me ajudaram e me suportaram durante a realização dos testes de desempenho.

E agradeço especialmente à minha esposa, aos meus pais, às minhas irmãs e aos meus avós, por sempre me incentivarem a continuar estudando e a nunca desistir, mesmo nos momentos mais difíceis pelos quais passei.



# Resumo

Nos últimos anos, a evolução dos processadores e redes para computadores de baixo custo foi muito maior se comparada com o aumento do desempenho dos discos de armazenamento de dados. Com isso, muitas aplicações estão encontrando dificuldades em atingir o pleno uso dos processadores, pois estes têm de esperar até que os dados cheguem para serem utilizados.

Uma forma popular para resolver esse tipo de empecílio é a adoção de sistemas de arquivos paralelos, que utilizam a velocidade da rede local, além dos recursos de cada máquina, para suprir a deficiência de desempenho no uso isolado de cada disco.

Neste estudo, analisamos alguns sistemas de arquivos paralelos e distribuídos, detalhando aqueles mais interessantes e importantes. Por fim, mostramos que o uso de um sistema de arquivos paralelo pode ser mais eficiente e vantajoso que o uso de um sistema de arquivos usual, para apenas um cliente.



# Abstract

In the last years, the evolution of the data processing power and network transmission for low cost computers was much bigger if compared to the increase of the speed of getting the data stored in disks. Therefore, many applications are finding difficulties in reaching the full use of the processors, because they have to wait until the data arrive before using.

A popular way to solve this problem is to use a parallel file system, which uses the local network speed to avoid the performance bottleneck found in an isolated disk.

In this study, we analyze some parallel and distributed file systems, detailing the most interesting and important ones. Finally, we show the use of a parallel file system can be more efficient than the use of a usual local file system, for just one client.



# Sumário

<b>1</b>	<b>Introdução</b>	<b>1</b>
<b>2</b>	<b>Conceitos e Definições</b>	<b>3</b>
2.1	Conceitos Básicos	3
2.1.1	Nomes e Localização	3
2.1.2	Cache	3
2.1.3	Transparências para o Usuário	4
2.2	Serviços Oferecidos pelos SADs	5
2.2.1	Serviço de Nomes Distribuído	5
2.2.2	Serviço de Arquivos Distribuído	6
2.2.3	Serviço de Diretórios Distribuído	6
2.3	Algumas Características Desejadas em SADs	7
2.3.1	Disponibilidade	7
2.3.2	Escalabilidade	8
2.3.3	Segurança	8
2.3.4	Tolerância a Falhas	9
2.3.5	Operações Atômicas	10
2.3.6	Acesso Concorrente	11
2.3.7	Replicação de Arquivos	12
2.4	Conclusões Sobre o Estudo Realizado	13
<b>3</b>	<b>Sistemas de Arquivos Paralelos e Distribuídos</b>	<b>15</b>
3.1	Alguns Exemplos de Sistemas de Arquivos Distribuídos	15
3.1.1	Os Primeiros SADs	15
3.1.2	NFS	16
3.1.3	AFS	18
3.1.4	CODA	21
3.1.5	SPRITE	24
3.2	Alguns Exemplos de Sistemas de Arquivos Paralelos	27
3.2.1	BRIDGE	27
3.2.2	PVFS	29
3.2.3	PVFS2	32

3.2.4	NFSP	36
3.2.5	CEFT-PVFS	37
3.2.6	GFS	38
3.2.7	SVA	43
3.3	Análise Comparativa	44
<b>4</b>	<b>Análise de Desempenho</b>	<b>45</b>
4.1	PVFS	45
4.2	NFSP	50
4.3	CEFT-PVFS	53
4.4	Análise Conclusiva	55
<b>5</b>	<b>PVFS2 vs. Ext3</b>	<b>57</b>
5.1	Ambiente Utilizado	57
5.1.1	PVFS2	57
5.1.2	Ext3	58
5.2	Sobre o Processo de Testes	59
5.2.1	Geração dos Arquivos para Teste de Leitura	59
5.2.2	Leitura e Escrita Paralela e Concorrente	60
5.2.3	Leitura Seqüencial vs. Aleatória	60
5.2.4	Um Arquivo Para Leitura e Vários Para Escrita	61
5.2.5	Garantindo Que os Dados Vêm do Disco	61
5.2.6	Aumentando a Largura da Banda da Rede	62
5.2.7	Scripts Para Testes Automatizados	63
5.2.8	Processamento dos Resultados	63
5.3	Análise dos Resultados de Leitura	63
5.3.1	Variação do Tamanho do Bloco de Leitura	64
5.3.2	Variação do Tamanho do Arquivo	64
5.3.3	Variação do Número de Servidores	69
5.3.4	Variação do Grau de Paralelismo	71
5.3.5	Apenas Um Processo, Sem Acesso Concorrente	71
5.3.6	Variação na Velocidade dos Discos	72
5.4	Análise dos Resultados de Escrita	72
5.4.1	Variação do Tamanho do Bloco de Escrita	74
5.4.2	Variação do Tamanho do Arquivo	74
5.4.3	Variação do Número de Servidores	76
5.4.4	Variação do Grau de Paralelismo	79
5.4.5	Apenas Um Processo, Sem Acesso Concorrente	79
5.4.6	Variação na Velocidade dos Discos	80
5.5	Conclusões Gerais	80
5.5.1	Rede Rápida É Mais Importante Que Discos Rápidos	82
5.5.2	Aproveitamento da Banda dos Discos Não É Linear	82
5.5.3	PVFS2 É Mais Rápido Que Ext3 Em Uma Rede Ethernet 1000Base-T	87

---

<b>6 Conclusão</b>	<b>93</b>
6.1 Principais Contribuições . . . . .	94
6.2 Trabalhos Futuros . . . . .	95
<b>A PSplit: Parallel Split</b>	<b>97</b>
A.1 Parâmetros da Linha de Comando . . . . .	97
A.2 O Processamento Paralelo . . . . .	98
A.3 Interpretando os Resultados Apresentados . . . . .	99
A.4 Exemplos de Uso . . . . .	100
<b>B Introdução aos Discos Magnéticos</b>	<b>101</b>
B.1 Fatores Para Análise de Desempenho . . . . .	101
B.2 Interfaces Entre o Disco e o Computador . . . . .	103
B.3 Latência . . . . .	104
B.4 Evolução dos Discos vs. Processadores e Rede . . . . .	106
<b>Referências Bibliográficas</b>	<b>107</b>



# Lista de Figuras

2.1	Exemplo de uma árvore de diretórios . . . . .	3
2.2	Cache no cliente e no servidor . . . . .	4
2.3	Estrutura de diretórios distribuída . . . . .	7
2.4	Problema na escrita concorrente . . . . .	11
2.5	Agglomerado de servidores de arquivos, com visão transparente para os clientes . . . . .	12
3.1	Volumes, VSGs e AVSGs . . . . .	22
3.2	Migração de processos . . . . .	24
3.3	Árvores de diretórios e tabela de prefixos no SPRITE . . . . .	25
3.4	(a) Estrutura <i>Interleaved File</i> e (b) visão geral do sistema BRIDGE . . . . .	28
3.5	Visão geral do sistema PVFS . . . . .	30
3.6	Clientes acessando o PVFS . . . . .	31
3.7	Fluxo de dados pelo <i>kernel</i> . . . . .	31
4.1	PVFS: banda passante total na leitura . . . . .	47
4.2	PVFS: banda passante total na escrita . . . . .	48
4.3	PVFS: banda passante total para volume de dados V- . . . . .	48
4.4	PVFS: banda passante total para volume de dados V+ . . . . .	49
4.5	NFSP: banda passante total na leitura . . . . .	50
4.6	NFSP: banda passante total na escrita . . . . .	51
4.7	NFSP: banda passante total para volume de dados V- . . . . .	51
4.8	NFSP: banda passante total para volume de dados V+ . . . . .	52
4.9	Desempenho de leitura do CEFT-PVFS ao variarmos o número de clientes . . . . .	53
4.10	Desempenho de leitura do CEFT-PVFS ao variarmos a quantidade de dados lidos . . . . .	54
4.11	Desempenho do CEFT-PVFS desviando de servidor com disco sobrecarregado . . . . .	55
4.12	Desempenho do CEFT-PVFS desviando de servidor com rede sobrecarregada . . . . .	56
5.1	Sistemas de arquivos paralelos vs. locais . . . . .	58
5.2	Médias das velocidades de leitura de cada um dos discos, nos vários modos PIO . . . . .	62
5.3	Desempenho do Ext3 para os vários tamanhos do bloco de leitura, lendo 1GB de dados em modo PIO 0 . . . . .	64

5.4	Desempenho do PVFS2 com (a) 1, (b) 2, (c) 4 e (d) 8 nós de dados, variando tamanho do bloco de leitura, considerando 1GB de dados lidos em modo PIO 0 . . . . .	65
5.5	Desempenho do Ext3 variando a quantidade de dados lidos, em blocos de 64KB, modo PIO 0 . . . . .	66
5.6	Desempenho variando a quantidade de dados lidos, para bloco de leitura de 64KB, usando sistema de arquivos PVFS2 com (a) 1, (b) 2, (c) 4 e (d) 8 nós de dados, modo PIO 0 . . . . .	67
5.7	Desempenho variando a quantidade de dados lidos, para blocos de dados de 64KB, usando (a) 1, (b) 2, (c) 4, (d) 8, (e) 16 e (f) 32 threads, modo PIO 0 . . . . .	68
5.8	Desempenho (a) e ganho percentual sobre o Ext3 (b) ao aumentar concorrência no acesso, lendo 1GB de dados, blocos de dados de 64KB, modo PIO 0 . . . . .	69
5.9	Desempenho (a) e ganho percentual sobre o Ext3 (b) ao aumentar concorrência no acesso, lendo 1GB de dados, bloco de leitura de 64KB, modo PIO 1 . . . . .	72
5.10	Desempenho (a) e ganho percentual sobre o Ext3 (b) ao aumentar concorrência no acesso, lendo 1GB de dados, bloco de leitura de 64KB, modo PIO 2 . . . . .	73
5.11	Desempenho (a) e ganho percentual sobre o Ext3 (b) ao aumentar concorrência no acesso, lendo 1GB de dados, bloco de leitura de 64KB, modo PIO 3 . . . . .	73
5.12	Desempenho do Ext3 variando o tamanho do bloco de escrita, para 1GB de dados, modo PIO 0 . . . . .	74
5.13	Desempenho do PVFS2 usando (a) 1, (b) 2, (c) 4 e (d) 8 nós de dados, variando o tamanho do bloco de escrita, considerando 1GB de dados gravados, modo PIO 0 . . . . .	75
5.14	Desempenho variando a quantidade de dados gravados, para bloco de leitura de 64KB, sistema de arquivos Ext3, modo PIO 0 . . . . .	76
5.15	Desempenho variando a quantidade de dados gravados, para bloco de escrita de 64KB, usando sistema de arquivos PVFS2 com (a) 1, (b) 2, (c) 4 e (d) 8 nós de dados, modo PIO 0 . . . . .	77
5.16	Desempenho variando a quantidade de dados gravados, para bloco de escrita de 64KB, usando (a) 1, (b) 2, (c) 4, (d) 8, (e) 16 e (f) 32 threads, modo PIO 0 . . . . .	78
5.17	Desempenho (a) e ganho percentual sobre o Ext3 (b) ao aumentar concorrência no acesso, escrevendo 1GB de dados, blocos de 64KB, modo PIO 0 . . . . .	79
5.18	Desempenho (a) e ganho percentual sobre o Ext3 (b) ao aumentar concorrência no acesso, escrevendo 1GB de dados, blocos de 64KB, modo PIO 1 . . . . .	80
5.19	Desempenho (a) e ganho percentual sobre o Ext3 (b) ao aumentar concorrência no acesso, escrevendo 1GB de dados, blocos de 64KB, modo PIO 2 . . . . .	81
5.20	Desempenho (a) e ganho percentual sobre o Ext3 (b) ao aumentar concorrência no acesso, escrevendo 1GB de dados, blocos de 64KB, modo PIO 3 . . . . .	81
5.21	Ganho percentual do PVFS2 sobre o Ext3 para os vários modos PIO, lendo 1GB de dados, blocos de 64KB, usando (a) 1, (b) 2, (c) 4, (d) 8, (e) 16 e (f) 32 threads . . . . .	83
5.22	Velocidade do PVFS2 e do Ext3 para os vários modos PIO, lendo 1GB de dados, blocos de 64KB, usando (a) 1, (b) 2, (c) 4, (d) 8, (e) 16 e (f) 32 threads . . . . .	84
5.23	Ganho percentual do PVFS2 sobre o Ext3 para os vários modos PIO, escrevendo 1GB de dados, blocos de 64KB, usando (a) 1, (b) 2, (c) 4, (d) 8, (e) 16 e (f) 32 threads . . . . .	85

---

5.24	Velocidade do PVFS2 e do Ext3 para os vários modos PIO, escrevendo 1GB de dados, blocos de 64KB, usando (a) 1, (b) 2, (c) 4, (d) 8, (e) 16 e (f) 32 threads . . .	86
5.25	Aproveitamento que o Ext3 e o PVFS2 tiveram, relativo à velocidade média agregada disponibilizada pelos discos dos servidores, para os vários modos PIO, lendo 1GB de dados, blocos de 64KB, usando (a) 1, (b) 2, (c) 4, (d) 8, (e) 16 e (f) 32 threads . . .	88
5.26	Aproveitamento que o Ext3 e o PVFS2 tiveram, relativo à velocidade média agregada disponibilizada pelos discos dos servidores, para os vários modos PIO, escrevendo 1GB de dados, blocos de 64KB, usando (a) 1, (b) 2, (c) 4, (d) 8, (e) 16 e (f) 32 threads	89
5.27	Aproveitamento dos sistemas de arquivos relativo à velocidade média agregada disponibilizada pelos discos dos servidores, para os modos PIO (a) 0, (b) 1, (c) 2 e (d) 3, lendo 1GB de dados, blocos de 64KB . . . . .	90
5.28	Aproveitamento dos sistemas de arquivos relativo à velocidade média agregada disponibilizada pelos discos dos servidores, para os modos PIO (a) 0, (b) 1, (c) 2 e (d) 3, escrevendo 1GB de dados, blocos de 64KB . . . . .	91
B.1	Organização lógica de um disco rígido . . . . .	101
B.2	Evolução tecnológica normalizada . . . . .	107



# Lista de Tabelas

B.1 Comparativo da evolução do disco rígido e do processamento de dados . . . . .	106
---	-----



# Capítulo 1

## Introdução

O **sistema de arquivos** é uma parte importantíssima do sistema operacional, pois fornece uma visão abstrata dos dados persistidos (também chamado de armazenamento secundário), além de ser o responsável pelo serviço de nomes, acesso a arquivos e de sua organização geral.

Um **Sistema de Arquivos Distribuído** (SAD) tem o objetivo de fornecer os mesmos serviços e recursos que um sistema de arquivos convencional, pelo menos na visão dos clientes que o acessam, sendo que um SAD pode ser acessado por qualquer máquina dentro de uma rede, de forma controlada ou não, além de não ser centralizado.

Seja local ou remoto, muitas aplicações precisam acessar o sistema de arquivos para obter os dados a serem processados, ou então para persistir os resultados obtidos. Tais arquivos são armazenados através de meios físicos, como dispositivos magnéticos e óticos, cuja desempenho no acesso não evolue na mesma proporção que a velocidade dos processadores ou da transmissão de dados (veja tabela B.1 no Apêndice B), seja internamente à máquina (como no acesso à memória, registradores, etc) ou externamente (como no acesso às outras máquinas via rede).

Isso torna o acesso aos arquivos um gargalo<sup>1</sup> para praticamente todas as aplicações, especialmente para aquelas que buscam informações armazenadas remotamente, que além do tempo de espera dos dados vindo pela rede, há também o tempo de espera pelo meio físico de armazenamento, que pode ser acentuado pelo acesso concorrente causado por outros pedidos simultâneos.

Assim, para melhorar o desempenho no acesso aos dados armazenados remotamente, surgiram os **Sistemas de Arquivos Paralelos** (SAP), que são sistemas de arquivos distribuídos especializados em fornecer alto desempenho no acesso concorrente. Porém, muitos deles se preocupam apenas com desempenho, deixando de lado a qualidade de serviço encontrada em muitos outros SADs, como a segurança no acesso ou até mesmo a alta disponibilidade (isto é, manter o serviço funcionando em casos de quebra de parte do sistema, também chamado de *fail over*).

Devido às vantagens de se usar SADs mais velozes e ao mesmo tempo confiáveis, vários projetos foram iniciados e muitos produtos foram criados. A pesquisa nessa área envolve não só grandes empresas, como a IBM (com o GPFS [JKY00]) e a Sun (com o NFS [SRO96, Kon]), mas também vários institutos de pesquisa acadêmica, como a Universidade de Minnesota (com o GFS [SRO96]), a Uni-

---

<sup>1</sup>Para informações sobre por que o acesso a dados físicos de um disco magnético é considerado um gargalo nos sistemas de arquivos, veja o Apêndice B.

versidade da California (com o RAMA – *Rapid Access to Massive Archive* [MK97, MK93]), a Universidade Nacional de Chiao Tung (com o Pasda – *Parallel Storage Devices Accelerator* [JFC94]) e a Universidade de Clemson (com o PVFS [CIRT00, Had00] e mais recentemente o PVFS2 [Ram02]).

Infelizmente, somente algumas dessas soluções foram desenvolvidas para fornecer todas as qualidades de serviço desejáveis em um sistema de arquivos, dificultando, assim, sua operação por usuários finais. Além disso, muitos pesquisadores, desenvolvedores e usuários não conhecem os benefícios que cada SAD pode proporcionar e acabam adotando um que não seja o mais indicado para resolver seus problemas ou satisfazer suas expectativas e necessidades.

Desse modo, surge a necessidade de se avaliar as soluções hoje existentes, comparando os problemas que vieram resolver, suas qualidades, utilidades, serviços oferecidos e deficiências, através de uma resenha, encontrada no capítulo 3. Nela descrevemos alguns dos sistemas de arquivos mais conhecidos, além de outros promissores, dando ênfase ao PVFS [CIRT00], um sistema de arquivos paralelo de código-fonte aberto, e aos seus variantes, como o NFSP [LD01], cuja intenção é tornar o acesso ao PVFS transparente para clientes NFS, e o CEFT-PVFS [ZJQ<sup>+</sup>03], que é uma melhoria no PVFS para proporcionar tolerância a falhas. No capítulo 4 realizamos um estudo do comportamento destes sistemas de arquivos através da análise de resultados experimentais, provenientes dos trabalhos [Lob03, ZJQ<sup>+</sup>03].

Por fim, nossa principal contribuição para este trabalho está no capítulo 5, onde descrevemos e analisamos um experimento com o PVFS2, que é acessado por apenas um cliente para leitura de dados através de múltiplas threads. Tal experimento é comparado, sob as mesmas condições, com o acesso ao sistema de arquivos local Ext3, a fim de mostrarmos que existem situações e condições em que o desempenho do PVFS2 é melhor até mesmo do que o desempenho do próprio sistema de arquivos local.

## Capítulo 2

# Conceitos e Definições

Neste capítulo encontram-se alguns conceitos e serviços disponibilizados pelos sistemas de arquivos distribuídos e paralelos, assim como algumas de suas características, qualidades e soluções [Gal00, Tan92] que os pesquisadores e desenvolvedores da área tentam prover.

### 2.1 Conceitos Básicos

#### 2.1.1 Nomes e Localização

A maioria dos arquivos armazenados em um sistema de arquivos possui um nome e um caminho, que o identifica unicamente em tal sistema. Um caminho representa um nó de uma estrutura de diretórios, que pode ser representada como uma árvore (veja fig. 2.1). Tal árvore possui somente uma raiz. Cada nó pode possuir árvores ou arquivos.

Dessa forma, para localizar um arquivo em uma árvore de diretórios (usados para agrupar arquivos) basta seguir o caminho do arquivo e, ao chegar no diretório final, procurar pelo nome de tal arquivo. A forma como esse nome e esse caminho são definidos depende muito do sistema operacional. Por exemplo, no Unix um caminho é definido como uma seqüência de nomes de diretórios, todos separados pelo caractere '/'. O último nome dessa seqüência pode ser o nome do arquivo ou de um diretório.

Em sistemas distribuídos, é possível encontrar o nome da máquina em que o arquivo se encontra dentro dessa definição de caminho. Porém procura-se deixar isso transparente para o usuário.

#### 2.1.2 Cache

Para melhorar o desempenho no acesso aos arquivos de um sistema, procura-se guardar informações muito acessadas em memória, para evitar a sobrecarga de se ter que obtê-las novamente do meio físico onde estão armazenadas.

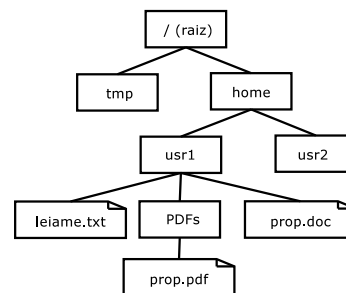


Figura 2.1: Exemplo de uma árvore de diretórios

Isso ajuda muito na economia de tempo de processamento pois para acessar dados remotos, por exemplo, o sistema está limitado à velocidade da rede, que, mesmo rápida, estará limitada à velocidade do meio físico de armazenamento do servidor remoto, pois este ainda precisaria procurar os dados, carregá-los na memória e enviá-los para o cliente.

Mesmo no acesso a dados locais, a velocidade de acesso à memória é muito maior que a velocidade de acesso ao meio de armazenamento, por exemplo, um disco rígido, que precisaria mover o braço de leitura até a trilha em que se encontram os dados e esperar até que a rotação do disco traga-os à cabeça de leitura<sup>1</sup>.

Em sistemas de arquivos distribuídos, pode-se ter caches tanto no cliente como no servidor (veja figura 2.2), evitando assim que o cliente acesse muito a rede para obter os dados do servidor, enquanto que o servidor diminui o acesso ao meio físico de armazenamento dos dados para enviá-los ao cliente.

O uso de cache é uma boa solução para o problema de desempenho no acesso aos arquivos, porém existem problemas como o de sincronização dos dados do cache com os dados do meio físico. Assim, se algum outro cliente alterar os dados no servidor, este precisa avisar a todos os clientes que seus caches podem estar com uma versão antiga dos dados.

Além disso, o tamanho do cache é reduzido, o que gera a necessidade de um algoritmo para saber quais dados devem permanecer no cache e quais podem ser removidos para dar lugar a novos dados. O ideal, segundo [Tan92], é remover somente os dados que não serão mais acessados, deixando somente aqueles que serão. Como não é possível prever isso, foram estudadas várias técnicas e algoritmos para que o resultado final chegue o mais próximo disso. O algoritmo *LRU*<sup>2</sup> (Least Recently Used), segundo [Tan92], é o que melhor se aproxima do ótimo e, talvez por isso, o mais usado nesse tipo de situação. Assim, sempre que um novo dado é acessado, este é incorporado ao cache. Se o cache estiver cheio, são removidos os dados que foram acessados há mais tempo, para dar lugar aos dados que estão vindo. Porém, se os dados retirados do cache tiverem sido alterados, estes devem ser enviados de volta ao servidor ou ao disco para serem gravados. Naturalmente, conforme o padrão de uso pode ser mais interessante usar outras políticas de substituição.

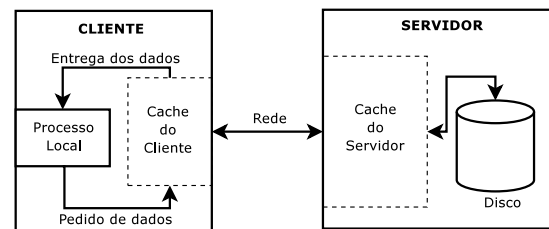


Figura 2.2: Cache no cliente e no servidor

### 2.1.3 Transparências para o Usuário

Alguns SADs implementam características que o tornam transparentes para o usuário, que não precisa saber detalhes sobre o sistema de arquivos. Algumas dessas transparências [Gal00] são:

- **Nome:** O nome do recurso a ser utilizado (como um arquivo ou diretório) não deve indicar ou conter indícios de onde este está localizado;
- **Localização:** O usuário não precisa fornecer a localização física do recurso para encontrá-lo;

<sup>1</sup>Para maiores informações sobre os tempos de acesso em discos magnéticos, veja o Apêndice B.

<sup>2</sup>Existem várias outras políticas, mas foge ao escopo deste texto citá-las.

- **Acesso:** O usuário não perceberá se o arquivo que está sendo usado é local ou remoto. Essa é a filosofia usada no sistema de arquivos virtual (**VFS**) do Solaris e do Linux;
- **Replicação:** Os arquivos do SAD podem ter cópias armazenadas em locais diferentes. O usuário não deve perceber que existem várias cópias do mesmo arquivo. Para ele só será apresentada uma e quem a escolherá é o SAD;
- **Concorrência ou Paralelismo:** Vários usuários podem acessar o mesmo arquivo ao mesmo tempo, mas isso não deve ser perceptível para esses usuários;
- **Falha:** O SAD deve garantir que o acesso aos arquivos seja ininterrupto e sem falhas, sem que o usuário saiba como isso é tratado.

## 2.2 Serviços Oferecidos pelos SADs

Para proporcionar um ambiente simples e fácil de usar, escondendo toda a complexidade por trás dos engenhosos algoritmos e idéias desenvolvidas pelos pesquisadores em sistemas de arquivos distribuídos, existem vários serviços<sup>3</sup> oferecidos pelos SADs. Muitos deles acabaram se tornando essenciais e são adotados em vários sistemas. Além disso, por ser muito comum encontrá-los, vários possuem uma interface comum independente da implementação, para ajudar o desenvolvedor das aplicações que irão usar tais SADs.

### 2.2.1 Serviço de Nomes Distribuído

O serviço de nomes se preocupa em indicar a localização de um determinado arquivo, dado o seu nome ou caminho. Se a localização do arquivo estiver armazenada no nome dele, como por exemplo *jaca:/tmp/teste*, então esse serviço de nomes não provê transparência de localização. Para prover essa transparência, o nome ou caminho de um arquivo não deve ter indícios de sua localização física. Caso esse arquivo mude de lugar, ou tenha várias cópias, o seu nome ou caminho não precisará ser alterado para ser encontrado. Para isso, o serviço precisa oferecer ou **resolução por nomes**, ou **resolução por localização**, ou ambos.

Resolução por nomes mapeia nomes de arquivos legíveis por humanos para nomes de arquivos compreensíveis por computadores, que normalmente são números, facilmente manipuláveis pelas máquinas. Por exemplo, o endereço *www.example.com* é mapeado para o IP 192.0.34.166. Através desse conjunto de números é possível encontrar uma máquina na rede Internet, utilizando-se de tabelas de rotas de endereços mascarados<sup>4</sup> que indicam como chegar à posição desejada.

Resolução por localização mapeia nomes globais para uma determinada localização. Por exemplo, números de telefone possuem código do país, da localidade, etc. Se transparência por nome e por localização estiverem sendo utilizadas simultaneamente, pode ser muito difícil realizar um roteamento para determinar a localização de um determinado nome. Soluções com servidores centralizados ou distribuídos são opções, porém os centralizados podem se tornar um gargalo, enquanto

<sup>3</sup>Os serviços descritos aqui também são encontrados em [Gal00].

<sup>4</sup>As máscaras são aplicadas no IP que se quer atingir e o resultado é usado como chave para encontrar a rota na tabela de rotas.

os distribuídos precisam usar alguma técnica de descentralização, como por exemplo cada servidor é responsável por um determinado subconjunto de arquivos, ou cada um resolveria a localização de determinados tipos de arquivos.

### 2.2.2 Serviço de Arquivos Distribuído

O serviço de arquivos é responsável por fornecer operações sobre os arquivos que compõem o sistema, que podem ser armazenados de diferentes formas, dependendo do seu tipo e uso. Por exemplo, arquivos que compõem um banco de dados podem ser armazenados em formato de registros, arquivos que são usados por aplicações multimídia costumam ser armazenados em formato contínuo no disco, para agilizar sua leitura, etc.

Esse serviço também cuida das propriedades dos arquivos, como data de criação, data de alteração, tamanho, dono do arquivo, permissões de leitura, escrita e execução, além de qualquer outra informação relevante. Tais informações são chamadas também de **meta-dados**.

Também é responsabilidade desse serviço manter a integridade das operações realizadas nos arquivos. Por exemplo, quando uma aplicação altera algum arquivo, todas as demais aplicações que estiverem acessando-o devem perceber essa alteração o mais rápido possível.

Existem dois tipos de implementação de serviço de arquivos: **acesso remoto** e **cópia remota**, que podem ser com ou sem estado. No caso do acesso remoto, o cliente não possui um espaço para guardar os arquivos que estiver usando e toda e qualquer operação realizada com os arquivos será sempre através da rede. Isso pode deixar o sistema muito lento, já que depende da velocidade da rede.

Já no caso da cópia remota, o cliente recebe uma cópia do arquivo para trabalhar e, quando tiver terminado, devolve as alterações para o servidor. Isso só funciona se o cliente tiver espaço suficiente para armazenar o arquivo. A velocidade da rede só influenciará durante as transmissões do arquivo de um local a outro e a implementação só será considerada muito eficiente caso o arquivo seja totalmente alterado. Porém, se o cliente só se interessar por um determinado trecho do arquivo, recursos de transmissão estarão sendo gastos sem necessidade. Daí existe uma variante dessa implementação onde somente os blocos que se quer trabalhar são enviados para o cliente, chamada de **cache de bloco**.

É possível também que esse serviço lide com replicação de arquivos, que ajuda no acesso concorrente, dando maior velocidade para as aplicações dos clientes, ajudando-o também a deixar os arquivos sempre disponíveis, caso algum servidor fique fora do ar.

### 2.2.3 Serviço de Diretórios Distribuído

Esse serviço é responsável por manter a organização dos arquivos armazenados no sistema. Ele fornece uma interface para que os usuários possam arranjar seus arquivos de forma hierárquica, que é estruturada em diretórios e subdiretórios. Na maioria dos casos, um subdiretório só tem um único pai.

O serviço precisa manter uma lista de todos os diretórios ativos, junto com seus respectivos arquivos. Ele precisa ter a capacidade de identificar e remover arquivos que não estejam em diretório algum, como por exemplo quando um diretório é removido. No caso em que são permitidos

múltiplos pais, essa tarefa não é tão fácil, pois os arquivos ou subdiretórios ainda podem estar sendo referenciados por outros diretórios ou recursos. Para resolver esse problema, é colocado um contador de referências para arquivos e diretórios, que se chegar a zero significa que o arquivo ou diretório não possui nenhum outro recurso (como *hard links*, subdiretórios, etc.) apontando para ele, podendo então ser removido. Note que, geralmente, os links simbólicos não influenciam nesse contador.

Exemplificando, a figura 2.3 mostra uma estrutura de diretórios distribuída em dois servidores. Se for requisitada a remoção da ligação<sup>5</sup> A-E (um *hard link*), ou da ligação B-E, ou da ligação C-E (um link simbólico), somente a ligação pedida será removida. Porém, somente as duas primeiras alteram o contador do diretório E, indicando quantas referências apontam para ele. Assim, se forem removidas as ligações A-E e B-E esse contador chegará a zero e os nós E, F e G serão removidos. No caso, o diretório F está em um servidor diferente do diretório raiz a ser removido, mas o serviço de diretórios deve cuidar disto.

Algumas das operações sobre diretórios oferecidas pelos serviços de diretórios são criação, remoção, alteração, listagem, alteração de permissões. Além delas, o serviço também influencia no gerenciamento de arquivos, como nas operações de criação, remoção, mudança de nome, busca, duplicação, entre outras.

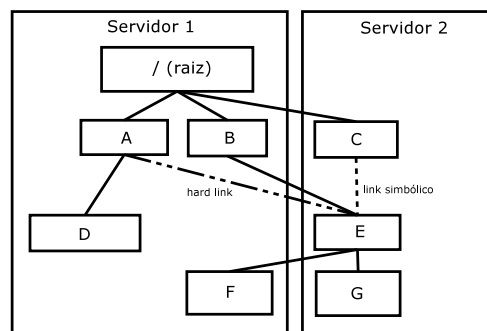


Figura 2.3: Estrutura de diretórios distribuída

## 2.3 Algumas Características Desejadas em SADs

Qual sistema de arquivos usar em um sistema distribuído? Para resolver essa dúvida, primeiramente analisa-se qual o tipo de aplicação que será utilizada e, a partir disso, tenta-se descobrir o que é mais importante para ela, como tolerância a falhas, acesso concorrente, ou alguma outra característica. Algumas dessas características [Gal00, Kon94] são descritas nessa seção.

### 2.3.1 Disponibilidade

Depender de um serviço da rede que saiu do ar por questões de falta de energia elétrica, manutenção ou falha do hardware é algo inconveniente, ainda mais quando ocorre perda dos dados. Dessa forma, existem vários estudos para evitar que os serviços deixem de ser oferecidos, seja qual for o motivo.

Se um servidor cair, ficar fora do ar ou da rede, o sistema de arquivos não pode perder informações e nem ficar inacessível total ou parcialmente. Além disso, o usuário não precisa saber como isso foi implementado. Ele simplesmente requisita um arquivo e o sistema de arquivos deve entregá-lo, mesmo que algum dos servidores esteja fora do ar. O arquivo não pode ficar indisponível e isso deve ser totalmente transparente ao usuário. Isso se chama transparência a falhas.

É muito comum sistemas ficarem indisponíveis não por falha do próprio servidor, mas por falha

<sup>5</sup>ao dizer ligação X-Y entende-se que o diretório ou link para Y está no diretório X.

na rede de comunicações. Caso um cliente deseje acessar um arquivo que está em um servidor inacessível naquele momento, o sistema operacional do cliente costuma travar o processo que o pediu até que o servidor volte a ficar acessível.

Uma das soluções para se resolver esse problema é a replicação dos dados, ou seja, o mesmo arquivo, ou parte dele, deve estar em diferentes servidores. Assim, caso algum servidor fique fora da rede, outro fornecerá os mesmos arquivos. Alguns sistemas de arquivos distribuídos foram criados levando esse conceito às últimas conseqüências, como é o caso do CODA, detalhado na seção 3.1.4.

### 2.3.2 Escalabilidade

Os sistemas distribuídos são, em geral, projetados e configurados pensando-se na configuração da rede naquele momento. Porém essa rede pode aumentar, ou seja, dezenas ou centenas de novos nós podem ser adquiridos e conectados nesse sistema. A menos que se tenha considerado essa situação no momento do projeto da rede, dificilmente um sistema de arquivos distribuído apresentará bom desempenho para servir a todos os clientes após esse crescimento [Kon94].

Vários problemas podem ocorrer, dentre eles, a inutilização da vantagem de se usar cache quando o servidor responde a vários pedidos de vários clientes. O servidor mantém os dados enviados em cache, para permitir uma rápida resposta caso esse mesmo dado seja requisitado novamente. No caso de se ter muitos clientes, têm-se muitos pedidos diferentes, fazendo com que as tabelas do cache sejam atualizadas com freqüência, sem a reutilização dos dados lá contidos.

Caso se tenha cache do lado dos clientes, ao se alterar um arquivo que está sendo usado por muitas outras máquinas, o servidor terá que avisá-las que o cache local das mesmas está inválido e todas deverão se atualizar com a versão do servidor, causando sobrecarga.

Por outro lado, caso se tenha estimado que a rede seria muito grande e se tenha distribuído o sistema de arquivos em muitos servidores, fica difícil descobrir onde um arquivo está armazenado fisicamente. Por exemplo, se para abrir um arquivo um cliente tiver que perguntar para cada servidor se ele é o responsável por aquele arquivo, certamente haverá um congestionamento na rede. Caso se tente resolver isso colocando um servidor central para resolver todos os caminhos para os arquivos, indicando a localização do mesmo, tal servidor sofrerá sobrecarga.

Um sistema escalável é um sistema que leva em conta esses problemas e tenta evitar sua ocorrência quando o número de clientes aumenta muito. O ANDREW é um sistema de arquivos que conseguiu adotar uma solução satisfatória [Kon] para esses problemas, através da descentralização das informações e da hierarquização da rede. Esse SAD é descrito na seção 3.1.3.

### 2.3.3 Segurança

Compartilhar arquivos entre vários ambientes e usuários é uma das vantagens que os sistemas de arquivos distribuídos trazem. Porém, deixar que outras pessoas possam acessar arquivos confidenciais é um grande problema. Dessa forma, torna-se necessário adotar mecanismos de segurança, para evitar que pessoas desautorizadas tenham acesso aos arquivos do sistema.

Sistemas Unix adotam um método baseado em **permissões** para controlar o acesso aos seus arquivos [Kon94]. Cada arquivo possui informações sobre quais usuários podem acessá-lo e de que maneira.

Nos sistemas distribuídos que executam sob o Unix, quando um servidor recebe um pedido para enviar dados de um determinado arquivo, ele também recebe informações sobre qual usuário está tentando realizar tal acesso. Com isso, verifica se tal usuário tem permissão suficiente para realizar essa solicitação, fazendo uma comparação com as informações de permissões do arquivo.

Outra forma de implementar esse controle de segurança é um sistema baseado em **capacidades** [Tan92], que consiste em enviar ao servidor uma prova de que ele possui a capacidade de acessar um determinado arquivo. Na primeira vez que o usuário acessa tal arquivo, é enviado ao servidor sua identificação e o servidor, por sua vez, retorna um código que é a sua prova de capacidade para acessar aquele arquivo. Nas próximas requisições, o cliente não precisa se identificar novamente, bastando apenas enviar a prova de sua capacidade. Deve-se tomar cuidado para não criar provas de capacidade que sejam fáceis de ser forjadas.

É possível, também, implementar o controle de segurança através de **listas de controle de acesso** [Tan92], onde cada elemento da lista possui as permissões que cada usuário tem para acessar determinado arquivo. Isso evita que se crie, por exemplo, uma matriz de arquivos por usuários, onde cada elemento representa o tipo de acesso (o que utilizaria muita memória, dado que muitos desses elementos seriam iguais). O sistema de arquivos distribuído ANDREW utiliza esse mecanismo de listas de controle de acesso.

O controle no acesso aos arquivos é uma das medidas de segurança para protegê-los. Porém, caso haja outras máquinas no caminho de duas máquinas confiáveis, existe o risco de se ter dados interceptados ou, até mesmo, adulterados. Uma forma de se resolver esse problema é criptografar as informações antes de enviá-las.

O sistema de arquivos SWALLOW [Kon94] é um sistema de arquivos distribuído que transmite os arquivos criptografados. Ele funciona em um sistema baseado em capacidades, onde a prova da capacidade é a chave criptográfica. A vantagem é que o servidor não precisa verificar se a prova da capacidade é autêntica: se ela não for, o cliente não conseguirá decodificar os dados.

Meios mais modernos e eficazes para o controle da segurança no acesso e manipulação dos dados armazenados podem ser encontrados em [Gal00].

### 2.3.4 Tolerância a Falhas

Durante a transmissão dos dados entre servidores e clientes, falhas podem ocorrer, seja por excesso de tráfego de pacotes pela rede, seja por algum dos servidores estar sobrecarregado. Além disso, podem ocorrer falhas de hardware, especialmente dos mecanismos de armazenamento, de transmissão, etc. Esses problemas acontecem em grande parte porque os sistemas distribuídos são implementados sobre redes de computadores que não são totalmente confiáveis.

Dessa forma, um sistema distribuído precisa usar um protocolo de comunicação com capacidade para detecção de erros de transmissão. Assim, caso uma mensagem chegue alterada no seu destino, o protocolo precisa perceber isso e retransmiti-la. Isso deve ocorrer também para mensagens que se perderam no caminho. Um outro problema que a rede pode ter é o seu particionamento por tempo indeterminado.

Além disso, o hardware dentro das máquinas também pode apresentar falhas. Por exemplo, um disco rígido pode deixar de funcionar de um momento para outro. Nesse caso, soluções como redundância física do equipamento (realizada através de hardware) ou redundância controlada pelo

próprio sistema distribuído, que cuidaria de replicar os dados, já evitaria a perda das informações armazenadas.

Seja qual for o problema, o sistema deve evitar que o cliente fique aguardando uma resposta por muito tempo, ou que seus dados sejam danificados ou até mesmo perdidos. Isso significa que o serviço precisa ter **disponibilidade** e **confiabilidade**.

Porém, essas características podem ser conflitantes se não forem bem aplicadas. Por exemplo, para garantir a confiabilidade é necessário implementar redundância dos dados, porém a complexidade que isso gera pode aumentar demais a carga do servidor, comprometendo a disponibilidade, pois as respostas aos clientes seriam mais lentas.

Outro mecanismo que auxilia a confiabilidade é a **transação**. Ela evita que o conteúdo de algum arquivo fique em um estado inconsistente caso haja uma queda do servidor ou cliente durante a execução de alguma operação sobre o mesmo. Maiores detalhes sobre transações são descritas na próxima seção.

### 2.3.5 Operações Atômicas

Uma operação em um arquivo é dita **atômica** quando as etapas da mesma não podem ser percebidas por outros processos exteriores a esta operação [Tan92]. Assim, antes dessa operação o arquivo apresenta um estado e após outro, sem que apresente nenhum outro estado intermediário durante a operação. Caso alguma etapa falhe durante a operação, o arquivo volta ao estado inicial. Dessa forma, ou todas as etapas são realizadas com sucesso, ou nenhuma será realizada. Operações de leitura, escrita, criação ou remoção de um arquivo são implementadas de forma atômica pela maioria dos sistemas de arquivos.

**Transações** são mecanismos que permitem realizar uma seqüência de operações de forma atômica. Tais mecanismos disponibilizam determinados comandos para os usuários para que possam escolher quais operações serão executadas dentro de transações. Para montar uma transação, existem os comandos *início* e *fim*. O comando de início avisa ao sistema que todas as operações a partir daquele ponto estarão dentro da transação e o comando de finalização indica que não virá mais nenhuma operação para aquela transação.

Assim, caso alguma dessas operações falhe, o sistema ou desfaz, ou aborta todas as alterações que as operações antes daquela realizaram. Isso é chamado de *rollback* ou *abort*. Caso todas as operações sejam executadas sem problemas ou erros, ao chegar no fim da transação é realizado um *commit*, ou seja, todas as alterações que foram executadas são efetivadas e persistidas, de tal forma que outros processo possam percebê-las.

Com isso as transações implementam a semântica do tudo ou nada, ou seja, ou todas as operações são executadas com sucesso, ou nenhuma será executada. Isso faz das transações um importante mecanismo de tolerância a falhas, pois elas evitam que pequenas falhas prejudiquem a integridade de todo o sistema.

Embora as transações sejam implementadas de forma praticamente obrigatória em sistemas de bancos de dados, elas não costumam ser implementadas em sistemas de arquivos. Os sistemas LOCUS e o QuickSilver [Kon94] são algumas exceções à regra, pois implementam transações em sistemas de arquivos distribuídos.

### 2.3.6 Acesso Concorrente

Vários usuários podem acessar vários arquivos, ou os mesmos arquivos, sem sofrer danos, perda de desempenho ou quaisquer outras restrições. Isso tudo deve ocorrer sem que o usuário precise saber como o acesso é realizado pelos servidores. Assim, é necessário haver transparência de concorrência e de paralelismo.

O maior problema encontrado nas implementações desse tipo de solução é quanto à sincronização dos arquivos, o que inclui leitura e escrita concorrente. A leitura concorrente pode ser implementada facilmente se não houver escrita concorrente, pois quando um arquivo estiver sendo lido, certamente ninguém poderá escrever nele. Caso também se queira escrita concorrente, deve-se levar em conta que quando um cliente escreve em um arquivo, todos os leitores devem ser avisados que o arquivo foi alterado e todos escritores precisam tomar cuidado para não escrever sobre as alterações que foram feitas por outros. Assim, ou vale a última alteração, ou os escritores discutem entre si para tentar fazer uma fusão das alterações.

Para se ter uma idéia da complexidade desse problema, imagine duas operações bancárias simultâneas na mesma conta (exemplificado na figura 2.4). Uma delas é um saque de R\$100,00 e outra é um depósito de R\$1000,00. Antes dessas operações, suponha que essa conta possua R\$100,00 de saldo e também suponha que esse valor esteja armazenado em um arquivo de um sistema de arquivos distribuído. Quando o cliente da conta for realizar o saque, a aplicação irá armazenar em memória o valor atual do saldo, assim como acontecerá com a aplicação do outro caixa que estará recebendo o depósito. Esta aplicação, então, irá adicionar ao saldo o valor do depósito e gravará no arquivo o novo saldo, que será de R\$1100,00. Porém, a primeira aplicação irá subtrair do valor armazenado em memória (que para seu contexto é de R\$100,00) o valor do saque e gravará o resultado (R\$0,00) no mesmo arquivo, sobrescrevendo o valor lá existente. Dessa forma, o cliente perderia seu depósito.

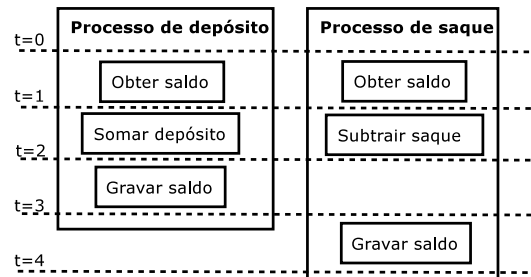


Figura 2.4: Problema na escrita concorrente

Para evitar esse tipo de problema, as aplicações que operam dessa forma podem agrupar um conjunto de operações no sistema de arquivos como sendo uma única transação, deixando a cargo do sistema operacional gerenciar a melhor forma de executar isso. Existem alguns mecanismos para o controle dessa concorrência, como por exemplo o uso de bloqueios, o controle de concorrência otimista e o de controle por data e hora, que são encontrados em [Tan92, Gal00].

O mecanismo de bloqueios destaca-se por ser amplamente utilizado, e baseia-se no bloqueio do arquivo que se quer acessar antes de acessá-lo, através de uma chamada ao sistema operacional ou ao sistema de arquivos. Caso um segundo processo queira usar o mesmo arquivo, tentará realizar o bloqueio, usando o mesmo comando que o primeiro processo. O sistema operacional ou o sistema de arquivos o avisará caso esse arquivo esteja bloqueado. Se estiver, cabe ao processo decidir se espera na fila pelo desbloqueio ou se continua seu processamento sem o acesso ao arquivo. Esse desbloqueio é realizado pelo processo detentor do arquivo, através de um comando, similar ao usado para o bloqueio.

Através desses bloqueios, é possível tornar as transações serializáveis, isto é, o resultado da operação de várias transações simultâneas é o mesmo obtido se elas fossem realizadas uma após a outra [Kon94]. Um protocolo para a realização dessa serialização é o protocolo de bloqueio de duas fases, onde na primeira fase ocorre o bloqueio de todos os arquivos a serem usados nessa transação e, na segunda fase, a liberação conjunta de todos os arquivos, após a realização das operações dentro dessas fases.

Porém esse protocolo pode gerar um travamento (*deadlock*), onde um processo esperaria a liberação de um arquivo que foi bloqueado por outro processo, que também estaria esperando a liberação de um arquivo que foi bloqueado por aquele primeiro processo, por exemplo. Para evitar travamentos em sistemas distribuídos, existem técnicas e algoritmos que fogem do escopo deste trabalho, devido à complexidade, mas que são descritos em [Gal00, Tan92].

### 2.3.7 Replicação de Arquivos

Em um ambiente distribuído, pode-se também distribuir a carga causada pelo acesso aos arquivos nos vários servidores que compõe o sistema. Pode-se replicar os arquivos em mais de um servidor, ou então replicar somente os arquivos mais acessados, ou ainda replicar somente os pedaços dos arquivos que costumam ter um alto nível de acesso. Note que o uso de cache em um sistema de arquivos pode ser encarado como uma replicação de arquivos, embora seu objetivo seja principalmente desempenho.

Além disso, se um sistema de arquivos oferece essa funcionalidade, a eficiência e a confiança do serviço de arquivos é generosamente aumentada. Eficiência em termos de tempo de resposta, carga do servidor e tráfego de rede. Confiança caso um determinado servidor caia ou fique indisponível, pois o serviço de arquivos ainda pode realizar suas obrigações por possuir cópias dos dados em outro ponto da rede.

Dessa forma, replicação de arquivos provê tolerância a falhas, já que o usuário pode não perceber que o servidor que ele estava usando caiu e que outro entrou no lugar para prover o recurso que ele estava usando. Daí o sistema também deve oferecer transparência de replicação (veja figura 2.5), pois o usuário não precisa saber como o sistema cuida da replicação desse arquivo.

O maior problema nessa característica do SAD é que a implementação pode ser muito complicada, pois é necessário manter os dados sincronizados e coerentes ao mesmo tempo. Existem soluções centralizadas e distribuídas [Gal00] para esse tipo de problema.

A centralizada consiste de um único servidor que cuida dos pedidos dos clientes através de *handles*. Com esses *handles*, os clientes acessam diretamente os arquivos através dos servidores secundários. Porém, caso o servidor primário caia, nenhum outro cliente conseguirá abrir nenhum outro *handle*, somente os que já estavam abertos continuam acessando os arquivos.

No caso da solução distribuída, existem dois tipos de implementações: a primeira utiliza comunicação em grupo, que consiste em quando ocorrer uma alteração por algum dos servidores,

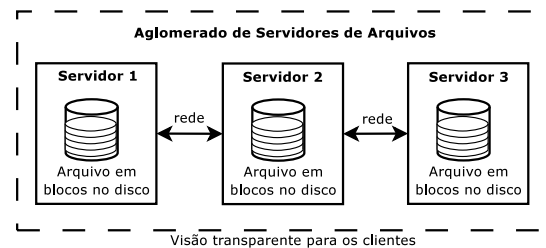


Figura 2.5: Aglomerado de servidores de arquivos, com visão transparente para os clientes

este manda um *broadcast* para os outros servidores dizendo que o arquivo foi alterado. Estes, por sua vez, podem alterar esse arquivo imediatamente ou somente quando forem utilizá-lo; a segunda utiliza votação e números de versão. Isso significa que quando um cliente pedir permissão para alterar um arquivo, os servidores votarão entre eles pra saber quem possui a versão mais recente. Esse servidor será o servidor padrão daquele arquivo e seu número de versão será incrementado.

Todas essas idéias, além de serem complicadas de implementar, geram alguns problemas. Manter a sincronização entre os servidores, para o caso de alterações no sistema de arquivos, é uma delas. Tal tarefa não pode interferir no desempenho (por causa da *disponibilidade*) e nem no uso do sistema pelos usuários (devido à *transparência*).

## 2.4 Conclusões Sobre o Estudo Realizado

Muitos serviços oferecidos pelos sistemas de arquivos distribuídos são tão transparentes para os usuários que estes acabam nem se dando conta da complexidade envolvida no acesso aos seus dados. Infelizmente tal complexidade gera custos, que costumam estar associados ao baixo desempenho do sistema como um todo.

Cabe ao projetista e aos desenvolvedores dos sistemas de arquivos a tarefa de ponderar desempenho, segurança e qualidade de serviço, para suprir as necessidades dos usuários conforme as aplicações às quais se destina tal sistema de arquivos.

No próximo capítulo, detalhamos alguns deles, indicando a razão pela qual foram criados, quais tipos de problema resolvem, quais os serviços oferecidos, suas maiores qualidades e seus maiores problemas.



## Capítulo 3

# Sistemas de Arquivos Paralelos e Distribuídos

Neste capítulo encontram-se alguns dos mais importantes sistemas de arquivos distribuídos e paralelos já desenvolvidos. Para cada um deles, fizemos uma breve descrição, contando algumas características próprias e importantes que o fizeram se destacar dentre outros do mesmo gênero, além de suas vantagens e seus maiores problemas.

### 3.1 Alguns Exemplos de Sistemas de Arquivos Distribuídos

Os sistemas de arquivos distribuídos se destacam pelo acesso remoto aos dados de forma transparente para os usuários. Nas seções a seguir, realizamos uma resenha sobre alguns deles, começando com aqueles que deram início a toda pesquisa na área. É importante deixar claro que a maior parte dessa resenha foi baseada nos textos [Kon, Kon94].

#### 3.1.1 Os Primeiros SADs

O primeiro SAD que se tem notícia, segundo [Kon94], usava a *ARPANET*, rede construída pelo Departamento de Defesa dos Estados Unidos em 1969, e entrou em funcionamento em 1973. Ele disponibilizava um repositório de dados para computadores que não possuíam capacidade de armazenamento adequada. Era chamado de *Datacomputer* e usava um serviço parecido com o FTP atual.

Depois dele, veio o *Interim File Server* (IFS), criado por pesquisadores do *Xerox Palo Alto Research Center* (PARC). Ele já organizava os arquivos privados e compartilhados em uma árvore de diretórios. Um sistema subsequente foi o *Woodstock File Server* (WFS), criado também pelo PARC, que permitia enviar aos clientes somente páginas dos arquivos, ao invés de enviar o arquivo completo, possibilitando trabalhar assim com máquinas sem discos locais.

Em 1977, o PARC criou o *Xerox Distributed File System*, destinado a oferecer uma base para a implementação de sistemas gerenciadores de banco de dados. Ele já implementava transações

atômicas envolvendo vários arquivos e servidores, usando um protocolo de duas fases e o acesso a pequenos trechos de arquivos.

Depois dele vieram o LOCUS (1980) que já implementava transparência de localização, replicação e transações atômicas aninhadas; o SWALLOW (início dos anos 80) do MIT, que usava uma técnica de controle de acesso concorrente baseado em *timestamps*; o *Acorn File Server* (início dos anos 80), desenvolvido para implantação de uma rede de microcomputadores em escolas a um custo muito baixo; e o VICE (1984), ancestral do AFS e do CODA.

### 3.1.2 NFS

*Network File System* [SRO96, Kon, Kon94, ML03], sistema de arquivos distribuído desenvolvido inicialmente pela Sun, é o SAD mais utilizado em sistemas Unix. Em 1985 a Sun tornou público seu protocolo, o que permitiu que outras empresas e desenvolvedores pudessem criar clientes e servidores compatíveis. Hoje em dia, já é possível encontrar implementações do NFS (tanto cliente como servidor) para quase todos os sistemas operacionais existentes, inclusive sistemas não-UNIX, como o Windows. Isso também foi facilitado pelo fato do NFS definir uma interface RPC<sup>1</sup> [SM88] que utiliza uma representação de dados independente de máquina chamada *External Data Representation* (XDR).

As versões mais usadas do NFS são as 2 e 3, porém já existe a RFC3530 [SCR+03] que descreve o NFSv4. Assim como as versões antigas são incompatíveis entre si, essa nova versão também será. As diferenças e características de cada uma dessas versões, levando em conta funcionamento, desempenho e segurança, estão detalhadas na seção a seguir.

#### Características do NFSv2 e NFSv3

Os servidores NFSv2 e NFSv3 não guardam o estado das transações realizadas. Isso faz com que não percam nenhuma informação enviada em caso de queda, agilizando sua recuperação. Os clientes também não precisam se preocupar com essa queda, pois basta pedir os dados novamente para o servidor, até que ele responda.

Por outro lado, servidores sem estado não conseguem gerenciar *locks* e nem realizar transações atômicas. Existem soluções disponibilizadas à parte para resolver alguns desses problemas, como um servidor de *locks*, chamado de *Network Lock Manager* [NLM98], para auxiliar as políticas de acesso a arquivos de forma concorrente. Também pelo fato do NFS não manter estado, ele não pode controlar o acesso concorrente aos seus arquivos e nem garantir a sua consistência.

No NFSv3 o mecanismo de cache do servidor foi alterado para possuir tamanho variável (antes era constante) e sua política de escrita foi alterada do *write-on-close* (ao se fechar o arquivo, este é gravado em disco) para o *delayed-write* (o arquivo é gravado em disco após ficar algum tempo no cliente sem ser alterado). Assim, caso um arquivo seja usado constantemente e depois apagado, nada será gravado.

Outra vantagem do protocolo NFSv3 em relação ao NFSv2 é o fato de que este último limitava o tráfego de dados de arquivos em blocos de 8KB, enquanto que aquele permitiu enviar dados entre

---

<sup>1</sup>*Remote Procedure Call*, método usado para que uma máquina possa executar operações em outra máquina.

servidor e cliente em blocos de até 56Kbytes via UDP. Além disso, no NFSv2 o servidor só retorna o resultado da gravação desses 8Kbytes após eles estarem gravados fisicamente. Isso consumia muito tempo pois só se gravava em blocos de 8KB. No NFSv3 o disco pode gravar uma quantidade de dados maior simultaneamente, pois o servidor retorna uma resposta do pedido de escrita ao cliente antes de realmente gravar os dados no disco, acumulando-os para escrever de uma só vez.

## Segurança

No NFSv2, o cliente era responsável pelo controle de acesso aos arquivos, sem nenhuma validação por parte do servidor). Isso mudou na versão 3, onde o servidor passou a tomar tal decisão, usando o mesmo esquema de segurança dos sistemas de arquivos locais Unix. Quando um cliente faz um pedido, ele envia o *uid* e o *gid* do usuário solicitante, e, através de uma consulta às permissões do arquivo local em questão, o servidor decide se libera o acesso ao cliente ou não.

Porém, isso necessita de sincronização de *uid* e *gid* entre as máquinas da rede. Para resolver isso foi criado o *Network Information Service* (NIS), um serviço de informações distribuído que é usado para fornecer tais informações a todos os nós da rede.

Percebe-se facilmente a fragilidade disso, dado que se um cliente não for confiável ele pode fornecer *uids* e *gids* falsos, podendo, assim, acessar e alterar arquivos de outros usuários. Para resolver esse problema, o NFS possui a possibilidade de autenticação mútua entre cliente e servidor, baseada no método DES de criptografia, onde as chaves são fornecidas pelo NIS. Porém, a informação trafegada não é criptografada, o que possibilita que intrusos possam obter pedaços de arquivos que trafeguem pela rede.

## O Novo Protocolo NFSv4

Após alguns anos após o lançamento da especificação do protocolo NFSv3, foi criada uma nova versão que revê vários conceitos que não estavam presentes nos protocolos anteriores, que causam mudanças drásticas [ML03] no que se conhecia até então sobre o NFS. Essa nova versão está disponível para o Linux a partir da versão 2.6 do seu *kernel*.

Nela, o servidor mantém o estado dos arquivos em conjunto com os clientes, diferentemente das versões anteriores. Assim, é possível que um determinado cliente pergunte ao servidor o que outros clientes estão fazendo com determinado arquivo. Isso pode indicar ao cliente se vale a pena ou não realizar um cache dos dados de forma mais agressiva.

É possível também bloquear e compartilhar partes de arquivos através do próprio protocolo NFS, sem a necessidade de servidores externos (como o NLM). O mecanismo para isso é baseado em *leases*, ou seja, um cliente NFS pede ao servidor um contrato de bloqueio temporário (*lease*) e deve manter contato com o mesmo para continuar prolongando esse contrato conforme a necessidade.

Além disso, foi introduzido um esquema de delegação de arquivos, onde o cliente NFS pode acessar e modificar o arquivo dentro do seu cache local sem a necessidade de mandá-lo para servidor, até que o servidor contate o cliente avisando que outro cliente gostaria de acessar o arquivo, quando então este é atualizado no servidor. Isto reduz o tráfego de rede consideravelmente nos casos em que os clientes não desejam acessar um conjunto de arquivos concorrentemente.

Quanto à comunicação entre cliente e servidor, o NFSv4 usa chamadas RPC compostas, ou seja, uma mesma chamada RPC pode conter uma operação complexa envolvendo bloqueio, abertura, leitura, etc. Essas chamadas são realizadas através de conexão TCP, ao contrário das versões mais antigas, que usam UDP.

Em relação à segurança, o NFSv4 possui mecanismos sofisticados, e todas as implementações de clientes obrigatoriamente devem tê-los. Dentre esses mecanismos estão inclusos **Kerberos 5** e **SPKM3**, juntamente com o tradicional **AUTH\_SYS** [Eis99]. Além disso, uma nova API foi criada para permitir estender esse mecanismo no futuro.

No NFSv4 a interpretação das listas de controle de acesso (ACLs) foi padronizada tanto para o ambiente Posix quanto para o Windows. Os nomes de usuário e grupo são armazenados em forma de texto e não mais como valores. Além desses, existe a possibilidade de se ter outros atributos, conforme a necessidade. Todos eles são armazenados na codificação UTF-8.

Por fim, todos os protocolos NFS existentes (tais como stat, NLM, mount, ACL e NFS) convergem para uma única especificação, proporcionando uma melhor compatibilidade com os *firewalls* de rede, além de introduzir no protocolo suporte a migração e replicação de arquivos.

### Análise Crítica

O NFSv4 tornou sua manutenção e uso muito mais simples, por possuir, agora, controle de bloqueios encapsulado no mesmo protocolo e não mais através de sistemas de terceiros, além de permitir o controle da consistência dos arquivos que estão nos caches dos seus clientes.

O controle da segurança no acesso aos arquivos era muito simplificado e frágil, permitindo que clientes não confiáveis pudessem acessar arquivos de maneira desonesta. Isso foi resolvido na versão 4 do protocolo, onde mecanismos avançados de segurança e autenticação foram incorporados.

Outro problema era o grande consumo de recursos da rede nas operações entre cliente e servidor (devido à interface RPC/XDR). Associado à política de cache, o NFSv3 não é muito recomendado para aplicações que necessitam de acesso contínuo aos arquivos. O NFSv4 resolve esse problema pois é possível enviar múltiplos pedidos ao servidor através da mesma chamada RPC, além do uso do cache ter melhorado por conta do controle de estado no acesso aos arquivos.

Uma excelente característica é a transparência que o sistema de arquivos dá para o usuário final, que nem sequer percebe estar lidando com arquivos remotos. Na versão 4, onde os controles de bloqueios e estado são nativos do protocolo, isso é ainda mais evidente, dando a impressão de que se está usando o sistema de arquivos local.

Além disso, o fato de ter sua especificação aberta para que qualquer um possa implementar seu servidor ou cliente permitiu que ele se tornasse o sistema de arquivos distribuído mais utilizado no mundo.

### 3.1.3 AFS

O projeto *ANDREW* [Kon, Kon94] começou na Universidade Carnegie-Mellon em 1983, com apoio da IBM. Seu objetivo era projetar e implementar um sistema distribuído para o ambiente acadêmico de ensino e pesquisa, que ofereceria a cada professor e aluno uma estação de trabalho com um sistema operacional compatível com o UNIX BSD. Além disso, a partir de qualquer um desses

computadores, o usuário deveria ter a mesma visão do sistema. Essa transparência de localização deveria ser altamente escalável, podendo aceitar de 5 a 10 mil estações de trabalho nessa rede.

Ao lado da escalabilidade, um outro problema importante que os desenvolvedores iriam enfrentar era a segurança. Com tantos clientes e usuários, era praticamente impossível confiar em todos sem provocar uma fragilidade na segurança de todo o sistema. O ANDREW sofreu modificações gradualmente durante sua existência, que foi dividida em três fases:

### AFS-1

Durante o ano de 1985, o AFS-1 operou 100 estações de trabalho e 6 servidores. O desempenho do sistema era razoável tendo até 20 clientes conectados por servidor, porém um trabalho pesado que algum deles realizasse poderia degradar o funcionamento do sistema como um todo de forma intolerável. Além disso, a administração do sistema era complicada, dado que os administradores não dispunham de ferramentas adequadas.

### AFS-2

A partir de toda a experiência adquirida com o AFS-1 e com todos os seus testes de desempenho, foi possível criar uma nova versão muito mais eficiente. Eficiência ganha com o uso de algoritmos melhores para manutenção da consistência do cache, além de uma melhoria na implementação da resolução de nomes, comunicação e estrutura dos servidores. Funcionou desde o final de 1985 até meados de 1989.

O AFS-2 [Kon] trouxe o conceito de *callback*, que permite ao cliente abrir e fechar um arquivo várias vezes sem precisar acessar o servidor. Quando um cliente recebe um arquivo do servidor, ele também recebe um *callback*, que é uma promessa de que ele está com a versão mais recente do arquivo, que pode ser quebrado ou quando um cliente atualiza um arquivo, ou quando o servidor recebe uma nova versão desse arquivo de um outro cliente. A cópia local pode ser utilizada quantas vezes se desejar, contanto que o cliente possua um *callback* válido.

O problema de escalabilidade foi amenizado ao se passar grande parte do trabalho dos servidores para os clientes: todas as operações de leitura e escrita são realizadas na cópia local do arquivo. Somente quando o arquivo alterado é fechado ele é então transferido de volta para o servidor. Uma consequência desta técnica é que o AFS utiliza semântica de sessão (e não a semântica UNIX de acesso concorrente a arquivos). Assim, um cliente só perceberá a alteração de um arquivo, feita por um outro cliente, quando ele abrir o arquivo depois que o outro já o tiver fechado.

Como vários usuários passaram a depender do sistema, percebeu-se a importância da disponibilidade dos dados, já que a queda de um servidor provocava interrupção dos trabalhos por vários minutos. Assim, em 1987 iniciou-se o desenvolvimento de um sistema de arquivos de alta disponibilidade, baseado na escalabilidade e segurança do AFS, denominado CODA (mais detalhes na seção 3.1.4).

### AFS-3

O AFS-3, ou simplesmente AFS, foi uma iniciativa de tornar o ANDREW um sistema comercial no início de 1990. Para tanto, era necessário adotar alguns padrões, como por exemplo o *Virtual File System* (VFS) da SUN, possibilitando integrá-lo a outros sistemas de arquivos.

Foi implementado o protocolo Kerberos para autenticação mútua entre clientes e servidores, resolvendo assim o problema de segurança no acesso aos dados. A proteção dos arquivos é baseada em listas de controle de acesso, que especificam quais usuários, ou grupos, têm que tipo de acesso sobre eles.

Além disso, a partir dessa implementação os arquivos deixaram de ser cacheados em sua totalidade e passaram a ser transferidos, conforme a necessidade, em blocos de 64 Kbytes, reduzindo assim a latência da abertura e tornando possível o acesso a arquivos grandes que não cabem no disco local.

### Princípios do AFS

A fim de atingir seus objetivos, foram adotadas algumas regras para o desenvolvimento do ANDREW e, conseqüentemente, do AFS:

1. Sempre que for possível deve-se realizar as operações no cliente e não no servidor, distribuindo, assim, as tarefas entre as máquinas disponíveis, evitando sobrecarregar o servidor;
2. Sempre que possível usar o cache para diminuir o tráfego dos dados e a carga dos servidores;
3. Explorar os tipos de acesso aos arquivos. Por exemplo, manter arquivos temporários na máquina local, replicar em diversos servidores arquivos executáveis que são muito usados e raramente alterados, etc;
4. Replicar serviços e informações sempre que possível, evitando limitar a escalabilidade de todo o sistema à capacidade dessa máquina central;
5. Confiar no menor número possível de entidades (segurança);
6. Agrupar o trabalho sempre que possível. Por exemplo, realizar uma leitura de 50 KB é muito mais eficiente que realizar 50 leituras de 1 KB.

### Características do AFS

O espaço de nomes do AFS é dividido em duas partes: **os locais**, que consistem dos arquivos cacheados, temporários e daqueles necessários para a inicialização da máquina; **os remotos**, que são aqueles que podem ser encontrados a partir de qualquer cliente conectado na mesma rede.

Ao contrário do NFS, no AFS toda informação sobre os nomes dos arquivos e diretórios é armazenada nos servidores. Deste modo, a manutenção dos clientes é trivial e a uniformidade do espaço de nomes é uma conseqüência natural da configuração dos servidores. Quando um cliente precisa acessar um arquivo remoto, ele pergunta a todos os servidores por sua localização, que é então guardada em cache local para futuras consultas.

O acesso concorrente a arquivos pode ser controlado a partir de chamadas UNIX para *flock*, que administra bloqueios ao arquivo, de forma emulada. O responsável por esse bloqueio é o servidor que detém tal arquivo. Caso esse bloqueio dure mais de 30 minutos, o servidor automaticamente o libera, para evitar que a queda de um cliente impossibilite o acesso aos arquivos que ele bloqueou.

Em 1998 havia mais de 100 células AFS por todo o mundo dando a seus usuários a possibilidade de compartilhar seus arquivos através de diferentes continentes usando uma interface de sistema de arquivos parecida com a do UNIX. O AFS começou a ser comercializado pela Transarc Corporation, que foi comprada pela IBM. No momento em que esse texto foi escrito, o AFS estava na versão 3.6, sendo distribuído de forma independente do ANDREW. Para maiores informações visite <http://www-3.ibm.com/software/stormgmt/afs/library/>.

### Análise Crítica

O AFS é um sistema de arquivos distribuídos que evoluiu muito desde sua primeira versão. Pensando-se sempre em escalabilidade, transparência de localização e segurança, ele foi implementado usando conceitos simples, mas que são de extrema importância para se atingir tais objetivos.

Ele oferece um serviço altamente escalável e seguro, através da adoção de semântica de sessão no acesso concorrente a arquivos, na utilização de grandes caches no disco local do cliente e no uso de listas de controle de acesso, juntamente com o protocolo de autenticação mútua Kerberos.

Por causa do cache e da iniciativa de não se compartilhar arquivos temporários, os clientes necessitam obrigatoriamente de disco local. O espaço de nomes é dividido entre local e remoto, sendo que este último é mantido e organizado pelos servidores através de um banco de dados de localização.

#### 3.1.4 CODA

O CODA<sup>2</sup> (*Constant Data Availability*) [SRO96, Kon, AEK96, Kon94] começou a ser desenvolvido em 1987 pela Universidade de Carnegie Mellon, EUA, tendo sua origem a partir do AFS-2. Seu principal objetivo é fornecer operações desconectadas ao sistema de arquivos para computadores portáteis, que costumam ficar grande parte do tempo fora da rede. Isso provê uma máxima disponibilidade dos arquivos aos seus usuários.

Para que isso seja possível, o CODA implementa alguns mecanismos de replicação não presentes no AFS, dado que ele foi criado para lidar com estações de trabalho portáteis ou que permanecem conectadas aos servidores por curtos períodos de tempo.

### Replicação dos Dados

Pelo CODA, cada volume (um conjunto de diretórios do sistema de arquivos) é associado a um *volume storage group* (VSG), que consiste de um conjunto de servidores que replicam o volume. O conjunto de servidores acessíveis de um certo grupo em um certo momento é chamado de AVSG (*accessible VSG*). Essa organização é melhor visualizada na figura 3.1. A coerência entre as várias cópias de um arquivo é mantida por um sistema parecido com o de *callbacks* do AFS.

---

<sup>2</sup><http://www.coda.cs.cmu.edu/doc/html/index.html>

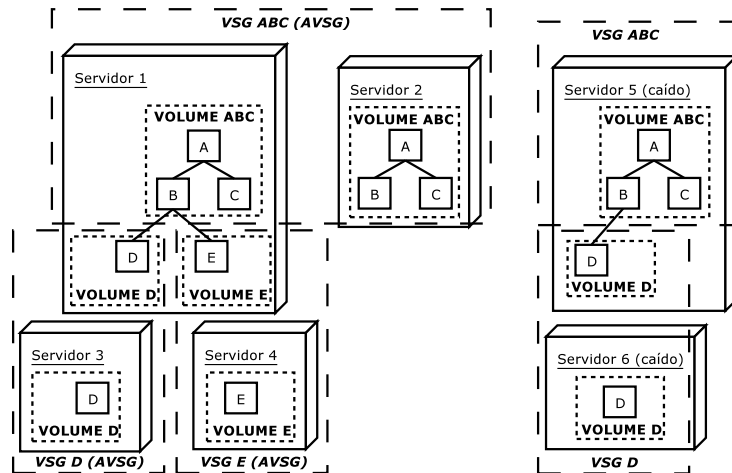


Figura 3.1: Volumes, VSGs e AVSGs

Quando um cliente envia uma atualização de um arquivo para o servidor, a atualização é enviada para todos os servidores AVSG usando um mecanismo denominado multiRPC. Além disso, são enviadas mensagens aos clientes quebrando o *callback* que eles possuem para aquele arquivo, invalidando o cache do mesmo.

Se um servidor que estava caído volta à rede, nada é feito inicialmente para atualizar seus arquivos. Porém, sempre que um cliente envia uma requisição para abrir um arquivo para o seu servidor preferido, ele também pede a todos os servidores AVSG que enviem a versão daquele arquivo que eles detêm. Assim, o cliente pode descobrir se existe algum servidor com uma cópia desatualizada, avisando-o para atualizar esse arquivo. Dessa forma, quem toma as iniciativas para atualização dos arquivos que possuem réplicas inconsistentes são os próprios clientes.

Essa replicação aumenta a disponibilidade dos arquivos, o que aumenta a segurança para os clientes encontrarem o que procuram e guardarem os dados que possuem. Por exemplo, se um computador portátil perder todos seus dados, a chance de recuperá-los com a replicação é maior. Além disso, o espaço em disco nos servidores tende a ser maior que nos clientes, facilitando ainda mais o uso dessa característica.

## Controle de Consistência

O CODA tenta prover ao máximo as operações desconectadas. Para isso, ele permite que os clientes possam ler e escrever seus arquivos de forma indiscriminada, a partir de qualquer servidor da rede que possua os dados que ele precise, mesmo que a rede seja particionada devido à queda de algum servidor ou conexão entre eles. Isso pode gerar perda de informação e acesso a dados inconsistentes quando, por exemplo, dois usuários alteram o mesmo arquivo em partições diferentes.

## Operações Off-Line

A parte mais interessante do CODA é a possibilidade de acessar um sistema de arquivos distribuído estando completamente desconectado da rede. Se um arquivo está armazenado localmente na máquina, o usuário pode ler e escrever no arquivo sem a prévia permissão do servidor.

Isso só é possível graças a um software chamado *venus*<sup>3</sup>, que é o responsável pelo sistema de arquivos do lado do cliente. Ele possui três estados de funcionamento:

- **Cacheando:** Esse é seu estado normal de funcionamento. Aqui a comunicação com os servidores é possível sempre que necessário, mas o cliente procura estar preparado para o caso de uma desconexão da rede, seja voluntária ou não;
- **Emulação:** Esse estado é atingido quando o cliente perde a conexão com os servidores. Nesse caso, o *venus* tenta fazer o papel dos servidores, disponibilizando as réplicas dos arquivos gravadas localmente, como se ainda estivessem sendo acessados através dos servidores;
- **Reintegração:** Assim que o computador é conectado à rede, entra-se no modo de reintegração, onde ele passa a fornecer aos servidores responsáveis, os arquivos em seu cache que sofreram alterações. Após o final dessa operação, volta-se ao primeiro estado.

## Desempenho

Alguns testes [SKM+93] realizados em situações normais de uso mostraram que o tamanho do cache local necessário para uma semana desconectado e o tempo de reintegração dos dados após esse mesmo período não são muito grandes.

Além disso, concluiu-se que os problemas de acesso concorrente, que poderiam causar conflitos na reintegração, são raros, dado que 99% das alterações dos arquivos são realizadas pelo mesmo usuário que já o alterou anteriormente.

O desempenho com 4 servidores replicados do CODA foi no máximo 5% pior que o do AFS, este sem replicação. Porém, o CODA se mostrou menos escalável que o AFS nesses testes [SKM+93].

## Análise Crítica

O CODA apresenta inovações que auxiliam usuários que necessitam de um sistema de arquivos distribuído de alta disponibilidade. Por exemplo, ele permite que um usuário defina os arquivos que devem estar acessíveis a todo momento, dando assim a facilidade de se conectar à rede por alguns instantes, atualizar seus arquivos e os da rede e voltar a se desconectar, para ir trabalhar em casa como se estivesse conectado.

A replicação dos dados permite aumentar ainda mais essa disponibilidade e a segurança dos dados, já que não só os servidores possuem os arquivos, mas também os clientes. O problema é que isso diminui as garantias de consistência dos arquivos em caso de acesso concorrente.

O CODA não respeita a semântica de sessão (ao contrário do AFS), dado que alterações realizadas por clientes desconectados são aceitas pelo sistema, mas não são informadas a outros usuários. Isso é tolerável, considerando o ganho extra de disponibilidade no sistema de arquivos.

<sup>3</sup><http://www.coda.cs.cmu.edu/doc/html/kernel-venus-protocol.html>

### 3.1.5 SPRITE

O SPRITE [Shi96] é um **sistema operacional distribuído**, desenvolvido pela Universidade da Califórnia. Seu projeto começou em 1984 com John Ousterhout [Ous96] e mais quatro estudantes de pós-graduação, motivados pelo fato dos sistemas operacionais da época não darem tanta importância ao suporte a redes locais para sistemas isolados.

O objetivo era criar um sistema operacional a partir do zero, implementando suporte à rede da melhor forma possível em seu *kernel*, de forma que várias máquinas cooperassem como se fossem uma só, isto é, onde os sistemas de armazenamento e processamento fossem compartilhados entre todos.

O sistema de arquivos foi desenhado para ser transparentemente distribuído entre os integrantes da rede, de tal forma que nenhum usuário percebesse se estava acessando um arquivo local ou remoto. Para alcançar alto desempenho, foi implementado um uso agressivo de cache de tamanho variável.

Além disso, o SPRITE é um sistema operacional que possibilita migrações de processos de modo transparente entre as máquinas, a fim de aproveitar os recursos computacionais de máquinas com pouca carga. Isso é alcançado facilmente pois os arquivos de *swap* da memória virtual são implementados como arquivos comuns nesse mesmo sistema de arquivos. Assim, basta congelar o processo, enviar todas as suas páginas para a memória virtual (que é visível em qualquer nó da rede, pois usam o mesmo sistema de arquivos), enviar para a outra máquina o conteúdo dos registradores juntamente com sua entrada na tabela de processos e descongelar o processo na nova máquina<sup>4</sup> (conforme exemplificado na figura 3.2).

Por compartilhar os sistemas de armazenamento e por distribuir o trabalho entre as máquinas, uma rede rodando SPRITE pode ser vista como um só sistema operacional. De fato, percebe-se que existe apenas uma partição raiz, uma base de dados de usuários e uma área de *swap* (memória-virtual) dentro do sistema de arquivos. Por exemplo, ao se executar o comando “finger”, verifica-se que retornará a lista de todos os usuários logados na rede e não em apenas uma máquina.

Além disso, uma rede SPRITE pode ser multi-plataforma, pois foi criado um arcabouço que separa informações dependentes de plataforma daquelas que são independentes, sendo que todos os nós da rede têm uma visão de todas as arquiteturas, permitindo assim um desenvolvimento entre plataformas.

#### Resolução de Nomes

A árvore de diretórios do SPRITE é dividida em sub-árvores disjuntas, chamadas de **domínios** (correspondentes aos volumes do AFS). Cada servidor é responsável por um ou mais domínios. Cada máquina da rede possui uma **tabela de prefixos**, que mapeia domínios em servidores.

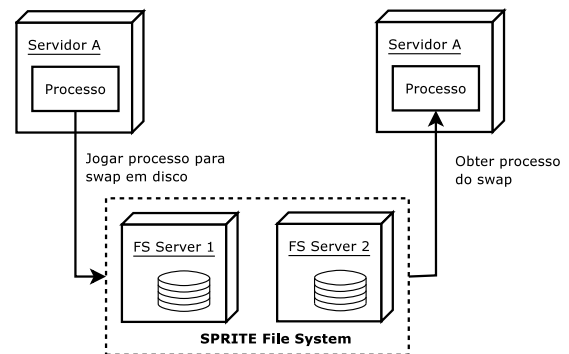


Figura 3.2: Migração de processos

<sup>4</sup>Vale a pena lembrar que isso não funcionaria entre máquinas de arquitetura de *hardware* diferentes.

Um domínio é identificado pelo maior prefixo do caminho do arquivo que se quer encontrar. Por exemplo, no caminho `/home/alunos/carvalho/arquivo.txt` o prefixo a ser procurado na tabela de prefixos seria `/home/alunos/carvalho`, se não encontrado então `/home/alunos`, se não encontrado então `/home` e, se não encontrado, então, por último, `/`. Com certeza um desses será encontrado, pois, na inicialização do cliente, este realiza um *broadcast* para encontrar a raiz desse sistema de arquivos, e a tabela de prefixos vai aumentando conforme há a necessidade de se acessar mais arquivos.

Quando se quer realizar alguma operação em um arquivo, procura-se o maior prefixo na coluna de prefixos que é também prefixo do caminho do arquivo. Com isso temos o servidor, para o qual serão enviados o índice e o caminho do arquivo sem o prefixo. O servidor tenta encontrá-lo e, caso consiga, envia para o cliente um **designador** desse arquivo. Com ele o cliente pode realizar operações no arquivo.

Caso o servidor não encontre o arquivo, ou caso ele perceba que o arquivo está em outro servidor, o cliente receberá uma resposta relacionada. A partir disso, tal cliente realiza um *broadcast* na rede perguntando pelo arquivo. O servidor responsável irá responder e o cliente poderá, então, atualizar sua tabela de prefixos com o domínio do servidor que respondeu. Isto ajuda na migração de domínios, pois os clientes que procurarem no servidor antigo serão redirecionados para o novo.

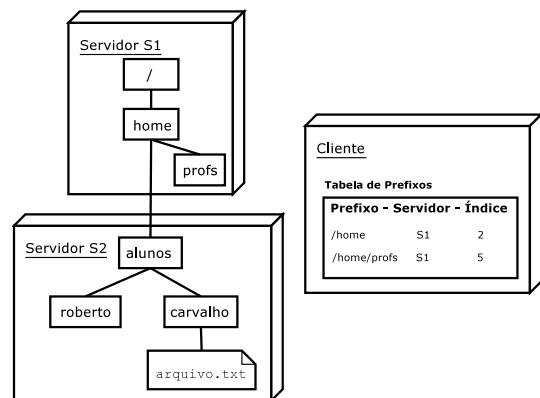


Figura 3.3: Árvores de diretórios e tabela de prefixos no SPRITE

## Cache

Os caches dos clientes armazenam os dados dos arquivos em blocos, sendo que somente aqueles que foram acessados estarão no cache. Já os servidores possuem em seu cache não somente blocos, mas também mapas da localização física dos mesmos no disco, que auxilia na manutenção do serviço.

Um dos grandes segredos para atingir alto desempenho foi a implementação da escrita em cache, ou seja, quando um processo pede para escrever dados em disco, eles são escritos nos blocos do cache, não precisando esperar pela resposta do servidor.

Com essa política, se o servidor ou o cliente caem, os dados do cache são perdidos. Se houver uma falha na escrita em disco, os processos já terminados não poderão ser avisados, o que comprometerá a coerência do sistema de arquivos.

Por outro lado, a adoção desse risco possibilita um ganho em eficiência decorrente das escritas atrasadas (*delayed-writes*). Levando-se em conta o fato de que 20% a 30% dos dados escritos em um sistema de arquivos são apagados em menos de 30 segundos (segundo [Shi96]), o uso dessas escritas atrasadas diminui a carga dos servidores e o uso da rede na mesma proporção.

O SPRITE oferece a semântica UNIX no acesso concorrente aos arquivos, isto é, mesmo quando vários clientes possuem em seus caches o mesmo arquivo, o SPRITE garante a consistência entre

essas cópias. Isto significa que o SPRITE garante que cada byte recebido por um pedido de leitura de um cliente é o resultado da última escrita naquela posição do arquivo, mesmo que ele esteja sendo compartilhado entre vários clientes. Tal garantia é realizada pelo servidor responsável por tal arquivo. Não há comunicação entre os clientes para isso. Para tanto, os clientes avisam o servidor quando abrem ou fecham arquivos (o que não acontece no NFS ou AFS).

## Disponibilidade

Pelo fato do SPRITE ter sido criado para oferecer alto desempenho com o menor investimento possível, ele não oferece nenhum tipo de replicação automática de domínios em vários servidores. Porém, para evitar que o sistema ficasse fora do ar por muito tempo, o SPRITE foi implementado para que as quedas fossem recuperadas o mais rapidamente possível.

Quando um servidor cai, toda a informação sobre os arquivos abertos nos clientes é perdida. Uma solução para isso foi manter nos clientes uma cópia dessas informações. Cada cliente sabe quais arquivos ele tem abertos e em que modo os mesmos estão. Assim, quando um cliente percebe que o servidor está em processo de recuperação, ele lhe envia essas informações. Quando todos os clientes que possuem arquivos abertos no servidor terminarem de enviar essas informações, teremos o estado do sistema refeito. Este método é chamado de *recuperação distribuída de estado*.

Além da retomada do estado do servidor, deve-se também verificar a consistência do sistema de arquivos físico. Para isso, foi adotado um sistema de arquivos baseado em log [vH03], diminuindo drasticamente o tempo de recuperação do mesmo, que caiu de vários minutos para alguns segundos.

No intuito de minimizar inconsistências no sistema de arquivos, um sistema de arquivos baseado em log se mantém informado sobre as mudanças que serão aplicadas em disco antes que elas aconteçam, através do armazenamento dos registros sobre essas mudanças em uma parte reservada do sistema de arquivos conhecida como *journal* ou *log*. Assim que esses registros estiverem seguramente gravados, o sistema de arquivos aplica as mudanças no disco de forma persistente e remove esses registros do log.

## Análise Crítica

O SPRITE é um sistema operacional distribuído cujo objetivo é criar uma rede de computadores comuns que tenha uma visão única por parte do usuário, onde todos os recursos são compartilhados entre todos os nós. Para isso, implementou-se um sistema de arquivos distribuído totalmente transparente, cujo desenho ajudou na elaboração de um método de migração de processos entre as máquinas de forma simples.

A resolução dos caminhos (*pathnames*) é efetuada através de tabelas de prefixos dinâmicas mantidas pelos clientes. Essas tabelas permitem um rápido acesso aos arquivos espalhados pelos domínios da rede.

A segurança dos dados que trafegam no SPRITE não é garantida, dado que todos os servidores e clientes que estão conectados na rede são considerados confiáveis, além do que nem sempre todas as permissões de acesso de cada diretório que constituem um caminho são verificadas.

Infelizmente a equipe de desenvolvimento do SPRITE se reduziu muito a partir de 1991, pois não haviam novos desenvolvimentos a serem feitos. Foi assim até 1993, quando se decidiu desativar

o projeto no início de 1994 (segundo [Ous96]). O SPRITE serviu de base para o estudo de sistemas de arquivos baseados em log [vH03], sistemas de arquivos baseados em distribuição de pedaços de arquivos entre servidores, recuperação em caso de quedas, sistemas de arquivos RAID<sup>5</sup>, entre outros.

## 3.2 Alguns Exemplos de Sistemas de Arquivos Paralelos

Sistemas de arquivos paralelos são SADs projetados para proporcionar alto desempenho sob grande demanda e concorrência de acesso. Como essa não é uma tarefa fácil, os projetistas acabam não se preocupando tanto com a transparência no acesso, a segurança ou mesmo a qualidade do serviço. Porém, isso vem mudando.

A seguir, realizamos uma resenha de alguns SADs que têm foco em desempenho, deixando um pouco de lado praticidade ou segurança, sendo muito usados por aplicações paralelas.

### 3.2.1 BRIDGE

O sistema BRIDGE [DSE88] foi um dos primeiros sistemas de arquivos que considerou distribuir não somente a carga entre os servidores, mas também partes dos arquivos. A maior preocupação no desenvolvimento desse sistema de arquivos distribuído foi diminuir o gargalo provocado pela transferência de dados dos arquivos e aumentar o desempenho de programas paralelos que necessitam acessar arquivos, e ao mesmo tempo deixando o programador livre da preocupação sobre como acessá-los ou onde os arquivos estão. Isso tudo seria trabalho do sistema de arquivos.

Tal gargalo pode ser considerado qualquer elemento no caminho entre o sistema de armazenamento físico e o programa que pediu os dados. Isso inclui a velocidade e latência do disco, da rede, do barramento de dados do computador sendo usado, etc.

O Sistema de Arquivos BRIDGE foi desenvolvido como um programa paralelo que mantém a estrutura lógica dos arquivos enquanto que os dados estão fisicamente distribuídos entre os vários componentes que compõe o aglomerado. A idéia usada para atingir esse objetivo foi a de *interleaved files*, onde blocos lógicos consecutivos são armazenados em diferentes nós físicos.

#### Interleaved Files

Um arquivo armazenado no esquema de *interleaved file* pode ser visto como uma matriz bidimensional (veja figura 3.4(a)), onde cada coluna é armazenada em um servidor de armazenamento diferente, que por sua vez possui um processador independente e é gerenciado por um sistema de arquivos local (LFS).

O serviço de nomes do BRIDGE é quem realiza o mapeamento de um dado arquivo de seu sistema com os vários arquivos locais que constituem seus blocos nos múltiplos servidores de armazenamento, usando o nome associado a eles pelo sistema de arquivos local.

Como o armazenamento de um arquivo se dá usando uma ordem *round-robin* para distribuição de seus blocos entre os vários servidores, esta informação é suficiente para mapear um nome de

---

<sup>5</sup>Sigla para *Redundant Array of Independent (or Inexpensive) Disks*.

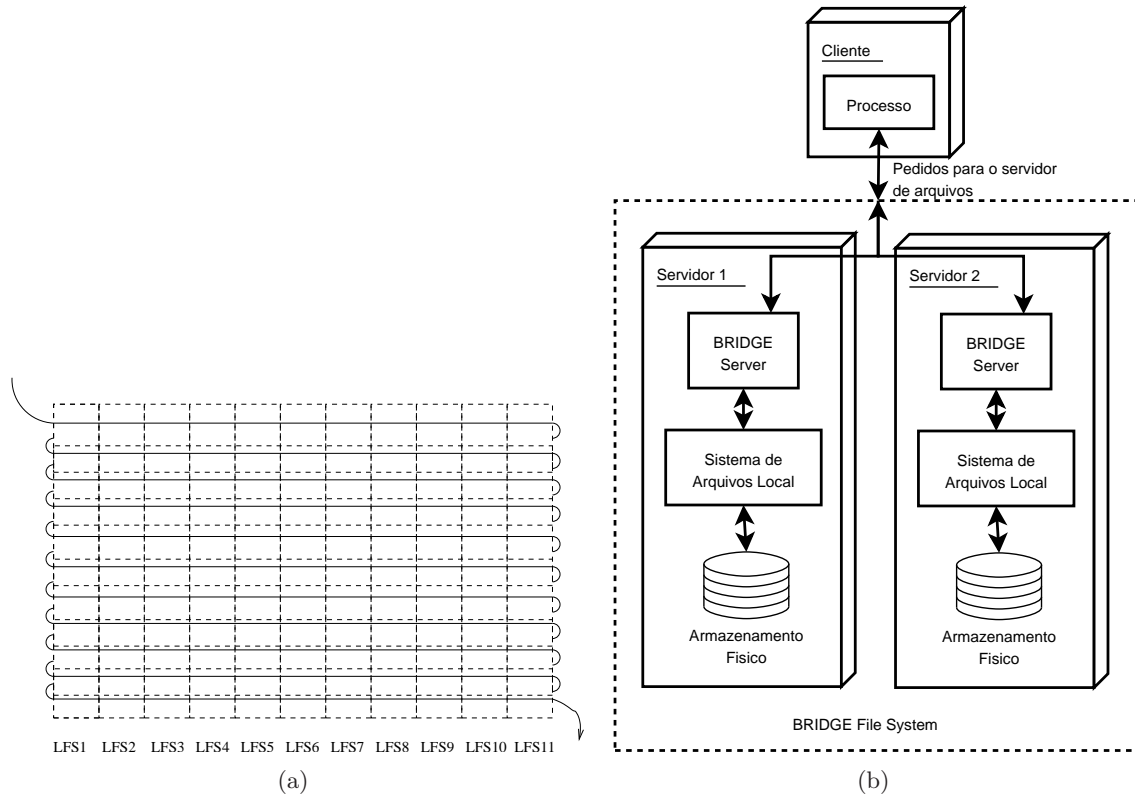


Figura 3.4: (a) Estrutura *Interleaved File* e (b) visão geral do sistema BRIDGE

arquivo e o número do bloco para um nome de arquivo correspondente local e seu bloco. Isso é uma grande vantagem, já que não é necessário ter algum servidor central para perguntar onde tal bloco de um arquivo se encontra, bastando apenas calcular sua localização.

Porém, um dos problemas desse tipo de armazenamento é quando ocorre uma remoção de algum bloco de um arquivo. Todos os blocos subsequentes ao removido deverão ser deslocados. Embora nem todos os sistemas de arquivos da época possuíssem uma operação para remover um bloco do meio do arquivo, uma solução para isso foi realizar uma reorganização *offline* dos blocos, para não interferir no desempenho dos clientes.

### Estrutura do BRIDGE

A estrutura usada no BRIDGE consiste de 3 itens: O BRIDGE Server, junto com um conjunto de ferramentas para manipulação dos dados, o sistema de arquivos local (LFS) de cada nó e o sistema de armazenamento físico, conforme figura 3.4(b).

O BRIDGE Server é a interface entre o sistema de arquivos do BRIDGE com os programas dos usuários. Ele disponibiliza funções como *open*, *read* e *write*, para que o formato *interleaved* fique transparente. Ele também disponibiliza ferramentas específicas para usuários que querem se

utilizar dos recursos paralelos desse sistema de arquivos. Por exemplo, quando  $t$  processos querem escrever  $t$  blocos (um para cada processo), as  $t$  operações serão realizadas com o maior nível de paralelismo possível. Algumas aplicações também podem estar no meio termo, ou seja, usando o paralelismo sem se preocupar com o formato *interleaved file*.

As ferramentas disponíveis para o BRIDGE Server são aquelas usadas em sistemas de arquivos comuns, como *copy*, *sort*, *grep*, etc, que para o usuário parecem simples e comuns, mas no servidor exigem vários tipos de operação, como, por exemplo, comunicação entre o BRIDGE Server e os vários sistemas de arquivos locais dos vários nós, que podem gerar novos processos nos nós a fim de fazer uma busca nos arquivos, por exemplo, devolvendo o resultado para o BRIDGE Server que irá finalizar a computação dos dados e apresentá-los ao usuário.

### Análise Crítica

A idéia usada nesse sistema de arquivos beneficia o desempenho de programas paralelos, pois apresenta um aumento significativo no acesso paralelo comparado com o acesso seqüencial. Problemas existem, como falta de tolerância a falhas e alguns problemas isolados (como a remoção de blocos dos arquivos, que necessita de uma reorganização interna), mas para a aplicação a qual ele foi desenvolvido, esses problemas não são os mais importantes, quando o que se quer é velocidade de acesso em ambientes paralelos.

### 3.2.2 PVFS

Atualmente os aglomerados de PCs têm se tornado cada vez mais populares para aplicações paralelas. Com isso, a demanda por software para esse tipo de plataforma tem crescido muito. Hoje em dia encontra-se todo tipo de software para o ambiente de computação paralela, como sistemas operacionais confiáveis, sistemas de armazenamento de dados local e sistemas de envio de mensagens.

O *Parallel Virtual File System* [CIRT00, Had00] se encaixa na área de sistemas de arquivos paralelo, pois é um sistema de arquivos distribuído desenvolvido para prover alto desempenho e escalabilidade paralela para aglomerados de PCs linux. Em geral, o PVFS promete 4 características:

- Um espaço de nomes consistente para todo o aglomerado;
- Acesso transparente para programas e aplicações já existentes, sem a necessidade de recompilá-los;
- Distribuição física de dados em múltiplos discos e múltiplos nós;
- Alto desempenho no acesso em modo usuário.

Para que um sistema de arquivos paralelo possa ser usado de maneira fácil, ele deve prover um espaço de nomes único em todo o aglomerado e deve ser possível acessá-lo através de utilitários comuns.

Para prover acesso de alto desempenho para os clientes do aglomerado, os dados armazenados no PVFS estão distribuídos entre os vários nós que compõe o aglomerado, assim como o **BRIDGE**

faz, porém usando algoritmos de distribuição diferentes. Cada um desses nós é chamado de *I/O node*.

Dessa forma, para se obter os dados de um determinado arquivo, é necessário acessar várias máquinas, utilizando-se, assim, de vários caminhos pela rede para chegar aos respectivos discos em que estão armazenados. Isso elimina o gargalo da transferência de dados que se tem quando toda a informação está em uma só máquina, distribuindo a carga e aumentando o potencial total da banda para múltiplos clientes.

Usar mecanismos tradicionais de chamadas de sistema para acesso a arquivos pode ser conveniente, mas pode causar uma sobrecarga muito grande para o sistema como um todo, especialmente o *kernel*. Assim, é possível acessar os arquivos do PVFS usando uma API (*Application Programming Interface*) disponibilizada como biblioteca, que contém operações comuns, além de outras específicas do PVFS, que contactam diretamente os servidores, evitando acessos ao *kernel* local. Essas bibliotecas, como a ROMIO MPI-IO [Tha03], podem ser usadas pelas aplicações ou por outras bibliotecas para acesso de alta velocidade ao PVFS.

## Os componentes do PVFS

O **servidor de meta-dados (MGR** na figura 3.5) é um programa que gerencia todos os dados que constituem informações sobre o arquivo (exceto seu conteúdo), como seu nome, sua localização na hierarquia de diretórios, seu dono, seus atributos e como seus dados estão distribuídos entre os vários nós de dados do sistema. Esse programa realiza todas as operações sobre os meta-dados dos arquivos atomicamente, evitando assim ter que implementar esquemas complexos de concorrência, *locks*, consistência, etc, para múltiplos acessos simultâneos.

O **servidor de dados (ION** na figura 3.5) gerencia o armazenamento do conteúdo dos arquivos, bem como a recuperação dos mesmos, nos discos locais conectados nos nós. Esse servidor grava os dados dos arquivos do PVFS em um sistema de arquivos local, através de chamadas a funções tradicionais, como *read()*, *write()* e *mmap()*, para acessá-los. Isso significa que pode-se usar qualquer tipo de sistema de arquivos local, como Ext2, Ext3 ou Reiser [vH03], por exemplo. Adicionalmente é possível usar suporte a RAID para que cada nó possua tolerância a falhas de disco de forma transparente e confiável para todo o sistema.

Os servidores do PVFS não possuem estado, da mesma forma que o NFS, o que simplifica sua implementação, que não considera casos como quando um cliente se desconecta da rede sem aviso prévio. Isso pode gerar problemas de consistência, pois o servidor pode não conter a versão mais recente do arquivo (caso o cliente possuísse um cache sujo), ou algum arquivo pode ficar bloqueado para escrita.

A **API nativa do PVFS** possibilita acesso em modo usuário aos servidores do PVFS. Esta biblioteca, chamada de *libpvfs*, cuida das operações necessárias para mover dados entre os clientes e servidores, mantendo-as transparentes para o usuário. Para operações que necessitam de meta-

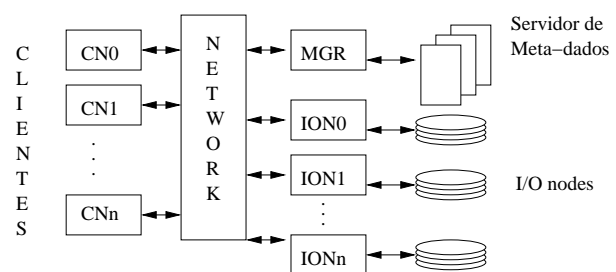


Figura 3.5: Visão geral do sistema PVFS

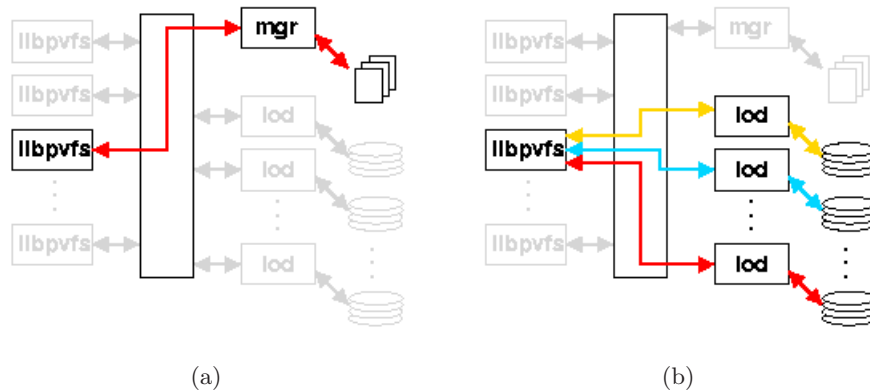


Figura 3.6: Clientes acessando o PVFS

dados, a biblioteca se comunica com o servidor de meta-dados, conforme figura 3.6(a). Para acesso aos dados dos arquivos, o servidor de meta-dados é deixado de lado e os servidores de dados são acessados diretamente, conforme figura 3.6(b). Essa é a chave para se obter um alto desempenho agregado no acesso aos dados.

O suporte no *kernel* do linux para o PVFS provê as funcionalidades necessárias para se usar o comando *mount* nos clientes. Isso permite acesso aos arquivos do PVFS sem necessidade de alteração das aplicações ou programas já existentes. Esse suporte não é necessário para se usar o PVFS, mas ele traz uma enorme conveniência para a interatividade com o sistema. Para isso, é necessário instalar um módulo no *kernel* do linux (existe um patch para carregá-lo diretamente no *kernel*) e um programa chamado (*pvfsd*) que se encarrega de buscar os dados para as aplicações. Ele se utiliza da biblioteca *libpvfs* para realizar essas operações. A figura 3.7 mostra como o fluxo de dados passa por esses componentes.

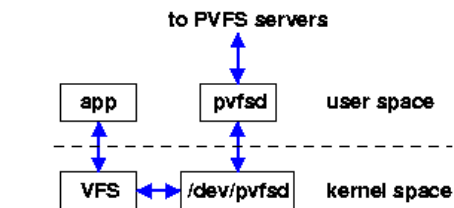


Figura 3.7: Fluxo de dados pelo *kernel*

Além disso, existe a implementação da interface ROMIO MPI-IO para o PVFS, possibilitando que aplicações paralelas se utilizem do PVFS sem precisar passar pelo *kernel*, além de poderem usar outras funções específicas desse sistema que possibilitam um ajuste fino no acesso e armazenamento dos arquivos.

### Análise Crítica

O PVFS é um sistema de arquivos distribuído e paralelo que se preocupa em diminuir o gargalo provocado pelo tráfego de dados, seja pela rede, seja pela velocidade do armazenamento físico, usando a mesma técnica de distribuição de dados encontrada no BRIDGE.

Alguns problemas existentes são quanto à segurança no acesso dos dados, já que se o cliente souber onde os dados estão, basta pedi-los para os nós de dados que eles responderão, sem se

preocupar com nenhum tipo de validação de permissão de acesso. Quem cuida da permissão é o servidor de meta-dados, sendo que esse mecanismo não é muito sofisticado. Outro problema existente é quando o servidor de meta-dados fica indisponível. Somente os arquivos já abertos continuarão sendo acessados, até serem fechados. Todos os outros arquivos do sistema não poderão ser acessados. Além disso, o servidor de meta-dados pode representar um gargalo no sistema, já que ele é único.

É um sistema de arquivos paralelo que já apresenta bons resultados, mesmo tendo problemas visíveis. Para aplicações paralelas e confiáveis, em uma rede privativa e fechada, ele pode ser usado sem grandes problemas de segurança.

### 3.2.3 PVFS2

O PVFS2 é uma reimplementação das melhores características da primeira versão do PVFS, usando uma nova arquitetura para torná-lo mais flexível. Isso possibilitou a implementação de novas características, técnicas e inovações que foram sendo discutidas e requisitadas durante as correções de defeitos da primeira versão.

As novidades [RLC02, Tea03] implementadas (ou ainda a serem implementadas) no PVFS2 e sua nova arquitetura estão detalhadas nas próximas seções. No capítulo 5 há mais detalhes sobre o desempenho do PVFS2.

#### Novas características

Suporte modular para múltiplos protocolos de rede e armazenamento

O PVFS1 foi desenvolvido com a idéia de que seus dados seriam acessados via soquete e armazenados em sistemas de arquivos locais.

Analisando os aglomerados de computadores existentes hoje, nota-se que existem muitas tecnologias diferentes em cada um deles, sendo que algumas são mais populares que outras. O mesmo ocorre com os sistemas de armazenamento de dados locais.

Dessa forma, o PVFS2 foi projetado usando o **BMI** [CRIW05] (*Buffered Messaging Interface*) como interface de acesso à rede e o **Trove**<sup>6</sup> como interface de acesso ao sistema de armazenamento físico. O objetivo é abstrair do projeto os detalhes do mecanismo de transmissão e armazenamento. Isso permite que um desenvolvedor personalize módulos específicos para seu ambiente, sem ter que alterar o núcleo do PVFS2.

Acesso a dados estruturados não-contínuos

Muitas aplicações científicas possuem estruturas de dados complexas que nem sempre podem ser armazenadas de forma contínua, pois isto certamente impacta o desempenho da aplicação como um todo.

---

<sup>6</sup>O Trove é uma interface desenvolvida pela equipe do PVFS2, sendo que até agosto de 2005 não havia um documento público descrevendo-a, a não ser pelo seu próprio código-fonte.

Assim como o PVFS1, o PVFS2 dá suporte ao ROMIO MPI-IO, que permite ao desenvolvedor da aplicação descrever seus dados usando tipos MPI, melhorando o desempenho na leitura dos dados persistidos.

#### Distribuição de dados flexível

No PVFS1 os dados são distribuídos entre os servidores de dados usando o algoritmo round-robin, isto é, um arquivo é dividido em blocos de igual tamanho e cada bloco subsequente é armazenado no próximo servidor, sendo que ao chegar no último servidor volta-se para o primeiro, até que todos os blocos estejam armazenados. Isso é muito eficiente como uma técnica genérica para quando não se conhece o padrão de acesso ao arquivo.

Porém, em geral sabe-se qual é o padrão de acesso de um arquivo e isso poderia ser usado para otimizar o acesso a ele. O PVFS2 permite que se informe esse padrão de acesso e decide qual a melhor forma de armazenar os dados para máxima eficiência, podendo até mesmo utilizar-se de redundância.

#### Servidores de meta-dados distribuídos

No PVFS1 o servidor de meta-dados (que armazena informações sobre estrutura de diretórios, data de criação de arquivos, etc) é centralizado, podendo representar um gargalo maior conforme o número de clientes aumenta.

O PVFS2 permite ter mais de um servidor de meta-dados, que pode ou não ser um subconjunto dos servidores de dados.

#### Suporte explícito à concorrência

Um sistema de arquivos paralelo deve ser extremamente eficiente quanto a prover dados para vários clientes simultaneamente.

O projeto do servidor e cliente PVFS2 foi baseado em uma máquina de estados que está intimamente ligada a um componente de monitoramento da finalização das operação em todos os sistemas envolvidos. Isto é, permite-se que se realize acesso sem bloqueios a todos os tipos de dispositivos. Isso dá suporte a operações assíncronas nativamente, facilitando a implementação do ROMIO MPI-IO.

#### Semânticas de consistência ajustáveis

Muitos sistemas de arquivos distribuídos implementam as semânticas POSIX, que são muito estritas. O NFS, por exemplo, não implementa essas semânticas, pois não garante que o cache de seus clientes estejam coerentes o tempo todo.

Por existirem vantagens e desvantagens em cada tipo de semântica, o PVFS2 permite que o usuário opte por uma semântica mais estrita, para permitir a implementação do ROMIO MPI-IO, ou mais relaxada, permitindo um uso mais amplo.

#### Mapeamento flexível de referências de arquivos para servidores

É possível reconfigurar os servidores de meta-dados para escolher onde armazenar um determinado arquivo. Isso é muito útil na administração do sistema de arquivos, para, por exemplo,

remover um servidor ou adicionar outro. Isso também pode ser feito sem a necessidade de se desativar o sistema de arquivos.

#### Redundância de dados e meta-dados

O PVFS1 possui um grande problema com relação à tolerância a falhas: caso um servidor saia da rede, perde-se o acesso aos seus dados. Pode-se utilizar um sistema RAID de disco para evitar a perda dos dados, mas isto não garante tolerância à falhas.

Está sendo estudado para versões futuras do PVFS2 um sistema de redundância relaxada dos dados. A idéia é realizar uma cópia dos dados e meta-dados de um servidor em outro, utilizando-se de uma operação explícita ao cliente. Isto significa que o cliente PVFS2 teria que realizar essa cópia.

A desvantagem nisso está em realizar operações de forma atômica e em encontrar formas de se evitar uma grande perda de desempenho. A vantagem é que a operação seria otimizada, ao criar as informações redundantes em paralelo.

## Arquitetura do PVFS2

### Servidores

No PVFS1 cada servidor tem papel distinto: servir meta-dados ou somente dados. Além disso, o servidor de meta-dados é único.

No PVFS2, cada servidor pode atuar tanto como servidor de meta-dados como também de dados. A definição do papel que cada um vai representar está no arquivo de configurações, lido durante a inicialização. Além disso, pode-se ter múltiplos servidores de meta-dados.

### Redes

Como já mencionado, utilizando-se do BMI é possível que o PVFS2 se comunique por TCP/IP, InfiniBand<sup>7</sup>, Myricom<sup>8</sup> ou qualquer outro protocolo de rede que venha a ser implementado.

### Interfaces

Os clientes podem acessar o PVFS2 através de duas interfaces: UNIX nativo, representado pelo cliente do sistema operacional, ou ROMIO MPI-IO. Ambas as formas seguem o mesmo perfil que foi desenvolvido para o PVFS1.

### Interações cliente-servidor

Durante o primeiro acesso ao PVFS2, os clientes acessam algum dos servidores para obter informações sobre a configuração do sistema de arquivos. Esse processo ocorre de forma similar ao NFS: para abrir um arquivo, o cliente pede ao servidor um *handle*. Tendo um *handle*, o cliente pode acessar qualquer trecho do arquivo, desde que tenha permissão de acesso. Quando esse *handle* expirar, o servidor avisará o cliente no momento do acesso.

---

<sup>7</sup><http://www.infinibandta.org/about/>

<sup>8</sup><http://www.myricom.com/>

Esse tipo de estratégia permite que um processo possa passar seu *handle* a outro processo, que evita uma nova busca pelo arquivo junto ao servidor. Como os clientes e servidores não possuem estado, uma desvantagem é que se um arquivo é removido, o cliente que tiver o *handle* ainda poderá acessá-lo por um tempo, até expirar. Esse tipo de problema também ocorre em sistemas de arquivos locais.

#### Consistências do ponto de vista do cliente

O PVFS2 permite que vários clientes realizem escritas simultâneas em regiões não-coincidentes dos arquivos, até mesmo em regiões não-contínuas, de forma atômica. Isso possibilita paralelizar a escrita sem correr o risco de se gerar inconsistências entre servidor e clientes.

Quanto à consistência do cache, o PVFS2 permite colocar no cache do cliente a estrutura de diretórios do servidor de meta-dados. Isso pode gerar inconsistências temporárias, pois caso haja alguma mudança em tal estrutura, o cliente ficará desatualizado por um certo tempo (configurável).

#### Consistência do sistema de arquivos

Ao realizar alterações na estrutura de diretórios do PVFS2, o sistema de arquivos é bloqueado enquanto essa tarefa é realizada. Foi notado que esse tipo de tarefa não representa um gargalo na maioria das aplicações, mesmo em larga escala.

Porém, esses bloqueios não ocorrem em todas as operações. Por exemplo, para criar um arquivo deve-se:

1. criar uma entrada no diretório;
2. criar um objeto de meta-dados;
3. apontar a entrada no diretório para o objeto de meta-dados;
4. criar um conjunto de objetos de dados para o novo arquivo e apontá-los aos objeto de meta-dados.

Cada uma dessas operações é realizada atomicamente, mas o conjunto delas não. Isso é um problema para o PVFS2, caso a execução dessas tarefas seja interrompida.

### Análise Crítica

O PVFS2 realmente evoluiu muito em comparação ao PVFS original. As novas características que estão sendo adotadas permitem que ele seja cada vez mais utilizado, o que ajuda os desenvolvedores a entender a real necessidade que os pesquisadores têm de um sistema de arquivos paralelo para suas aplicações.

A mudança na forma como o código foi implementado facilita sua evolução, atraindo desenvolvedores de plataformas específicas a criar módulos robustos para o PVFS2, que permitem usar esse SAP em cada vez mais aglomerados de computadores.

Conforme nossos testes da versão 1.0.1 (descritos no capítulo 5), muitos dos itens citados nesta resenha não foram implementados, ou ainda estão em testes e portanto não puderam ser analisados,

porém a facilidade na instalação e a estabilidade e desempenho alcançados com relação à sua primeira versão mostram que essa área de pesquisa continua em constante evolução.

### 3.2.4 NFSP

Concebido e mantido pelo *Laboratoire d'Informatique et Distribution* do *Institut d'Informatique et Mathématiques Appliquées de Grenoble* (IMAG), França, o *NFSP* [LD01] é a união da estabilidade, confiança e compatibilidade do NFS com o alto desempenho do PVFS. A idéia é aplicar todos os objetivos do PVFS através do uso do seu próprio código dentro da implementação do NFS.

Assim como no PVFS, o objetivo desse sistema de arquivos paralelo é oferecer alto desempenho em aglomerados, de forma transparente para as aplicações clientes, que enxergariam os vários servidores como sendo um sistema de arquivos único e consistente. Ele usa o mesmo esquema de servidores do PVFS, ou seja, um servidor de meta-dados (contendo informações sobre data de modificação, permissões de acesso, etc) e vários servidores de dados dedicados (onde os dados dos arquivos a serem armazenados estariam distribuídos).

A principal diferença com relação ao PVFS (embora use suas idéias e grande parte de seu código) é o protocolo de comunicação entre servidores e clientes, que é o tão comum e disseminado NFS. Dessa forma, é possível utilizar qualquer cliente NFS para acessar e manipular as informações contidas nos servidores, diminuindo de forma considerável a manutenção e administração dos clientes.

Esses foram os fatores que incentivaram o laboratório de computação distribuída do IMAG a desenvolver o NFSP. Nele havia um aglomerado de 225 máquinas onde cada uma delas possuía 11GB de espaço livre em disco. Foram pesquisados vários sistemas de arquivos distribuídos modernos, como o AFS, CODA, xFS, NFS, PVFS, para tornarem todo esse espaço em disco espalhado entre as máquinas em um sistema de arquivos único e distribuído. O que se procurava era um sistema robusto e fácil de manter em uma rede segura. Alguns dos sistemas pesquisados eram muito complexos para se instalar e administrar (como o PVFS), outros não estavam disponíveis para linux (como o xFS) e outros ainda disponibilizavam recursos desnecessários ao ambiente desejado (como o AFS). Como nenhum dos sistemas apresentou as características desejadas, decidiu-se desenvolver um sistema que as atendesse.

#### Características

O protocolo usado no NFSP é o NFSv2 (RFC1094), construído sobre os mecanismos do SunRPC, usando o formato XDR para dados. Com isso, qualquer cliente NFS pode acessar o NFSP como se estivesse acessando um servidor NFS comum.

No lado do servidor há dois processos rodando, o **rpc.mount**, que implementa o protocolo para montar o sistema de arquivos, e o **rpc.nfsp** que é o responsável por tratar as requisições dos clientes e transmitir as informações solicitadas. No lado do cliente basta ter um cliente compatível com o NFS para montar o sistema de arquivos.

O servidor do NFSP foi criado em modo-*kernel* e modo-usuário. Como o modo-*kernel* é muito “intrusivo”, pois seu funcionamento depende muito da versão do *kernel* que se possui, além de complicar a instalação, seu desenvolvimento foi abandonado. Segundo [LD01], embora o servidor

em modo-usuário sofra com um pouco mais de sobrecarga (os dados precisam passar do espaço de usuário para o espaço de *kernel* antes de trafegar pela rede, e vice-versa), isso é aceitável dada a simplicidade de instalação que se ganha, além do que o sistema operacional não congelará caso ocorra algum defeito por parte do servidor NFSP.

Porém, a segurança na transmissão dos dados é muito deficiente, o que compromete a confiança tanto dos clientes como do servidor. Ele possui as mesmas características de segurança que foram apresentadas para o NFS e o PVFS, que consiste em verificar as permissões de acesso aos arquivos de forma não segura. Mas, assim como no PVFS, esse é um sistema de arquivos desenvolvido para ser acessado em uma rede segura.

Existe uma implementação inicial de redundância dos dados armazenados, através da adoção de VIOD (*Virtual IOD*) no lugar do IOD (*IO Deamon*). Um IOD é um nó do sistema de arquivos usado para armazenar dados. Um VIOD representa vários IODs que possuem os mesmos dados, ou seja, possuem cópias entre si dos dados que armazenam. O intuito não é apenas criar alta disponibilidade, mas também proporcionar melhor acesso concorrente, ao distribuir a carga entre os IODs do mesmo VIOD.

### Análise Crítica

O NFSP partiu da idéia do uso de vários sistemas de arquivos locais que estariam distribuídos em uma rede de computadores, podendo acessá-los de uma forma simples, usando clientes já prontos, diminuindo assim a tarefa de administração e manutenção destes. Para isso, foi criado um servidor NFS que possui características do PVFS, ou seja, é um sistema de arquivos paralelo que distribui os dados dos arquivos entre vários servidores, proporcionando ótimo desempenho no acesso concorrente aos mesmos, além de fornecer uma interface já bem conhecida para comunicação com seus clientes.

Pensando-se em melhorar esse SAD, foi desenvolvida uma forma de gerar alta disponibilidade e alta escalabilidade para acesso concorrente aos arquivos ao replicá-los entre vários servidores através de *virtual IO nodes* (VIOD).

### 3.2.5 CEFT-PVFS

O CEFT-PVFS (*Cost-Effective, Fault-Tolerant Parallel Virtual File System*) [ZJQ<sup>+</sup>03] é uma extensão do PVFS convencional que explora ao máximo o paralelismo, através da replicação dos servidores do aglomerado.

### Características

Existem dois grupos de servidores para o CEFT-PVFS: um primário e um secundário. Ambos possuem a mesma quantidade de nós, sendo que cada um tem seu próprio servidor de meta-dados. Os dados contidos no grupo primário são encontrados também no grupo secundário. A idéia é fornecer vários caminhos para a busca do mesmo dado, dobrando a capacidade do sistema de atender muitas requisições.

Assim, para pedidos de leitura os clientes podem acessar tanto um grupo como o outro, pois os dados armazenados em ambos são os mesmos. Dessa forma, caso um servidor esteja sobrecarregado,

ou tenha caído, o cliente acessa o outro, causando uma distribuição de carga ou uma tolerância à falha, respectivamente. Ambos os casos são totalmente transparentes para o cliente.

Um ganho imediato no CEFT-PVFS com relação ao PVFS é a possibilidade de atender até o dobro do número de clientes (veja a seção 4.3), além de permitir que haja o dobro de dados em memória cache, reduzindo o número de acessos ao disco. Obviamente, esses ganhos só serão aproveitados enquanto houver banda suficiente para transmissão dos dados.

Com relação à escrita dos dados, todos os clientes mandam seus pedidos para o grupo de servidores primário. Este se encarrega de atualizar seus dados e retornar ao cliente um código de erro ou sucesso. Existem quatro formas de replicação de dados, que são a combinação dos seguintes itens:

- Atualização síncrona ou assíncrona (através de um processo em *background*);
- Pedidos de replicação vindos do grupo principal ou dos próprios clientes, que contatam diretamente os dois grupos.

A decisão de qual política utilizar fica a cargo do usuário durante a instalação do sistema de arquivos. Alguns testes de desempenho do CEFT-PVFS estão na seção 4.3.

### Análise Crítica

O CEFT-PVFS mostra, assim como o NFSP, que o PVFS é um bom ponto de partida para o desenvolvimento de sistemas de arquivos paralelos especializados em resolver determinados problemas. Além disso, mostra também que as deficiências encontradas no PVFS podem ser superadas, o que o torna cada vez mais interessante para ser usado em aglomerados de computadores.

Outro ponto em comum entre CEFT-PVFS e o NFSP é o armazenamento redundante dos dados em servidores distintos, o que supre, até certo ponto, a falta de distribuição de carga e tolerância a falhas do PVFS original.

### 3.2.6 GFS

O *Google File System* [GGL03] é um sistema de arquivos distribuído, altamente escalável, desenvolvido para lidar com aplicações que acessam dados intensivamente. Ele provê tolerância a falhas, funciona em máquinas de baixo custo e possui alto desempenho para um grande número de clientes.

### Características

O objetivo do GFS é fornecer um sistema de arquivos para os serviços disponibilizados pelo Google<sup>9</sup>, cuja demanda por acesso a dados cresce rapidamente. Ele compartilha muitas das características dos sistemas de arquivos já comentados até aqui, como desempenho, escalabilidade, tolerância a falhas.

---

<sup>9</sup><http://www.google.com>

A partir da observação do comportamento das aplicações, foram definidas algumas premissas para servirem de guia durante seu desenvolvimento:

- O sistema é composto de peças de baixo custo que podem falhar constantemente. Assim, deve-se monitorar e detectar os mais variados problemas, recuperando-se de forma automática (se possível) o mais rápido possível;
- Alto desempenho para tratar arquivos da casa dos GB é mais importante que para tratar arquivos pequenos. Haverá alguns milhões de arquivos no sistema;
- Para leitura, existem dois tipos de acesso muito comuns: leitura seqüencial de muitos dados (1MB ou mais) e leitura aleatória de poucos KBs, dentro de uma mesma região do arquivo. As aplicações que se preocupam com desempenho procuram realizar este último tipo de leitura de forma seqüencial, ao ordenar os pedidos antes de realizá-los;
- Para escrita, o caso mais comum é a adição de muitos dados no final dos arquivos. Pequenas escritas em posições arbitrárias devem ser possíveis, mas não necessitam eficiência;
- O sistema necessita implementar eficientemente semânticas bem definidas para que múltiplos clientes adicionem dados no mesmo arquivo concorrentemente. Atomicidade com o mínimo de sobrecarga para sincronização é essencial;
- Alta banda passante (isto é, capacidade de transmitir dados o mais rápido possível) é mais importante que baixa latência.

A API do GFS é bem familiar, embora não implemente o POSIX. Os arquivos são organizados em diretórios e identificados por caminhos (*pathnames*). Ela dá suporte a operações para criar, apagar, abrir, fechar, ler e escrever arquivos. Além disso, existem operações específicas, criadas para aumentar o desempenho de tarefas comuns requisitadas pelos clientes, que são: criar uma cópia de um arquivo ou árvore de diretório a baixo custo (*snapshot*) e adicionar dados ao final do arquivo (*record append*) atomicamente para múltiplos clientes.

## Arquitetura

Um aglomerado GFS é composto por um **servidor mestre** (*master*) e múltiplos **servidores de dados** (*chunk servers*). Cada qual é executado como um processo de usuário sobre o sistema operacional Linux. É possível executar cliente e servidor na mesma máquina, desde que haja recursos disponíveis para isso.

Os arquivos armazenados são divididos em pedaços (*chunks*) de tamanho fixo. Cada um desses pedaços é identificado por um *chunk handler* de 64 bits globalmente único, que é associado no momento de sua criação pelo servidor mestre. Esses *chunks* são armazenados como arquivos e são replicados em múltiplos servidores (por padrão em três), sendo que os usuários podem designar diferentes tipos de replicação para diferentes regiões do arquivo.

Tanto clientes como servidores não se utilizam de cache de arquivo, dado que o volume de dados trafegados é muito grande, o que necessitaria de muita memória. Isso elimina problemas

de coerência de cache e outras sobrecargas, simplificando o cliente. Porém, os clientes mantêm meta-dados em cache e os servidores se utilizam do cache do sistema operacional para os arquivos locais.

#### Servidor mestre

O servidor mestre gerencia e mantém informações de meta-dados, como espaço de nomes do arquivo e do *chunk*, mapeamento de arquivos para os *chunks* e a localização de cada réplica. Todos esses dados são mantidos em memória. Somente o espaço de nomes e o mapeamento de arquivo para *chunk* são persistidos em forma de log no sistema de arquivos, para evitar inconsistências em caso de queda.

A localização dos *chunks* não é persistida, mas carregada em memória durante a inicialização, onde o servidor mestre pergunta aos servidores de dados do aglomerado sobre a localização de cada um. Realmente, não é necessário persistir esses dados, já que os servidores de dados podem deixar de funcionar a qualquer momento, isto é, tal localização pode variar muito, sendo necessário manter uma comunicação periódica com eles para manter o estado do sistema atualizado (chamado de *HeartBeat*).

Outras tarefas do servidor mestre são coleta dos *chunks* que se tornaram órfãos, migração deles entre servidores de dados, fornecimento de identificadores para criação de novos *chunks*, entre outras.

O uso de um só servidor mestre simplifica a arquitetura do sistema e permite um gerenciamento mais sofisticados dos *chunks*, como posicionamento e replicação a partir do conhecimento global da situação do sistema de arquivos. Esse servidor é replicado, para o caso de tolerância a falhas, sem levar em conta alta disponibilidade.

Dessa forma, deve-se tomar cuidado para não se criar um gargalo. Para isso, assim como ocorre no PVFS, o servidor mestre do GFS somente é acessado pelos clientes para se obter informações sobre onde encontrar um pedaço do arquivo que se quer acessar, isto é, quais servidores de dados o replicam. Com tal informação, os dados são obtidos diretamente dos servidores de dados, sem necessidade de novos acessos ao servidor mestre.

Ao manter todos os meta-dados em memória, cria-se uma limitação no sistema: a quantidade de memória disponível. Porém, para cada *chunk* de 64MB se utiliza apenas 64 bytes de memória para meta-dados. Como a maioria dos arquivos é grande, muitos desses *chunks* estarão cheios e somente o último estará parcialmente preenchido. Além disso, o espaço de nomes utiliza somente 64 bytes para cada arquivo, pois os seus nomes são armazenados compactados usando somente o prefixo.

#### Servidor de dados

No servidor de dados são armazenados os pedaços dos arquivos (*chunks*). Cada *chunk* possui um tamanho fixo de 64MB, muito maior que os tamanhos de blocos dos sistemas de arquivos comuns. Isso possui vantagens importantes, dadas as características do GFS:

- Reduz o número de interações com o servidor mestre, pois escritas e leituras no mesmo *chunk* necessitam de apenas uma consulta inicial;

- Dentro de um *chunk* grande o cliente pode realizar várias operações, reduzindo a sobrecarga da rede ao se manter uma conexão aberta com o mesmo servidor de dados por muito mais tempo;
- Reduz a quantidade de meta-dados armazenados no servidor mestre.

Uma característica importante no GFS, e ao mesmo tempo interessante, é a forma como os dados são enviados para os servidores. A fim de se utilizar a rede de forma extremamente eficiente, os dados são enviados linearmente e simultaneamente ao longo da cadeia de servidores de dados, utilizando ao máximo a largura de banda disponível entre duas máquinas, reduzindo o tempo de comunicação total.

Por exemplo, se um cliente precisa atualizar *chunks* replicados nos servidores S1, S2, S3 e S4, ele envia os dados primeiramente para o servidor mais próximo<sup>10</sup> (na intenção de reduzir a latência), digamos S1. Este, ao começar a receber as informações já as repassa para o próximo servidor, digamos S2. S2 então copia os dados para algum servidor envolvido na transmissão que ainda não recebeu os dados e que está próximo, digamos S3. Este, por sua vez, manda para S4.

Quando S1 terminar de enviar seus dados, S2, S3 e S4 estarão também no final de suas tarefas, quase que simultaneamente. Ou seja, conforme os servidores vão recebendo os dados, já vão repassando-os para os outros servidores envolvidos, seguindo a idéia de *pipeline*, ao mesmo tempo que vão gravando os dados nos respectivos sistemas de arquivos locais.

Essa técnica se utiliza bem de uma característica típica de redes conectadas por *switches*: a banda de dados de cada máquina é independente do tráfego entre outras máquinas. Assim, enquanto o cliente manda dados para S1 na velocidade máxima da rede, S1 manda para S2, S2 manda para S3 e S3 manda para S4, todos usando o máximo da velocidade disponibilizada entre cada uma das máquinas.

Além disso, essa técnica utiliza toda a banda de entrada e saída disponível na rede para cada um dos servidores envolvidos, pois enquanto a banda de entrada está no seu limite para receber os dados, simultaneamente a máquina envia uma cópia dos dados para o próximo servidor, utilizando o máximo da banda de saída.

Transformando em números, em uma rede de 100Mbit/s, ao transmitir 120MB, a técnica de *pipeline* fará com que toda a operação demore por volta de 10 segundos (desconsiderando tempo de processamento e gravação). Por outro lado, se o cliente enviar uma cópia dos 120MB para S1, S2, S3 e S4, paralelamente ou serialmente, levará por volta de 40 segundos, pois são 480MB saindo de uma só máquina limitada a 100Mbit/s. Note que em ambos os casos tivemos um tráfego total de 480MB, sendo que no primeiro caso a banda total passante<sup>11</sup> chegou a 48MB/s, enquanto que no segundo chegou a apenas 12MB/s.

Porém, caso haja mais de um cliente enviando dados concorrentemente para os servidores, o desempenho geral do sistema sofrerá uma queda proporcional ao número de clientes, pois

---

<sup>10</sup>O cálculo da distância é realizado através dos números IP de cada máquina, que foram distribuídos de forma a facilitar essa tarefa.

<sup>11</sup>Quantidade total de dados dividido pelo tempo usado para a transferência completa.

embora a banda de saída de cada um dos clientes não chegue ao seu limite, a banda de entrada e saída dos servidores chegará, se tornando, assim, um gargalo.

Por exemplo, se o cliente C1 envia dados para o servidor S1 e o cliente C2 envia dados para o servidor S2 (supondo que C2 está mais próximo de S2 do que de S1), S1 e S2 irão replicar as informações vindas de C1 e C2 simultaneamente. Embora os 2 clientes estejam mandando dados a uma velocidade agregada máxima de 200Mbit/s, S1 receberá dados de S2 e vice-versa ao mesmo tempo em que isso ocorre, o que faz essa velocidade agregada não passar de 100Mbit/s.

### Leases

Para manter consistência e sincronização entre as réplicas dos *chunks* durante escritas e alterações de meta-dados, usa-se o conceito de *leases*. Assim, quando um cliente for escrever ele:

- Pede ao servidor mestre qual o servidor de dados que possui um *lease* para um dado *chunk*, juntamente com suas réplicas;
- Se não existir, o servidor mestre define um *lease*, associa a um servidor de dados que possui uma das réplicas, que será a primária a partir de então;
- Com os servidores de dados listados, o cliente pode então passar os dados a serem gravados para todas as réplicas e fazer o pedido de escrita para o primário;
- Este por sua vez realiza a escrita em si e passa o pedido para os outros servidores que detêm tal réplica. Os dados são gravados todos na mesma ordem em cada um deles, seguindo a ordem imposta pelo cliente;
- As réplicas secundárias enviam uma notificação à primária, avisando se tudo deu certo ou não. Em caso de falha, o estado do arquivo fica inconsistente. De qualquer forma, o primário avisa o resultado para o cliente.

O *lease* tem um tempo de expiração de 60 segundos, que pode ser renovado pela própria réplica primária, indefinidamente. Isso ajuda a minimizar a sobrecarga de gerenciamento do servidor mestre. Além disso, caso o servidor da réplica primária não responda, o servidor mestre pode dar outro *lease* para o mesmo *chunk* assim que o antigo expirar. Um caso que pode ocorrer também é do servidor pedir o *lease* de volta antes dele expirar.

### Operações específicas: *atomic record append* e *snapshot*

Foi observado que a maioria das escritas no GFS ocorriam somente no final dos arquivos (em geral de forma concorrente), adicionando mais dados, ao invés de alterar dados já existentes. Dessa forma, foi criada uma operação na API do GFS para realizar essa tarefa de forma rápida e simples para os clientes.

Essa operação é chamada de *atomic record append* e é responsável por escrever dados no final do arquivo, de forma concorrente, informando se a sua execução foi realizada com sucesso ou não. Por sucesso entende-se que todos os dados foram gravados com sucesso, assim como

foram replicados a todos os servidores de dados envolvidos. Falha indica que algum dado não foi escrito corretamente em algum servidor. Nesse caso, as alterações realizadas em alguns servidores não são desfeitas, mas o sistema garante que novos pedidos de escrita não serão influenciados por essa falha, isto é, após a falha as cópias terão o mesmo tamanho, mas não necessariamente os mesmos dados. Assim, as aplicações devem levar isso em conta e devem estar prontas para esse tipo de problema.

Os *snapshots* são operações muito utilizadas para se criar versões de um conjunto muito grande de dados. Assim, o GFS a implementa na sua API para facilitar o desenvolvimento das aplicações e melhorar o desempenho delas. Um grande uso é para a realização de operações baseadas em transação, onde cria-se uma cópia dos dados originais para ser alterada a vontade pela aplicação. Ao terminar, essa cópia pode ser apagada (*roll back*) ou gravada em cima do original (*commit*).

Essa operação é controlada pelo servidor mestre que, ao receber um pedido de *snapshot*, invalida todos os *leases* que se referenciam aos *chunks* a serem copiados antes de realizá-la, para impedir que clientes alterem os dados durante a realização da operação. Após isso, o servidor cria a cópia, que aponta para os mesmos *chunks* que os dados originais (por isso a alta velocidade dessa operação).

Quando um cliente pede por um *lease* para algum desses *chunks* copiados, digamos *C*, o servidor mestre irá primeiramente avisar aos servidores de dados para criar o *chunk C'*, cópia de *C*. Essa operação não necessitará trafegar todo o conteúdo do *chunk* via rede, já que todos os servidores de dados que possuem o seu espelhamento podem copiá-lo localmente. Feito isso, o servidor mestre entrega, então, o *lease* ao cliente, que passa a acessar os dados de *C'* da forma usual.

### Análise Crítica

O Google File System é um sistema de arquivos desenvolvido para uma finalidade clara e bem definida, para servir a aplicações específicas e de padrão de acesso conhecido. Isso o torna extremamente eficiente nos aspectos mais comuns em que vai ser requisitado, porém cria problemas em outros casos.

O fato das aplicações que o acessam terem que levar em conta as possíveis falhas na gravação dos dados torna esse sistema de arquivos pouco indicado para usuários comuns (caso ele fosse publicamente distribuído ou comercializado), pois os serviços oferecidos não são totalmente transparentes para as aplicações que o utilizam.

### 3.2.7 SVA

O *StorageTek Shared Virtual Array* [GST02] é uma plataforma de armazenamento de dados desenvolvida para prover alto desempenho em ambientes distribuídos, além de alta escalabilidade e segurança. A maioria dos serviços disponibilizados por esta plataforma é feita através de equipamentos mecânicos ou circuitos, ao contrário dos outros sistemas de arquivos analisados até agora.

Embora seja uma solução comercial <sup>12</sup> e de preço elevado, ela proporciona todas as características que o PVFS2 pretende implementar, como **RAID** entre servidores, visão lógica do volume (deixando assim transparente o acesso aos vários nós que armazenam os dados), possibilidade de aumentar o espaço em disco total dinamicamente (isto é, sem necessidade de desativar o serviço), simplesmente adicionando um novo servidor ao aglomerado, como nó de dados, etc.

### Análise Crítica

Por ser um sistema comercial e de alto custo, fica difícil realizar uma análise de um sistema como esse. Além disso, o fabricante disponibiliza acesso somente a documentações comerciais e técnicas que mostram vantagens do equipamento, mas não limitações ou problemas. Dados técnicos sobre seu funcionamento e algoritmos implementados também não estão disponíveis.

O objetivo desse sistema neste estudo é o de mostrar que existe interesse comercial em se resolver o problema de desempenho que a maioria dos sistemas de arquivos possui. Existem empresas grandes envolvidas e, como pode-se ver por essa simples descrição, elas estão propondo soluções funcionalmente semelhantes às implementações acadêmicas.

### 3.3 Análise Comparativa

O desenvolvimento nessa área continua intenso e aumentando significativamente. Agora que os aglomerados de PCs estão se tornando comuns, a necessidade de sistemas de arquivos que se utilizam desses recursos vem crescendo, assim como o estudo para aumentar a velocidade desses sistemas, já que a velocidade do tráfego de dados entre as aplicações e o armazenamento secundário não costuma acompanhar o aumento da velocidade dos processadores e memória.

Em geral existem sistemas de arquivos distribuídos para todos os tipos de aplicações e problemas que se quer resolver. Alguns são mais tradicionais, por isso ganham respeito e são largamente utilizados, como é o caso do NFS, que agora parte para a versão 4, outros são mais inovadores e criados para resolver problemas específicos, como é o caso do CODA, criado para ter alta disponibilidade ao extremo, ou o SPRITE, criado especificamente para prover as funcionalidades necessárias para um sistema operacional realmente distribuído. Além disso, é necessário também analisar o custo de cada solução a se adotar, pois alguns deles necessitam de hardware especial.

Infelizmente não existe um sistema de arquivos completo que se adequa a qualquer aplicação. Para decidir qual sistema de arquivos paralelo utilizar, o melhor é se informar, entre os candidatos, quais se adequam ao ambiente do problema, além de saber quais os prós e contras de cada um e para quais plataformas eles funcionam bem.

Escolhidos os candidatos, um bom teste de desempenho mostrará em que situações eles funcionam bem e quais problemas podem resolver. No próximo capítulo, encontram-se alguns testes de desempenho do PVFS, NFSP e CEFT-PVFS, além de algumas comparações com alguns dos sistemas de arquivos distribuídos discutidos neste capítulo.

---

<sup>12</sup><http://www.storagetek.com.br>

## Capítulo 4

# Análise de Desempenho

Neste capítulo encontram-se testes de desempenho sobre alguns dos sistemas de arquivos paralelos estudados no capítulo anterior, mais especificamente aqueles relacionados com o PVFS. É importante deixar claro que os resultados apresentados neste capítulo são uma revisão de testes realizados por terceiros e que nossos testes estão detalhados no capítulo seguinte.

### 4.1 PVFS

Foram extraídos de [Lob03] alguns resultados experimentais sobre o PVFS e, segundo este trabalho, o desempenho do PVFS varia de acordo com vários critérios, como por exemplo:

- O número de servidores de dados (*iods*): de forma simplificada, quanto maior o número deles maior é o fluxo de dados;
- O tipo de ação (leitura, criação, escrita);
- A localização dos dados: no cache do cliente, no cache dos servidores de dados (ou mais precisamente no cache do sistema de arquivos local desses servidores) ou no disco;
- O número de clientes acessando simultaneamente os dados;
- A capacidade dos vários dispositivos envolvidos (como discos, rede, switches, processadores, memória).

O desempenho está limitado também por fatores que impõe um valor máximo ao fluxo de dados transmitidos, que são:

- A banda passante agregada<sup>1</sup> dos *iods*;
- A banda passante agregada dos clientes;

---

<sup>1</sup>Quantidade total de dados dividido pelo tempo usado para a transferência completa.

- A velocidade de transmissão dos dados de cada um dos discos dos *iods*;
- A velocidade dos switches que conectam a rede;
- A capacidade do servidor de meta-dados e/ou dos *iods* de gerenciar muitas requisições simultâneas de vários clientes.

Os testes realizados usaram a seguinte configuração:

- Pentium III 735MHz com 256MB de RAM;
- Disco “Maxtor DiamondMax Plus 40, 15,3 GB (ref. 51536U3)” (interface IDE, 7200rpm, latência de 9ms, ATA 66, buffer de 2MB), 9GB disponíveis em uma partição para armazenamento dos arquivos de teste. A banda máxima medida para esse disco foi de aproximadamente 24,4MB/s;
- Disco “Quantum Fireball lct15 7,5 GB” (interface IDE, 4400rpm, latência de 6,8ms, ATA 66, buffer de 512Kb), 2,7GB disponíveis em uma partição para armazenamento de arquivos de teste. A banda máxima medida para esse disco foi de 15,3MB/s;
- Placas de rede Ethernet 100Mbit/s (aproximadamente 12,5MB/s). Todas as máquinas desse teste estavam conectadas através de um switch de 1Gbit/s.

A maneira adotada pelos autores dos experimentos para se medir o desempenho consiste em simular diferentes comportamentos e necessidades. Para se determinar a importância desse sistema de arquivos para uma aplicação concreta, foram analisados os vários casos em que o gráfico resultante dos testes representa uma situação encontrada em uma aplicação real, para que se pudesse ter uma resposta de como seria seu comportamento. Para isso mediu-se a banda passante agregada com:

- 4, 8 e 16 servidores de dados usando discos Maxtor em perfeitas condições de funcionamento (isto é, sem erros);
- Até 60 clientes;
- Acessos do tipo leitura e escrita;
- Para dois volumes de dados acumulados (isto é, a soma dos dados lidos ou escritos pelos clientes): V-, volume de dados é menor que a capacidade total de memória dos servidores de dados; V+, volume de dados é maior que essa capacidade.

As principais dificuldades encontradas nesses testes foram: a compreensão dos sistemas de arquivos e a análise de seus pontos fortes e fracos; a implementação de scripts/programas para automatizar a configuração, instalação e ativação dos vários sistemas de arquivos; a implementação de scripts/programas para iniciar e medir o desempenho das máquinas; encontrar alguns problemas e imperfeições dos sistemas envolvidos; encontrar problemas de software (como a ordem de

inicialização dos testes em cada máquina) e de hardware (como discos com defeito); diferenças nas configurações das máquinas, que possuem discos diferentes; e a duração dos testes.

A seguir, encontram-se os resultados dos testes (que também estão disponíveis em [Lob03]). O gráfico 4.1 compara a banda passante durante pedidos de leitura no PVFS. O gráfico 4.2 compara a banda passante durante pedidos de escrita para o PVFS. O gráfico 4.3 compara a banda passante quando a quantidade de dados a serem transmitidos cabe na memória RAM dos servidores de dados. O gráfico 4.4 compara a banda passante quando a quantidade de dados a serem transmitidos não cabem na memória RAM dos servidores de dados. Todas as medidas de banda passante estão em MB/s.

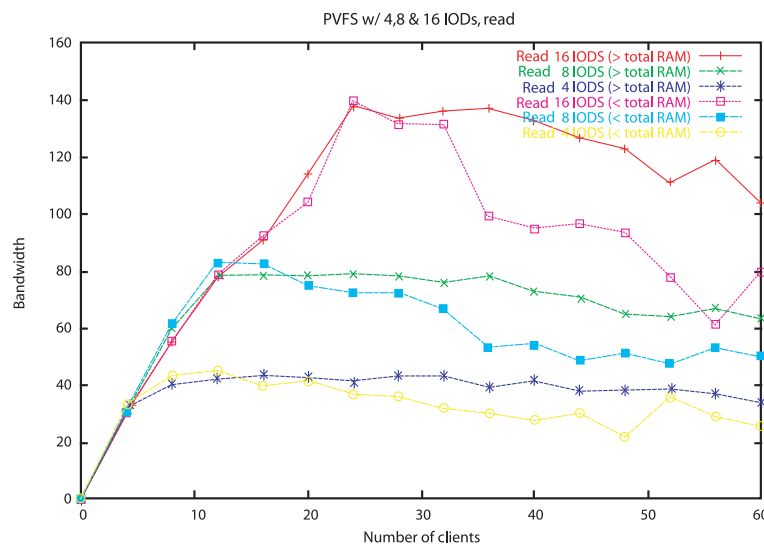


Figura 4.1: PVFS: banda passante total na leitura

Cada teste consiste na realização de operações de leitura ou escrita de arquivos pelos clientes. Cada cliente possui um arquivo para si, evitando assim concorrência no seu acesso, mas mantendo somente a concorrência no acesso ao servidor de dados. O tamanho desse arquivo varia de acordo com a quantidade de *iods*. No caso do V- ele é igual a 150MB x número de servidores de dados (600MB, 1200MB e 2400MB) e no caso V+ ele é igual a 400MB x número de *iods* (1600MB, 3200MB e 6400MB). A banda passante atingida é calculada pela divisão do volume total de dados transmitidos pelo tempo levado pelo cliente mais lento. A análise desses gráficos mostrou que:

- Seja qual for a quantidade de *iods*, a banda passante máxima atingida é claramente limitada pela banda passante das placas Ethernet (aproximadamente o número de nós vezes a velocidade da rede, ou seja:  $4 \cdot 11,5 \text{ MB/s}$ ,  $8 \cdot 11,5 \text{ MB/s}$  e  $16 \cdot 11,5 \text{ MB/s}$ ). Isto mostra que para um número de clientes não muito alto, o fluxo máximo teórico, que no caso é a banda passante agregada das placas de rede dos *iods*, pode ser alcançado. Nota-se também que este máximo não é alcançado quando o número de clientes é igual ao número de *iods*, indicando que o aproveitamento da banda não é ótimo;

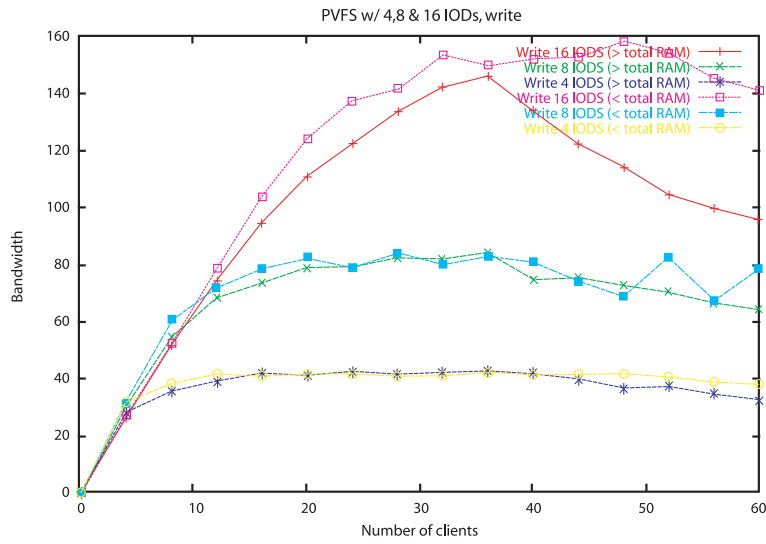


Figura 4.2: PVFS: banda passante total na escrita

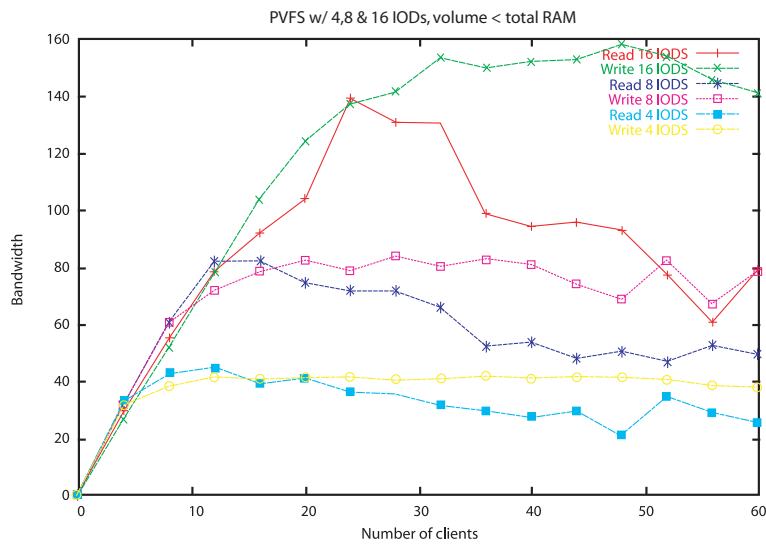


Figura 4.3: PVFS: banda passante total para volume de dados V-

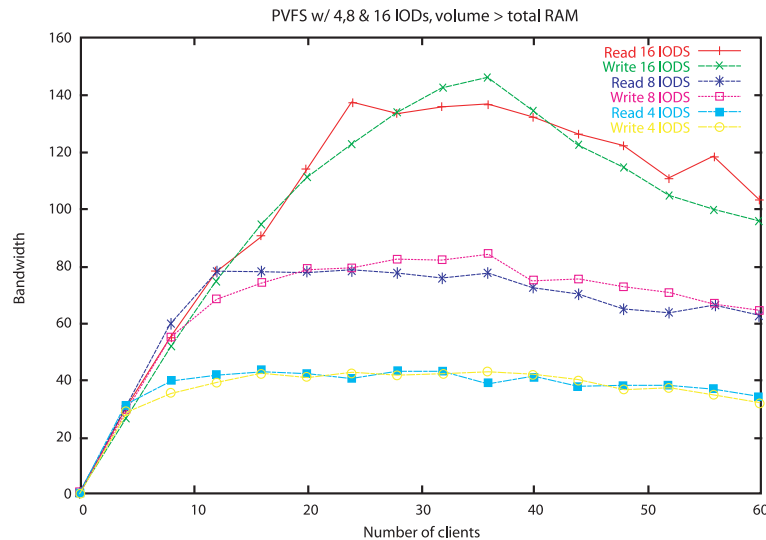


Figura 4.4: PVFS: banda passante total para volume de dados  $V >$

- A banda passante aumenta até atingir o máximo, quando então passa a se degradar pouco a pouco, especialmente quando o número de clientes aumenta, o que mostra que a largura de banda da rede não é o único critério limitante para o desempenho do sistema;
- A degradação no desempenho da escrita é mais marcante quando o volume de dados transferido é maior que o cache dos *iods*. Isto é explicado pelo fato de que os dados a serem escritos são inicialmente gravados em memória pelo sistema de arquivos local e só depois copiados assincronamente para o disco. Enquanto houver memória livre, não há urgência de se liberá-la, o que faz com que o disco não seja considerado um gargalo. Por outro lado, quando a memória livre passa a ser escassa, os dados a serem gravados são imediatamente enviados ao disco, para liberar a memória que estavam ocupando;
- A perda de desempenho em leitura é maior quando o número de *iods* é grande e mais pronunciada quando os dados não são colocados no cache. Isto é explicado pelo uso de uma política pouco justa com relação ao compartilhamento dos recursos utilizados pelos servidores de dados (memória, soquetes), isto é, os primeiros clientes a lançar pedidos são atendidos mais rapidamente que os seguintes, os quais disputam entre si o servidor (com uma concorrência maior). Como a banda passante é calculada em função do cliente mais lento, esta diminui conforme o número de clientes aumenta. Além do que, quando o número de clientes é grande, o fato de que o volume de dados lidos por cada um é pequeno (10 MB, 20 MB e 40MB por cliente respectivamente para 4, 8 e 16 *iods* para 60 clientes) não melhora as diferenças de tempo de execução. Quando, por outro lado, os dados são lidos do disco, a latência do disco é um fator que limita o uso exclusivo de recursos, levando a quase igualdade;
- É interessante notar como o desempenho em geral aumenta juntamente com o número de *iods*. Levando em conta as limitações do switch usado neste experimento, seria desnecessário,

do ponto de vista de desempenho, ter mais do que 16 nós de dados. Isso também é útil caso se deseje aumentar a capacidade de armazenamento do aglomerado.

## 4.2 NFSP

Foram extraídos de [Lob03] alguns resultados experimentais sobre o NFSP. Assim como na revisão dos testes do PVFS na seção anterior, existem quatro gráficos: O gráfico 4.5 compara a banda passante durante pedidos de leitura para o NFSP; o gráfico 4.6 compara a banda passante durante pedidos de escrita para tal sistema de arquivos; o gráfico 4.7 compara a banda passante quando a quantidade de dados a serem transmitidos cabe na memória RAM dos servidores de dados; o gráfico 4.8 compara a banda passante quando a quantidade de dados a serem transmitidos não cabe na memória RAM dos servidores de dados.

Segundo o autor dos experimentos, cada teste consiste em realizar operações de leitura ou escrita de arquivos pelos clientes. Cada cliente possui um arquivo para si, evitando assim concorrência no seu acesso, mas mantendo somente a concorrência no acesso ao servidor de dados. O tamanho desse arquivo varia de acordo com a quantidade de *iods*. No caso do V- ele é igual a 150MB x número de servidores de dados (600MB, 1200MB e 2400MB) e no caso V+ ele é igual a 400MB x número de *iods* (1600MB, 3200MB e 6400MB). A banda passante atingida é calculada pela divisão do volume total de dados transmitidos pelo tempo levado pelo cliente que demorou mais. Todas as medidas de banda passante estão em MB/s.

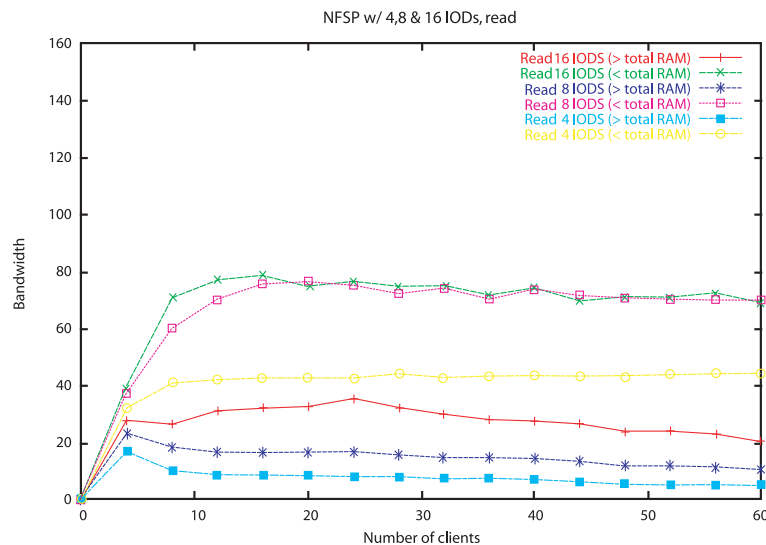


Figura 4.5: NFSP: banda passante total na leitura

Para analisar esses gráficos, deve-se levar em conta, ao observar a banda passante, que os dados podem estar no cache dos clientes, ou seja, existe um gerenciamento de cache do lado do cliente, algo que não existia nos testes do PVFS (pois ele não gerenciava tal tipo de cache), mas que é

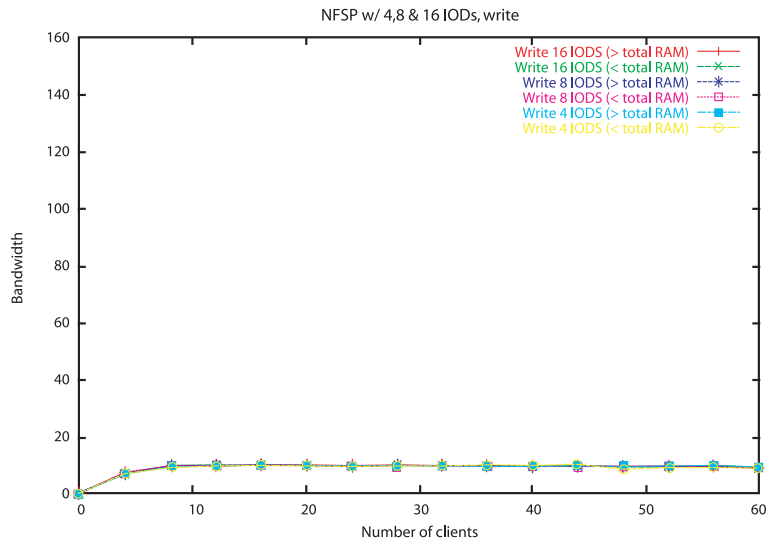


Figura 4.6: NFSP: banda passante total na escrita

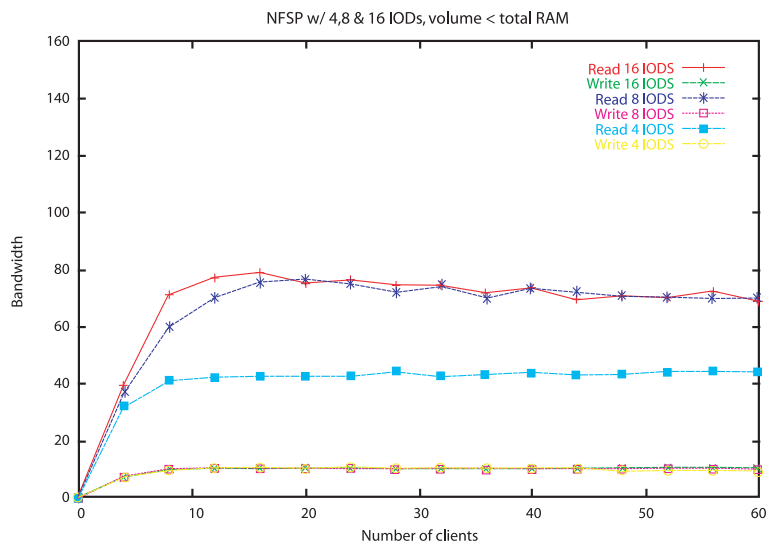


Figura 4.7: NFSP: banda passante total para volume de dados V-

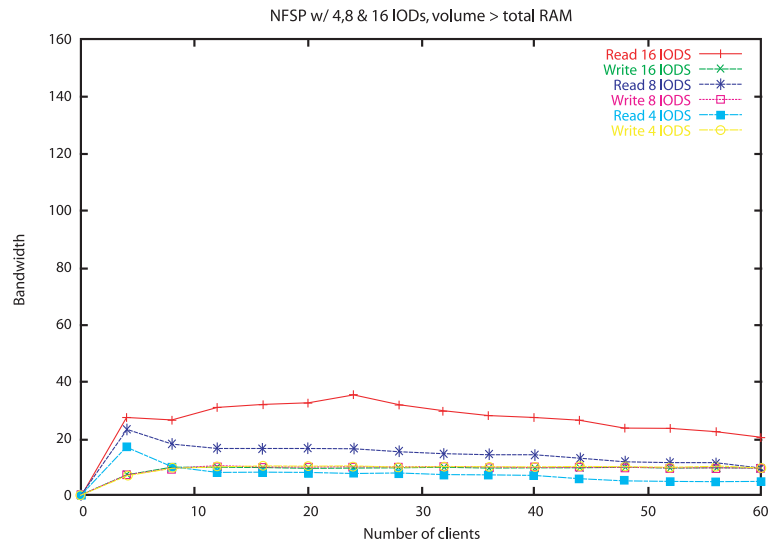


Figura 4.8: NFSP: banda passante total para volume de dados V+

de extrema importância no NFSP, pois pode representar um grande ganho de desempenho para determinados tipos de aplicação.

É fácil notar isso ao comparar os gráficos 4.1 e 4.5. No PVFS o desempenho não é tão influenciado pela quantidade de dados requisitados, mas sim pela quantidade de servidores e clientes. Já no NFSP a banda passante resultante na leitura de poucos dados (V-) não é menor que o dobro da banda passante resultante na leitura de mais dados do que o tamanho da memória RAM (V+), para a mesma quantidade de servidores e clientes. Além disso, pode-se tirar outras conclusões a partir desses gráficos:

- Para qualquer quantidade de *iods* e para qualquer volume de dados, a banda passante para escrita permanece estável para um grande número de clientes e não excede 10MB/s, que representa 81% da banda passante de rede do servidor de meta-dados. Dado que todas as informações a serem gravadas precisam passar pelo servidor de meta-dados, esse resultado não é de se surpreender;
- Considerando a leitura de dados do cache dos servidores de dados, a banda passante aumenta regularmente até atingir 44MB/s, o que representa 90% da banda agregada da rede dos *iods*. Para 8 *iods*, o desempenho é menor pois a banda fica em 76MB/s (o que corresponde a 77% do total ótimo) para 20 clientes. Esse valor vai caindo gradativamente até chegar a 70MB/s para 60 clientes. O resultado é mais surpreendente para 16 *iods*, pois a banda passante não passa de 78MB/s (o que corresponde a 40% do ótimo) para 12 clientes. Uma explicação é a saturação do processador do servidor de meta-dados. Isso indica que o servidor de meta-dados não poderia processar mais do que 10000 requisições por segundo, pois como cada requisição corresponde a um bloco de 8KB, tem-se  $78\text{MB}/8\text{Kb} = 9984\text{req/s}$ ;
- O desempenho é relativamente ruim quando os dados não estão no cache e é ainda pior

quando o número de clientes cresce. Com 4 *iods* obteve-se 17MB/s de uso de banda para 4 consumidores antes de decair gradativamente para 4MB/s para 60 clientes. Com 8 *iods* foi obtido 23MB/s para 4 clientes e caiu para 10MB/s para 60, sendo que a banda passante máxima atingida foi de 35MB/s para 20 clientes. Neste caso, o servidor de meta-dados não aparenta ser o gargalo, pois o número de requisições por segundo é baixo;

- Para um número suficiente de clientes, dobrar o número de *iods* em geral dobra a banda passante usada, o que significa que a carga é bem distribuída entre os servidores de dados.

### 4.3 CEFT-PVFS

Foram extraídos de [ZJQ<sup>+</sup>03] alguns resultados experimentais sobre o CEFT-PVFS, que estão detalhados nessa seção.

Conforme já mencionado na seção 3.2.5 todos os dados armazenados no CEFT-PVFS terão duas cópias: uma no grupo primário e outra no grupo secundário. Essa sobrecarga de armazenamento fornece uma melhor tolerância a falhas, ao mesmo tempo que possibilita um melhor paralelismo na leitura dos dados, pois disponibiliza os mesmos dados em ambos os grupos de servidores.

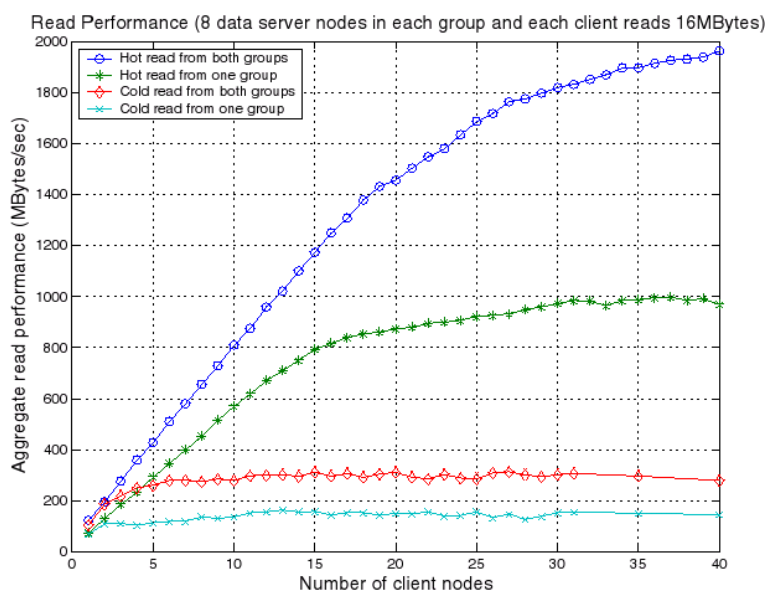


Figura 4.9: Desempenho de leitura do CEFT-PVFS ao variarmos o número de clientes

Isso permite que um cliente possa ler dados simultaneamente de ambos os grupos, reduzindo a sobrecarga no servidor, o que libera recursos para outros clientes. No gráfico 4.9 observa-se o ganho ao se utilizar o CEFT-PVFS com dois grupos contra o mesmo sistema de arquivos com apenas um grupo (cujo funcionamento é similar ao PVFS). Nele, entende-se por *hot read* a leitura dos dados que já estão no cache do servidor, minimizando os acessos ao disco, e por *cold read* a leitura dos dados que não estão em cache, maximizando o uso dos discos.

No gráfico 4.10 observa-se o desempenho do sistema de arquivos quando varia-se a quantidade de dados lidos. Conforme essa quantidade aumenta, o desempenho cai quando os dados são lidos do disco. Uma das causas aparentes, segundo os autores do teste, é que os dados não estão armazenados de forma contínua, provocando uma busca no disco.

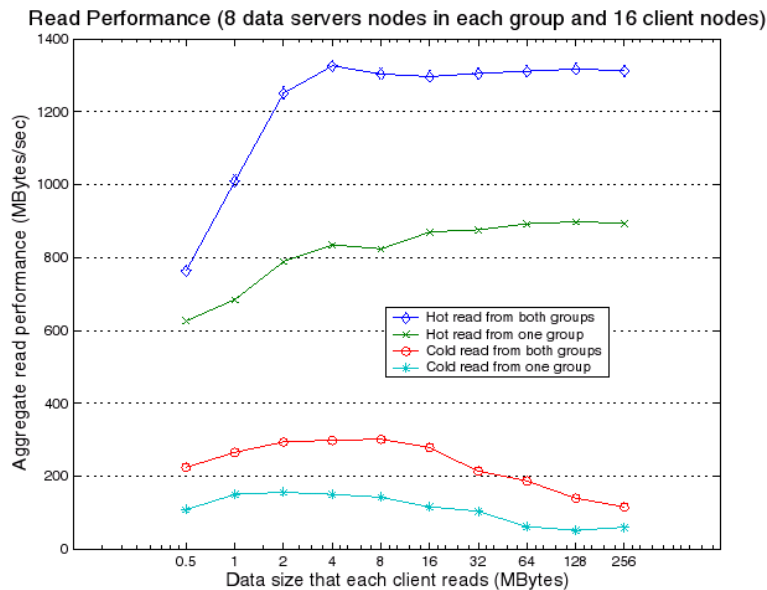


Figura 4.10: Desempenho de leitura do CEFT-PVFS ao variarmos a quantidade de dados lidos

Ainda segundo os autores, com o uso do CEFT-PVFS, houve um ganho de 69% e 91%, em média, ao se utilizar dois grupos de servidores, para os casos de leitura diretamente do cache e do disco, respectivamente.

## Monitoramento de Carga

Em um aglomerado de computadores é muito comum que um servidor seja também usado como cliente. Dessa forma, os recursos disponíveis são compartilhados entre os processos, o que pode degradar o serviço oferecido pelos servidores.

A redundância que o CEFT-PVFS proporciona fornece um caminho alternativo para que os clientes desviem desses servidores sobrecarregados. Para isso, cada servidor monitora o seu uso de processadores, discos e rede, enviando os resultados periodicamente para o servidor de meta-dados. Tal servidor organiza os pedidos dos clientes, informando-os qual o melhor servidor de dados para ser acessado naquele momento.

O gráfico 4.11 foi gerado a partir da sobrecarga do uso do disco de um dos servidores de dados, para mostrar as vantagens dessa estratégia para o caso *cold read*. Note que conforme a quantidade de dados lidos aumenta, desviar do servidor sobrecarregado começa a não valer mais a pena. Isso acontece devido ao cliente estar acessando apenas um dos grupos de servidores, ao invés de dividir a tarefa de leitura com o outro grupo.

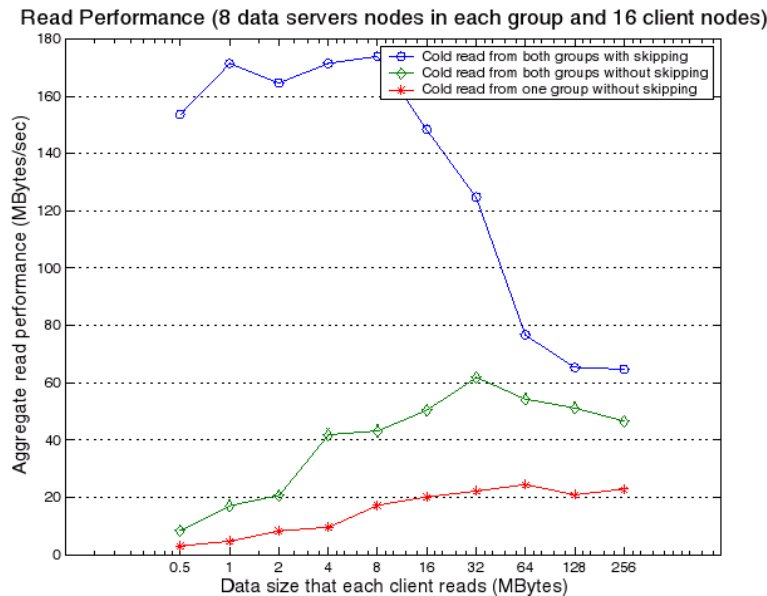


Figura 4.11: Desempenho do CEFT-PVFS desviando de servidor com disco sobrecarregado

Já para o caso *hot read*, como o uso do disco é mínimo, o gráfico 4.12 compara alguns casos onde a sobrecarga ocorre no uso da rede de um dos servidores. Conforme o número de clientes aumenta, desviar do servidor sobrecarregado vale mais a pena, pois o gargalo na rede deixa de ser no cliente e passa para o servidor. Ao estressar a rede, a banda agregada usada baixa de 2GB/s para 1,25GB/s. Ao se desviar do servidor cuja rede está sobrecarregada, a banda aumenta para 1,53GB/s, o que gera um ganho de 22,4%, quando se utiliza essa estratégia.

#### 4.4 Análise Conclusiva

Dentre os sistemas de arquivos paralelos considerados neste capítulo, todos os testes puderam comprovar que o PVFS possui grandes qualidades para lidar com o processamento concorrente de requisições vindas de múltiplos clientes.

Além disso, nota-se que é totalmente viável utilizá-lo como base para a construção de outros sistemas de arquivos paralelos mais específicos ou que possuem melhorias com relação ao PVFS, pois aproveita-se seu ganho de desempenho ao lidar com aplicações paralelas e concorrentes.

No momento em que esse texto foi escrito, não foram encontrados testes oficiais de desempenho sobre o PVFS2 para compará-lo com esses outros sistemas, a fim de se verificar se as novas características implementadas realmente funcionam e se melhoram o comportamento desse sistema.

Assim, foram realizados alguns experimentos simples com a versão 1.6.3 do PVFS1 e com a versão 1.0.1 do PVFS2, para se decidir qual dos dois seria usado nos testes do capítulo 5. Tais experimentos consistiam na leitura arquivos de 1GB localizados no PVFS1 e PVFS2, utilizando-se de várias threads. Não foram notadas grandes diferenças de desempenho entre eles e ambos se

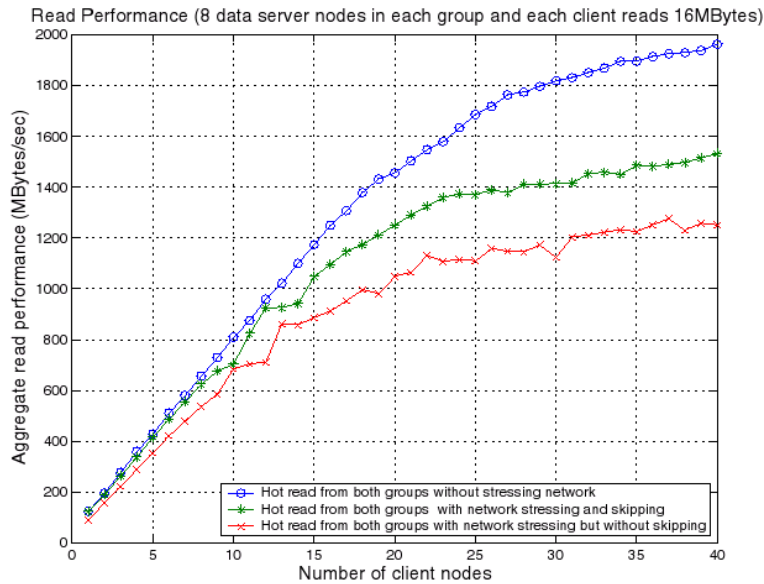


Figura 4.12: Desempenho do CEFT-PVFS desviando de servidor com rede sobrecarregada

comportaram de forma estável.

Como já era de se esperar, essa versão inicial do PVFS2 é considerada apenas uma re-arquitetura do PVFS1, não tendo ainda grande vantagem para o usuário final com relação ao seu antecessor, pois não implementa muitas das vantagens previstas e discutidas na seção 3.2.3. Devido às pequenas diferenças entre os dois, optamos por usar a versão mais recente em nossos experimentos do próximo capítulo.

# Capítulo 5

## PVFS2 vs. Ext3

No capítulo anterior, foram analisados testes e comparações de desempenho entre alguns sistemas de arquivos paralelos. Em todos os casos sempre houve vários clientes acessando um único sistema de arquivos, de forma simultânea e concorrente.

Neste capítulo, realizamos uma comparação pouco usual: colocamos apenas uma máquina como cliente, acessando o sistema de arquivos paralelo PVFS2, e analisamos seu comportamento contra o mesmo cliente, acessando o sistema de arquivos local Ext3.

À primeira vista parece estranho compará-los, já que têm propósitos distintos, porém, se houver largura de banda o suficiente na rede para o tráfego dos dados, um sistema de arquivos paralelo pode ser mais rápido que um sistema de arquivos local. Para deixar mais clara a idéia, a figura 5.1 mostra um caso onde existe uma rede muito rápida interligando servidores e clientes, onde os dispositivos de armazenamento são notavelmente mais lentos.

Grandes requisições enviadas ao PVFS2, mesmo que vindas de um único cliente, utilizam toda a banda disponível dos discos dos servidores, já que não têm a rede como gargalo, enviando o resultado para o cliente rapidamente. Comparando essa banda agregada com a banda disponível pelo sistema de arquivos local, percebe-se a clara vantagem em se utilizar o PVFS2.

Nas seções a seguir, demonstramos esse fato com a realização de testes concretos, simulando uma rede de dados mais rápida que o sistema de armazenamento, a partir da redução da velocidade dos discos de ambos, cliente e servidores.

### 5.1 Ambiente Utilizado

Antes de analisar os resultados encontrados, é importante conhecer a configuração utilizada no ambiente de testes e todas as condições que podem influenciar tais resultados.

#### 5.1.1 PVFS2

A instalação do PVFS2 foi feita conforme instruções do *Quick Guide* disponível no site oficial do sistema. Utilizamos a última versão estável disponível, 1.0.1, que é específica para a linha 2.6 do *kernel* do linux. Não tivemos grandes problemas para realizar essa instalação, embora tenha sido

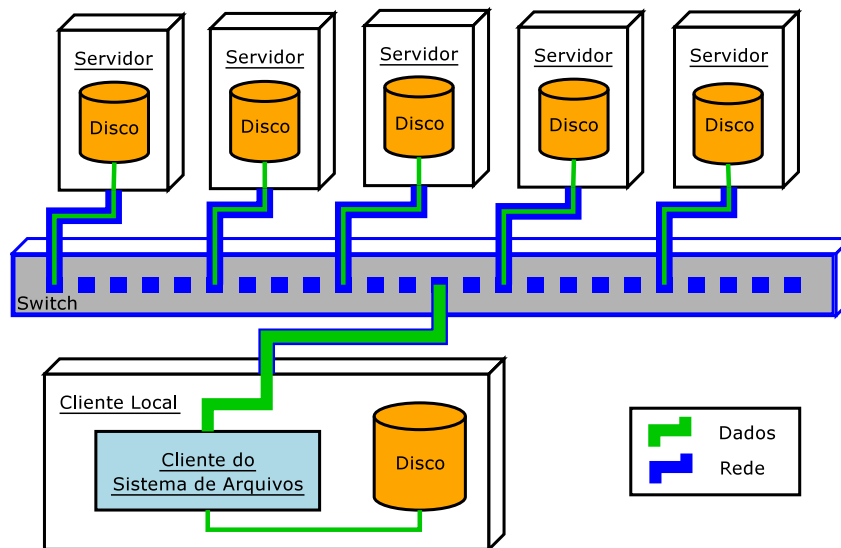


Figura 5.1: Sistemas de arquivos paralelos vs. locais

necessário alterar algumas constantes definidas em alguns arquivos do código fonte, pois o script de auto-configuração não estava funcionando corretamente em nosso ambiente.

O aglomerado usado para testes foi um *beowulf* do projeto de bioinformática do IME-USP, onde todas as máquinas possuem processadores Athlon de 1.2GHz, 768MB de memória RAM, 16GB de espaço livre em disco Ultra ATA/133, 7200 RPM, em uma partição montada como Ext3. A conexão à rede é feita por placas 10/100Mbit/s, diretamente ligadas a um switch de 100Mbit/s.

Em todas as máquinas foi instalada a distribuição Debian GNU/Linux, versão não-estável do mês de fevereiro de 2005. A versão do *kernel* foi a 2.6.10, sendo que optamos pelo pacote original da distribuição (isto é, não é uma versão compilada por nós).

A configuração do PVFS2 não foi alterada com relação à instalação padrão. A fim de obter o desempenho máximo do PVFS2, decidimos deixar o servidor de meta-dados dedicado para esse papel e variamos a quantidade de servidores de dados em 1, 2, 4 e 8 máquinas, dependendo do teste a ser realizado.

A máquina usada como cliente é distinta dos servidores, sendo que todos os testes de tomada de tempo utilizaram a interface UNIX para acesso ao PVFS2. A interface UNIX usada é um módulo do *kernel* que permite ao usuário montar o PVFS2 e utilizá-lo como um sistema de arquivos comum, de forma transparente.

### 5.1.2 Ext3

O sistema de arquivos local usado para nossos testes é o Ext3, que é uma extensão do Ext2 com suporte a *journaling*, ou *log*. Tal suporte evita que uma operação de escrita ou mudança mal executada na estrutura do sistema de arquivos deixe todo o sistema em um estado inconsistente. Esse tipo de problema costuma ocorrer quando um sistema de arquivos é desativado antes que ele

possa ser desmontado (por exemplo, queda de energia).

Para nossos testes, o sistema de arquivos local foi instalado em um disco IDE padrão Ultra ATA/133, 7200 RPM, do mesmo modelo dos discos sendo utilizados para armazenamento local dos dados do PVFS2. Isso garante uma comparação justa entre esses dois sistemas de arquivos. Para evitar qualquer tipo de discrepância quanto às condições durante a comparação, a máquina cliente usada para os testes do PVFS2 é a mesma utilizada para os testes do Ext3.

## 5.2 Sobre o Processo de Testes

Para se comparar o comportamento do PVFS2 e do Ext3, é necessário definir o que será testado e quais serão as condições de teste. Assim, preparamos alguns testes para a realização de leituras e escritas concorrentes a blocos de dados armazenados nesses dois sistemas de arquivos.

Porém, foi preciso definir algumas variáveis que indicariam mudanças no comportamento dos testes para cada valor que lhes for atribuído. Tais variáveis, bem como os valores usados nos testes, são:

- **Tamanho do arquivo:** 1MB, 2MB, 4MB, 8MB, 16MB, 32MB, 64MB, 128MB, 256MB, 512MB e 1GB;
- **Tamanho do bloco:** 1KB, 4KB, 16KB, 64KB, 256KB e 1MB;
- **Quantidade de threads:** 1, 2, 4, 8, 16 e 32;
- **Modos do disco<sup>1</sup>:** PIO 0, PIO 1, PIO 2 e PIO 3;
- **Local de armazenamento:** PVFS2 e Ext3;

Cada uma de nossas amostras, obtidas após os testes, é a combinação das variáveis acima. Para cada tipo de comparação que realizamos, fixamos algumas dessas variáveis e variamos as demais, apresentando os resultados em forma de gráficos de séries ou barras.

Para realizar todos esses testes, surgiu a necessidade de criarmos ferramentas confiáveis. Dentre elas, precisamos gerar arquivos com tamanhos variados, realizar leitura concorrente, podendo configurar tamanho de bloco, quantidade de threads, entre outras.

Além disso, foi necessário definir um processo que evitasse que um teste influenciasse outro, ou que variáveis fora de nosso controle pudessem influenciar os resultados. Nas seções a seguir descrevemos o processo e as ferramentas utilizadas para a realização desses testes.

### 5.2.1 Geração dos Arquivos para Teste de Leitura

Criamos um programa simples, chamado de **FGen** (ou *File Generator*), para a geração dos arquivos usados nos testes. O objetivo era criar um programa que gerasse uma certa quantidade de dados aleatórios em tamanho de bloco configurável, de forma seqüencial, o mais rápido possível,

---

<sup>1</sup>Para mais informações, veja a seção [B.2](#) do Apêndice [B](#)

isto é, minimizando o número de operações que não envolvessem chamadas de escrita para o sistema operacional.

A velocidade era primordial, pois haveria dois casos importantes: teríamos que gerar milhões de bytes a cada teste, pois ao alterarmos algumas variáveis do ambiente de teste, como por exemplo o número de servidores de dados PVFS2, seria necessário gerar todos os arquivos novamente; poderíamos ter que testar o desempenho de escrita dos sistemas de arquivos envolvidos (o que não ocorreu através dessa ferramenta).

Uma opção seria copiar os dados do sistema de arquivos local para o sistema paralelo, evitando assim ter que gerar novamente os arquivos. Mas em uma breve tomada de tempo, descobrimos que é mais rápido gerar um arquivo (apenas operação de escrita) do que copiá-lo (operações de leitura e escrita, intercaladas).

### 5.2.2 Leitura e Escrita Paralela e Concorrente

Para realizar as tomadas de tempo no acesso paralelo e concorrente do sistema de arquivos, precisaríamos de um programa que não se utilizasse muito do processador da máquina e que lesse ou escrevesse os dados o mais rápido possível. Além disso, tal programa deveria usar threads para realizar leitura ou escrita paralela e deveria usar tamanhos de bloco variável.

Dessa forma, surgiu o **PSplit** (ou *Parallel Split*, descrito melhor no Apêndice A), usado tanto para quebrar um arquivo em pedaços de igual tamanho, quanto simplesmente ler um arquivo utilizando-se de várias threads. Além disso, o PSplit também é usado para gerar arquivos com dados aleatórios, utilizando-se de várias threads, para testes de escrita concorrente.

O PSplit permite que o usuário escolha entre definir o tamanho de cada pedaço do arquivo a ser lido ou a quantidade de pedaços a serem lidos. Dada essa escolha, internamente são efetuados cálculos para se definir a quantidade de pedaços ( $n$ ) ou seus respectivos tamanhos ( $s$ ). Imediatamente são criadas  $n - 1$  threads (a thread principal também é usada para leitura de dados) que irão ler cada uma  $s$  bytes a partir do byte  $\frac{t \cdot s}{n}$  do arquivo de entrada, onde  $t$  é o número da thread, que varia de 0 (thread principal) a  $n - 1$ . Para escrita, deve-se especificar tamanho e número de arquivos a serem gerados.

Ao final do processamento, o PSplit informa o tempo total gasto para ler ou gravar todos os dados, além da velocidade de leitura média em bytes/s, entre a abertura do arquivo até o último byte lido ou gravado.

### 5.2.3 Leitura Seqüencial vs. Aleatória

A leitura usada pelo PSplit em nossos testes foi a seqüencial, para cada thread criada. Isso significa que cada thread possui uma posição inicial e final dentro de um arquivo. Ela então divide esse pedaço do arquivo em blocos de igual tamanho e os lê de forma seqüencial, da posição inicial até a final.

A leitura seqüencial foi escolhida por dois motivos. Ela é mais simples de se controlar, e o desempenho da leitura aleatória foi muito similar à seqüencial, em todos os casos, utilizando-se das mesmas condições. Para se medir isso, foram realizados alguns testes com o PSplit usando leitura

aleatória. Nela, cada thread sorteava blocos aleatoriamente, sendo que cada um era lido apenas uma vez, até que todos os blocos tenham sido lidos.

#### 5.2.4 Um Arquivo Para Leitura e Vários Para Escrita

Os testes de leitura foram baseados no acesso a apenas um arquivo, utilizando-se de várias threads, todas dentro de um único processo. Porém, os testes de escrita utilizaram-se de vários arquivos. A razão dessa decisão foi puramente prática.

Testes preliminares indicaram que essa decisão não influencia nos resultados, pois enquanto que no acesso a vários arquivos ocorrerá uma sobrecarga para abrir cada um deles, no acesso a um único arquivo cada thread deverá abrir o mesmo arquivo, anulando qualquer tipo de vantagem.

O único benefício que poderia-se esperar ao abrir o mesmo arquivo várias vezes seria algum tipo de otimização no cliente para evitar o contato com o servidor de meta-dados para cada pedido desse tipo, utilizando-se do cache. Porém, existindo ou não tal cache, os resultados não foram influenciados por isso.

Quanto ao uso de várias threads ou vários processos, o sistema operacional Linux os trata de forma muito parecida nas trocas de contexto, o suficiente para não afetar o desempenho dos testes realizados. Para mais informações sobre as diferenças entre processos e threads consulte [Aas05].

#### 5.2.5 Garantindo Que os Dados Vêm do Disco

O VFS (*Virtual File System*) do Linux é uma camada de abstração para todos os sistemas de arquivos que funcionam sob o Linux. Essa camada facilita muito o desenvolvimento de sistemas de arquivos, pois fornece uma interface com várias funções comuns já prontas, mas que podem ser sobrescritas.

Uma das vantagens é o controle do cache do sistema de arquivos, que é realizado automaticamente, caso não seja implementada uma mudança no seu comportamento.

No caso do PVFS2, seu cliente se utiliza desse cache, da mesma forma que o Ext3, que pode ocupar toda a memória física livre. Além disso, o servidor PVFS2 armazena seus dados em disco local, utilizando o próprio sistema de arquivos local, o que gera mais um nível de cache.

Isso certamente atrapalharia nossas tomadas de tempo, pois caso os dados a serem lidos já estivessem no cache local do servidor, não haveria acesso ao disco e caso eles estivessem no cache local do cliente, não haveria acesso à rede e muito menos ao disco.

Para evitar esse tipo de problema, foi criado um programa chamado **fillem**, que aloca toda a memória física da máquina. Porém, o gerenciamento de memória do Linux 2.6 foi desenvolvido pensando-se em tratar programas que alocam muita memória mas não a utilizam. Dessa forma, se um programa pedir memória, o Linux só irá realmente fornecê-la caso ela seja alterada. Nossos testes levaram isso em consideração, alocando a memória em blocos de 1MB e inserindo um valor aleatório em alguma posição.

A realização dessa tarefa antes de cada teste fez com que todo o cache do sistema de arquivos fosse removido, para dar espaço à essa grande alocação de memória. Isso foi realizado em todas as máquinas envolvidas, o que garantiu que cada arquivo a ser lido do PVFS2 ou do Ext3 não estivesse no cache da memória.

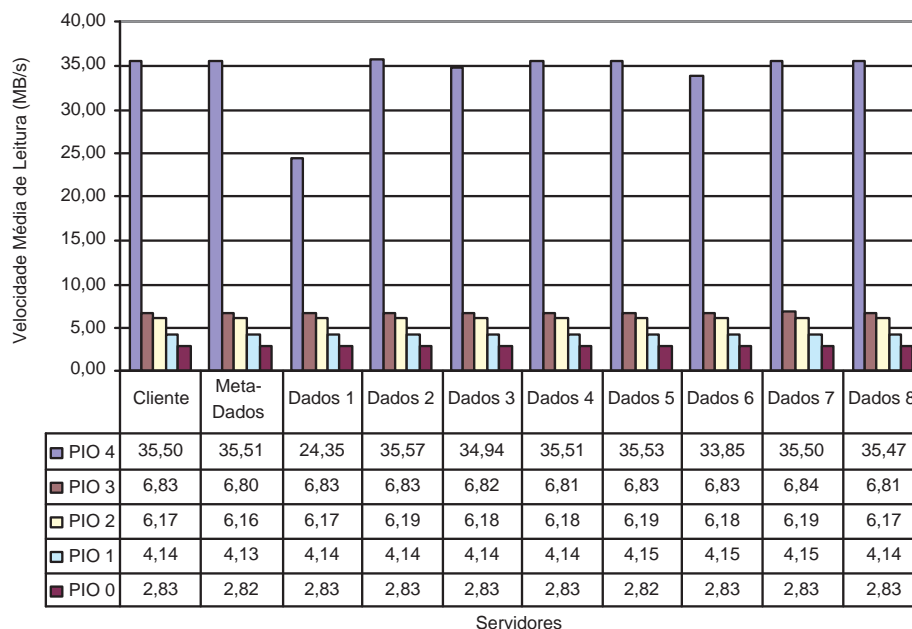


Figura 5.2: Médias das velocidades de leitura de cada um dos discos, nos vários modos PIO

### 5.2.6 Aumentando a Largura da Banda da Rede

A rede utilizada em nosso aglomerado possui velocidade máxima de 100Mbit/s (aproximadamente 12,5MB/s). Já a velocidade máxima de nossos discos é de 133MB/s, embora sua média seja de aproximadamente 35,5MB/s no modo PIO 4 UDMA.

Para se conseguir realizar uma comparação entre PVFS2 e Ext3 onde a rede é mais rápida que o disco, a primeira idéia seria aumentar a velocidade da rede. Uma rede de 10Gbit/s proporcionaria aproximadamente 1,25GB/s de transferência de dados, permitindo realizar tal comparação.

Porém, isso implicaria em mudar a configuração do aglomerado, o que envolveria um custo elevado. Assim, tivemos a idéia de reduzir a velocidade dos discos das máquinas, tanto cliente como servidores.

Para isso, utilizamos a ferramenta **hdparm**<sup>2</sup>, que permite mudar o modo de transferência de dados da interface IDE. Assim, mudando o modo de PIO 4 UDMA para PIO 0, a velocidade média do disco é reduzida de 35,5MB/s para apenas 2,8MB/s. A tabela da figura 5.2 mostra os valores obtidos para cada modo PIO em cada um dos discos usados em nossos testes.

Dessa forma, temos que a velocidade de cada um dos discos poderia ser reduzida mais de 10 vezes, enquanto que a velocidade da rede seria mantida, sendo até 4 vezes mais rápida que qualquer um dos discos. Assim, a rede deixaria de se tornar um gargalo para a saída dos dados vindos do

<sup>2</sup><http://freshmeat.net/projects/hdparm/>

disco. Com isso, consegue-se simular a situação onde a rede é mais rápida que os discos.

### 5.2.7 Scripts Para Testes Automatizados

Foi desenvolvido um script em linguagem Shell para Unix onde o objetivo principal era usar o PSplit de maneira simples e sem perda de tempo, agilizando os testes. Porém, devido à necessidade de se alterar alguns parâmetros do ambiente para determinados testes, esse script também é responsável por:

- ajustar a quantidade de servidores PVFS2;
- criar os arquivos de testes de leitura (usando o FGen);
- limpar o cache do sistema (cliente e servidores, usando o fillmem);
- reduzir ou aumentar a velocidade dos discos (usando hdparm);
- realizar os testes e coletar todos os resultados em arquivos distintos por tipo de teste (usando o PSplit).

Ao final, são gerados dados importantes sobre a execução, como uso do processador pelo processo ou pelo sistema, tempo total de execução, total de páginas usadas na alocação da memória, etc. Além disso, o PSplit gera informações mais precisas sobre a quantidade de dados trafegados e em quanto tempo.

Esses dados são armazenados em um arquivo para cada configuração do sistema de arquivos, disco, leitura ou escrita. Para cada arquivo, temos cinco linhas de resultado para cada configuração de número de threads, tamanho do bloco de leitura e quantidade de dados lidos.

### 5.2.8 Processamento dos Resultados

Conforme mencionado na seção anterior, cada teste foi executado cinco vezes. Isso foi usado para fins estatísticos na geração dos gráficos, evitando que elementos desconhecidos durante a execução (como um processo agendado previamente em um servidor) pudesse afetar os resultados.

Para calcular o valor final a ser usado na análise dos resultados, descartou-se os valores mínimos e máximos obtidos, com o objetivo de remover grandes discrepâncias, e usou-se a média dos outros três valores restantes. Porém, ao realizar a média com os cinco valores, sem descartar nenhum dos testes, observou-se uma diferença de no máximo 0,1% comparada com a estratégia de se remover os picos, ou seja, em ambos os casos temos resultados equivalentes.

## 5.3 Análise dos Resultados de Leitura

Após realizados os testes, todas as informações foram coletadas, organizadas, agrupadas e analisadas, para que pudessem ser usadas na comparação entre PVFS2 e Ext3. Nessa seção analisamos os resultados do comportamento dos sistemas de arquivos nas situações em que foram testados.

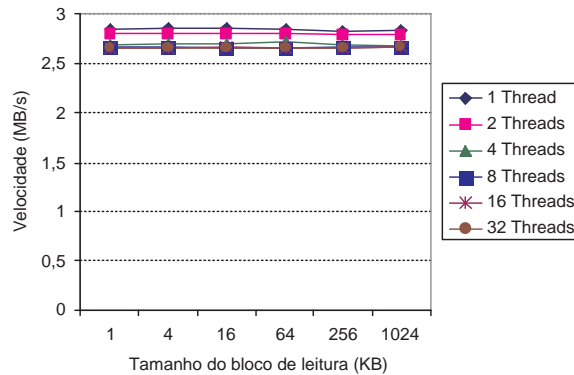


Figura 5.3: Desempenho do Ext3 para os vários tamanhos do bloco de leitura, lendo 1GB de dados em modo PIO 0

### 5.3.1 Variação do Tamanho do Bloco de Leitura

Notamos que a variação do tamanho do bloco de leitura de dados não influencia no desempenho de ambos os sistemas de arquivos testados. Observe no gráfico 5.3 que não existe grande alteração nos resultados quando varia-se o tamanho do bloco de leitura para o sistema de arquivos local. O mesmo acontece para o PVFS2, como observa-se no gráfico 5.4.

O resultado também se mantém ao variar o modo PIO e a quantidade de dados lidos. Dessa forma, todos os testes apresentados a seguir se utilizam de blocos de tamanho igual a 64KB, para simplificar a análise, já que essa variável não influencia o desempenho dos sistemas de arquivos em questão.

### 5.3.2 Variação do Tamanho do Arquivo

Entende-se como variação do tamanho do arquivo a quantidade de dados lidos durante os testes. Ou seja, para testes com arquivos de  $n$  bytes de tamanho, foram lidos exatamente  $n$  bytes do sistema de arquivos.

No gráfico 5.5 nota-se que a variação da quantidade de dados lidos não influencia o sistema de arquivos Ext3, porém vemos na figura 5.6 que isto afeta muito o PVFS2, especialmente quando se aumenta o nível de concorrência.

Note nos gráficos da figura 5.6 que a velocidade de leitura do PVFS2 cai quase que pela metade quando dobra-se a quantidade de threads mas fixa-se o volume de dados lidos. Ou, em outras palavras, o tempo total gasto para a leitura de um arquivo pequeno se mantém constante conforme o nível de concorrência aumenta.

Ao aumentar o número de servidores de dados do PVFS2, essa característica se mantém. Isso pode ser explicado pelo fato de que há uma sobrecarga inicial considerável quando se lê arquivos pequenos, que passa a ser menos relevante conforme a quantidade de dados aumenta. Tal sobrecarga é a soma dos tempos gastos com a criação das threads de leitura, a alocação de espaço em memória

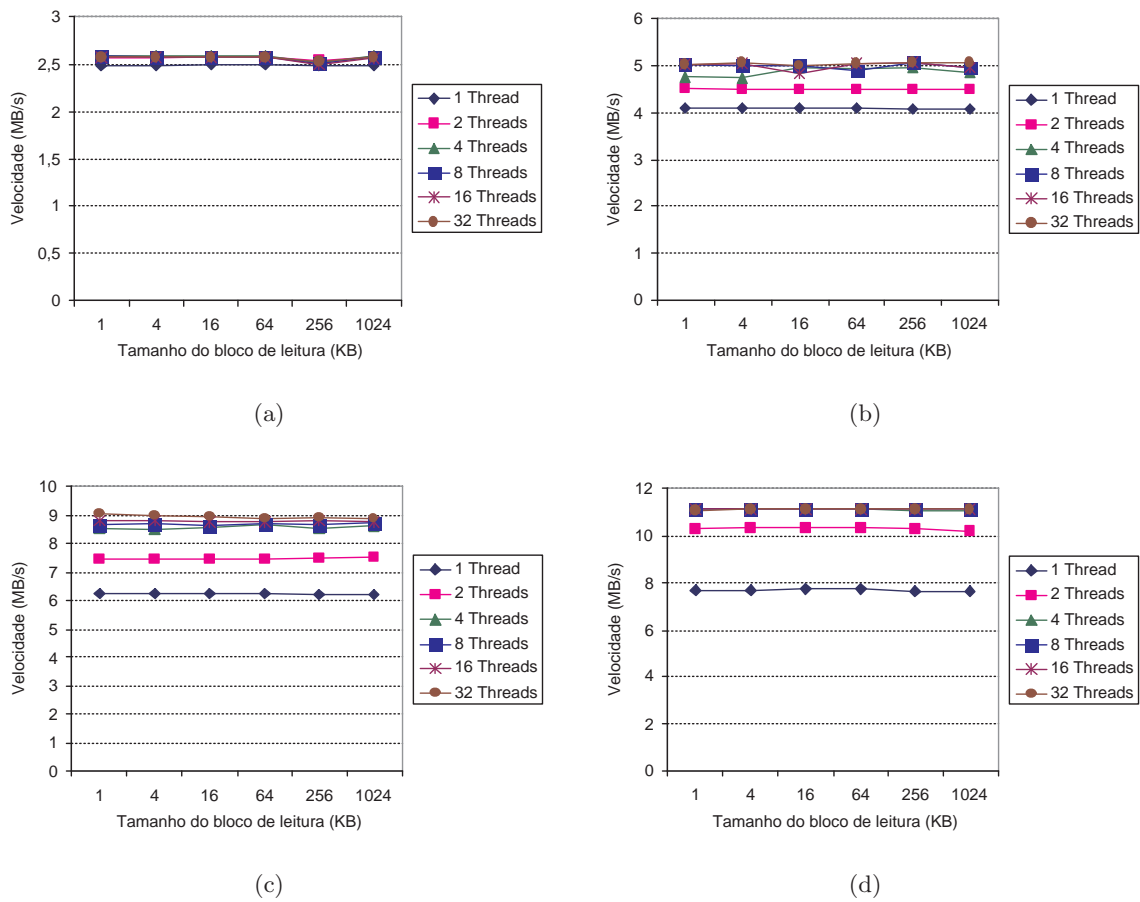


Figura 5.4: Desempenho do PVFS2 com (a) 1, (b) 2, (c) 4 e (d) 8 nós de dados, variando tamanho do bloco de leitura, considerando 1GB de dados lidos em modo PIO 0

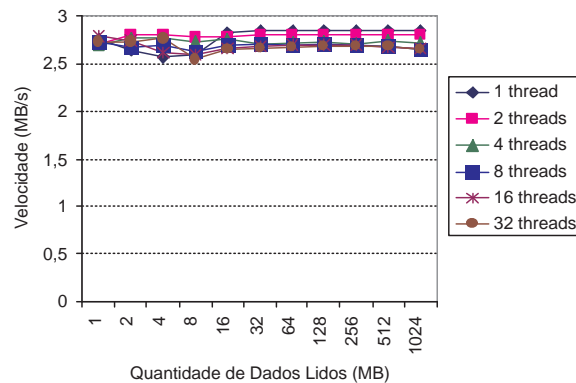


Figura 5.5: Desempenho do Ext3 variando a quantidade de dados lidos, em blocos de 64KB, modo PIO 0

para armazenamento das estruturas de leitura e os vários pedidos de abertura do arquivo, que cada thread necessita realizar.

Utilizando dos mesmos dados, mas mudando um pouco a forma de apresentá-los, pode-se comparar o Ext3 com as várias configurações do PVFS2. Note que o PVFS2 com apenas um nó de dados perde para o Ext3 (gráficos da figura 5.7), mas a partir de dois ganha. Note também que com apenas uma thread de leitura (veja gráfico 5.7(a)) não se tem ganho de desempenho tão grande conforme o tamanho do arquivo lido aumenta, como o que ocorre com duas ou mais threads.

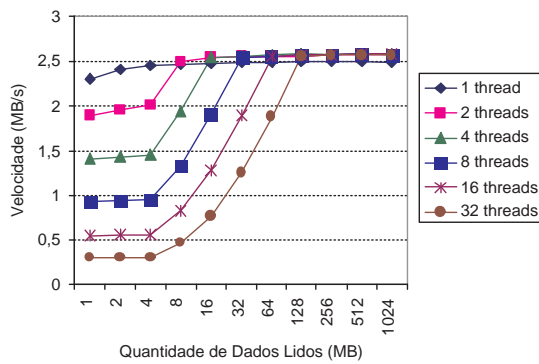
Ainda analisando o resultado com 1 thread, para 2 e 4 nós de dados, o desempenho cai cerca de 7% quando se aumenta a quantidade de dados lidos de 128MB para 256MB, e outros 7% de 256MB para 512MB. Porém, para 1GB de dados lidos a velocidade se mantém. Vários testes nessa situação foram executados, em diversos momentos, variando o tamanho do bloco de leitura e a quantidade de dados lidos, a fim de se ter certeza de que é uma característica do PVFS2 e não uma situação isolada gerada devido às condições momentâneas do aglomerado. Mesmo assim, o comportamento se manteve.

Ao usar 2 threads (gráfico 5.7(b)), o PVFS2 com apenas um nó de dados também perde do Ext3, mas com 2, 4 ou 8 nós a situação inverte, sendo que a velocidade de leitura aumenta a partir de 8MB de dados lidos e se estabiliza a 16MB.

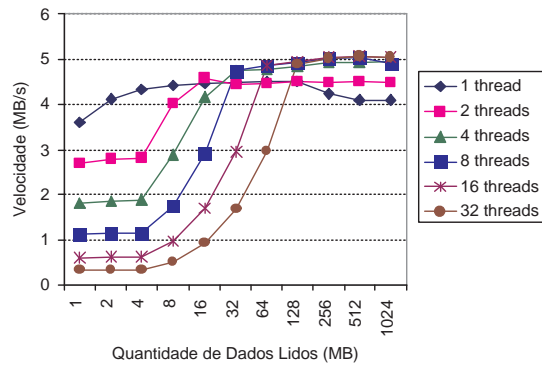
Com 4 threads (gráfico 5.7(c)), o Ext3 ganha para leitura até 4MB, mas perde do PVFS2 com mais de dois nós de dados a partir de 8MB. Nesse caso, o PVFS2 começa a aumentar a velocidade a partir de 4MB de dados para leitura, voltando a estabilizar aos 64MB (ou mais) de dados lidos.

Para 8 threads (gráfico 5.7(d)), 16 threads (gráfico 5.7(e)) e 32 threads (gráfico 5.7(f)) a análise é análoga, variando somente a quantidade de dados inicial, para o PVFS2 começar a escalar, e a final, quando o seu desempenho estabiliza. Note que as curvas de aumento de desempenho deixam de crescer exponencialmente em 16MB, 32MB, 64MB e 128MB, respectivamente para 4, 8, 16 e 32 threads.

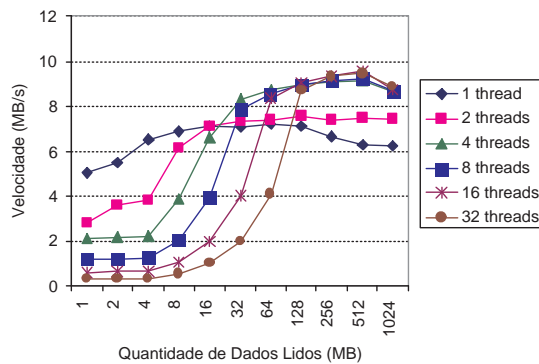
Ou seja, ao dobrar a quantidade de threads de leitura, a quantidade de dados total a serem lidos



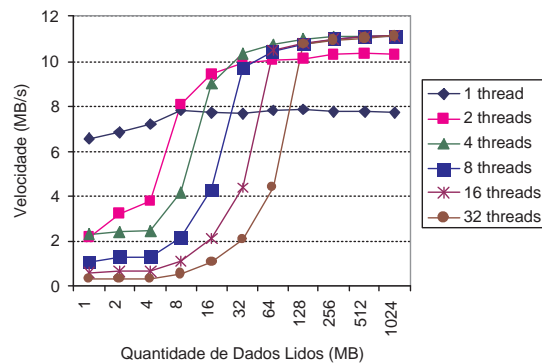
(a)



(b)

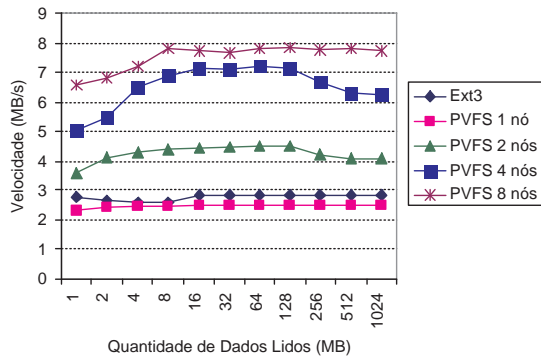


(c)

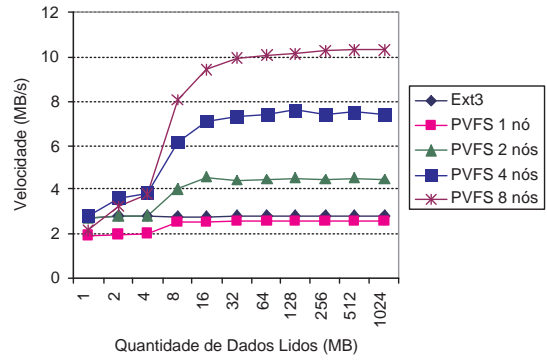


(d)

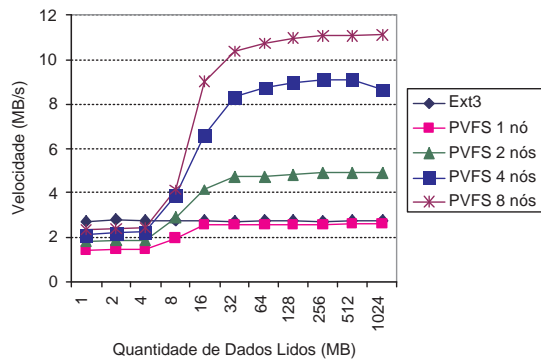
Figura 5.6: Desempenho variando a quantidade de dados lidos, para bloco de leitura de 64KB, usando sistema de arquivos PVFS2 com (a) 1, (b) 2, (c) 4 e (d) 8 nós de dados, modo PIO 0



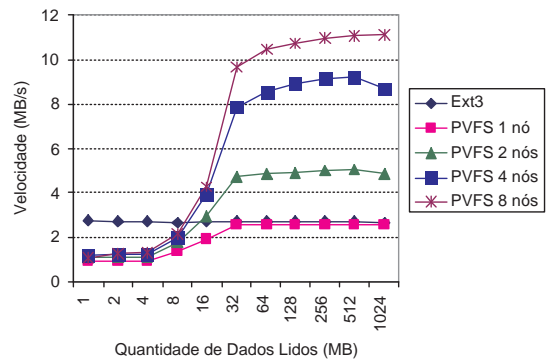
(a)



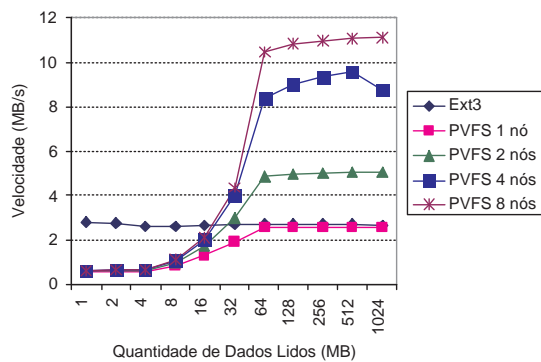
(b)



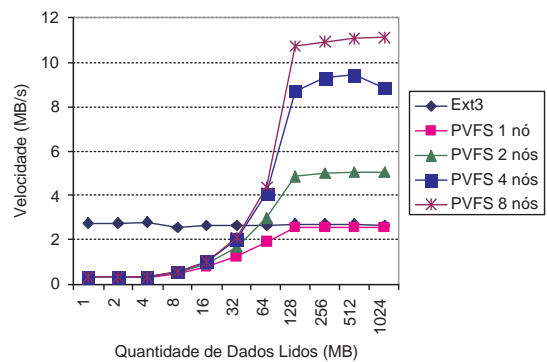
(c)



(d)



(e)



(f)

Figura 5.7: Desempenho variando a quantidade de dados lidos, para blocos de dados de 64KB, usando (a) 1, (b) 2, (c) 4, (d) 8, (e) 16 e (f) 32 threads, modo PIO 0

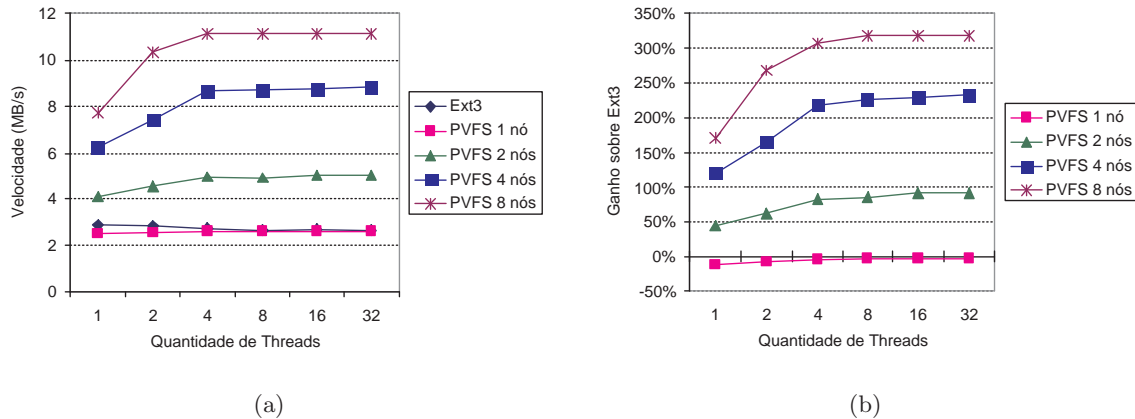


Figura 5.8: Desempenho (a) e ganho percentual sobre o Ext3 (b) ao aumentar concorrência no acesso, lendo 1GB de dados, blocos de dados de 64KB, modo PIO 0

para se manter o desempenho deve dobrar também. Isso é notado mais claramente no gráfico 5.6, onde percebe-se que não compensa aumentar muito a concorrência na leitura para uma certa configuração do PVFS2, sem que haja dados suficientes a serem lidos para compensar essa sobrecarga.

Dessa forma, para simplificar a análise, e já levando em consideração a influência causada pela variação da quantidade de dados a serem lidos sobre o grau de concorrência presente nos testes, todos os testes a seguir serão baseados na leitura de arquivos de tamanho fixo de 1GB, pois nosso foco é avaliar o desempenho dos sistemas dois sistemas para grandes quantidades de informações, sob alta concorrência.

### 5.3.3 Variação do Número de Servidores

Uma das formas de melhorar o desempenho do PVFS2 é aumentar a quantidade de servidores de dados. Essa melhora ocorre pois a banda disponibilizada pelos discos para a saída de dados é agregada, assim como a banda de saída pela rede, o que melhora o desempenho do serviço para um número crescente de clientes. Além disso, ter mais servidores significa ter mais memória disponível para cache de dados, o que ajuda no desempenho.

É interessante notar que essas e outras vantagens só serão realmente importantes caso existam muitos clientes acessando muitos servidores. Porém, foi observado em nossos testes que o desempenho também melhora quando se tem apenas um cliente os acessando.

O gráfico 5.8(a) mostra que conforme se aumenta a quantidade de servidores de dados, o PVFS2 passa a ser muito mais eficiente que o sistema de arquivos local, mesmo tendo apenas um cliente o acessando.

Porém, com apenas um servidor de dados o Ext3 é mais eficiente. Isso acontece pois, no acesso a sistemas de arquivos remotos, tem-se dois principais gargalos: velocidade do disco e banda disponível na rede. Ao utilizar apenas um servidor de dados se tem como limite a velocidade do disco dessa máquina, que é igual à velocidade do disco do sistema de arquivos local do cliente (vale

lembrar que os dados dos arquivos lidos não estão em cache, o que poderia ajudar no desempenho).

Já com dois ou mais servidores, o limite de saída de dados é a soma das velocidades desses discos, pois eles podem enviar dados em paralelo para o cliente, conforme a quantidade de requisições aumenta.

O gráfico 5.8(b) mostra o ganho de desempenho do PVFS2 em comparação com o Ext3, para as várias configurações de número de servidores de dados usadas nos testes. Com apenas um servidor, o PVFS2 fica abaixo do desempenho do Ext3 (de 3% a 12% mais lento), pelos motivos já apresentados, mas com dois servidores chega a ter quase o dobro da velocidade (de 44% a 90% de ganho), com quatro passa do triplo da velocidade (de 120% a 233% de ganho) e com oito passa do quádruplo da velocidade do Ext3 (de 171% a 318% de ganho).

Nota-se, por exemplo, que o ganho de desempenho esperado com quatro servidores deveria ser de até 400% (devido à velocidade média agregada dos discos), mas isso não é atingido por razões como:

#### Pedidos em paralelo a partir de um único cliente

Mesmo paralelizando as requisições, o cliente está limitado a apenas um processador. O tempo para realizar o tratamento concorrente das requisições depende muito da arquitetura da máquina cliente e em como ela lida com muitas tarefas simultâneas.

Alguns dos pedidos concorrentes acabam caindo em um mesmo servidor de dados, ao mesmo tempo

Como não se tem o controle sobre qual servidor cada uma das threads está acessando a cada instante, alguns servidores podem receber uma carga maior que outros, devido à concorrência no seu acesso. Isso ocorre devido aos blocos de dados dos arquivos estarem armazenados seqüencialmente nos servidores de dados, o que é uma característica do PVFS2. Assim, se o bloco  $i$  está armazenado no servidor  $i \bmod n$ , sendo  $n$  o número de servidores, então o bloco  $i + 1$  estará no servidor  $(i + 1) \bmod n$ .

Como os dados são lidos seqüencialmente a partir de uma posição do arquivo, existe uma grande chance de duas ou mais threads “caminharem” pelos mesmos servidores de dados por um certo tempo.

#### O tempo de processamento do cliente

Envolve o recebimento da requisição vinda de uma aplicação, a decisão de qual servidor de dados contactar (o que pode envolver acesso ao servidor de meta-dados para obter a resposta), a montagem do pedido dos dados para ser enviada ao servidor, o seu envio propriamente dito, o recebimento da resposta do servidor, a compreensão e validação dessa resposta e o envio dos dados recebidos para a aplicação que os requisitou.

#### O tempo de processamento dos servidores

Envolve o recebimento da requisição vinda do cliente, a compreensão e validação da mensagem, o pedido ao sistema de arquivos local pelos dados, a montagem do pacote e o seu envio pela rede.

### 5.3.4 Variação do Grau de Paralelismo

A grande vantagem divulgada sobre o PVFS2 é seu alto desempenho quando há muito acesso concorrente ao sistema de arquivos, vindo de vários clientes para vários servidores. Foi visto no capítulo anterior que o PVFS e seus variantes conseguem utilizar toda a banda disponível da rede para aumentar o desempenho de forma agregada, lidando com múltiplas requisições de dados de forma concorrente.

No nosso caso há apenas um cliente acessando vários servidores simultaneamente, o que não deve causar impacto nos servidores, mas pode prejudicar o cliente. Isto é, o cliente deve conseguir tratar os pedidos concorrentes das várias threads ou processos de forma eficiente, para não se tornar um gargalo, tanto para a realização dos pedidos aos servidores como para o recebimento e distribuição dos dados.

O gráfico 5.8(a) mostra que existe um aumento no desempenho do PVFS2 quando o nível de concorrência no acesso aumenta, especialmente quando há mais servidores de dados disponíveis. Nota-se que de 1 a 4 threads o desempenho aumenta e a partir daí se mantém, o que não ocorre com o Ext3, que sofre uma pequena degradação na velocidade de transferência nessa faixa.

### 5.3.5 Apenas Um Processo, Sem Acesso Concorrente

Conforme analisado até aqui, o acesso concorrente ao PVFS2 é muito mais eficiente que o mesmo tipo de acesso ao Ext3. Além disso, nota-se que quando o PVFS2 possui mais de 2 servidores de dados, acessá-lo de forma não-concorrente ou mono-tarefa também possui desempenho superior.

No gráfico 5.7(a) percebe-se isso mais claramente. Ele mostra o acesso ao PVFS2 com uma quantidade variável de servidores de dados contra o acesso ao Ext3, utilizando-se de apenas uma thread para a leitura dos dados. Com isso surgiram duas hipóteses que ocasionariam esse ganho inesperado de desempenho no PVFS2:

1. A existência de algum tipo de cache controlado internamente pelos servidores PVFS2;
2. O tamanho do bloco de leitura é muito grande e o cliente PVFS2 estaria contactando os servidores em paralelo.

Para desvendar ambos os casos, foi necessário recorrer à documentação da arquitetura do PVFS2 [Tea03]. Segundo ela, tanto os servidores quanto os clientes PVFS2 possuem caches internos, que melhoram o desempenho evitando tráfego de dados pela rede. Porém, por padrão a instalação do PVFS2 desativa tais caches, pois em caso de queda de alguma máquina, os dados que foram confirmados como gravados pelos servidores podem ser perdidos.

Quanto ao segundo caso, a arquitetura do PVFS2 foi explicitamente projetada para realizar tarefas em paralelo ou assincronamente sempre que possível. Esse pode ser um indício de que o cliente PVFS2 esteja realizando pedidos em paralelo aos servidores, especialmente quando o bloco de dados requisitado estiver em mais de um servidor.

Além disso, esse cliente pode receber as várias requisições e realizá-las de forma assíncrona, isto é, enviar os pedidos aos servidores o quanto antes e distribuir às threads os dados recebidos assim que chegarem, ao invés de enviar um pedido a um servidor e aguardar o recebimento da resposta, para depois processar o próximo pedido.

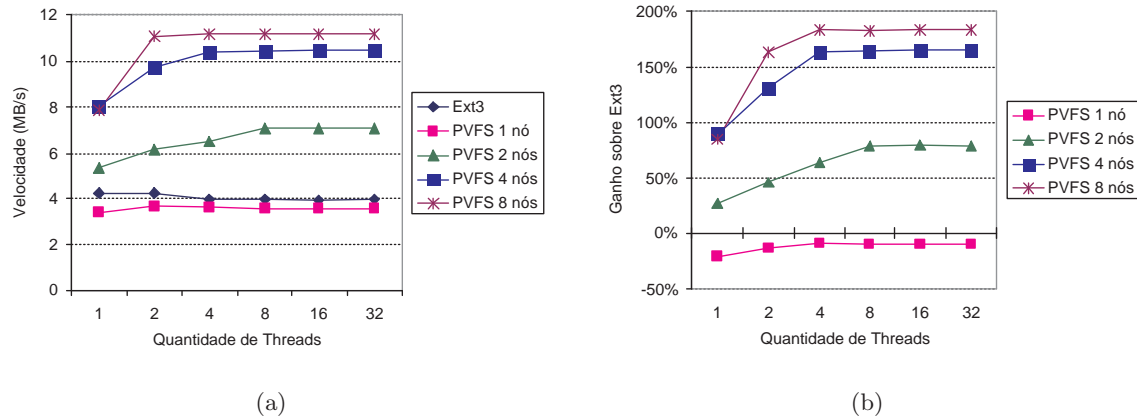


Figura 5.9: Desempenho (a) e ganho percentual sobre o Ext3 (b) ao aumentar concorrência no acesso, lendo 1GB de dados, bloco de leitura de 64KB, modo PIO 1

### 5.3.6 Variação na Velocidade dos Discos

A última variável usada nos testes, e também a mais importante, é a velocidade dos discos. É a partir dela que definiremos até quando vale a pena usar o PVFS2 no lugar no Ext3, pois conforme a velocidade média agregada dos discos se aproxima da velocidade máxima da rede, o PVFS2 começa a ter como gargalo a rede ao invés dos discos, dando vantagem para o Ext3.

Conforme já comentado, foram usados os modos PIO para realizar a variação da velocidade dos discos. Nos gráficos 5.9, 5.10 e 5.11 encontra-se o desempenho do PVFS2 e do Ext3, além do ganho que o PVFS2 teve sobre o Ext3, para as suas várias combinações de servidores e velocidades de disco.

Note que o PVFS2 e o Ext3 melhoram o desempenho conforme se aumenta a velocidade dos discos. Perceba também que a velocidade do PVFS2 não aumenta na mesma proporção que o Ext3. Isso ocorre porque o Ext3 está ligado diretamente ao disco, ou seja, o aumento de desempenho é proporcional.

Note também que o comportamento já discutido nas seções anteriores se manteve, independentemente do aumento da velocidade dos discos, embora perceba-se que o PVFS2 comece a dar sinais de que não conseguirá se manter a frente do Ext3 caso a velocidade dos discos continue aumentando.

## 5.4 Análise dos Resultados de Escrita

Além dos testes de leitura, foram realizados testes de escrita sob as mesmas condições, envolvendo as mesmas variáveis, no mesmo ambiente, para analisar o comportamento no caso de aplicações que realizam muitas escritas concorrentes.

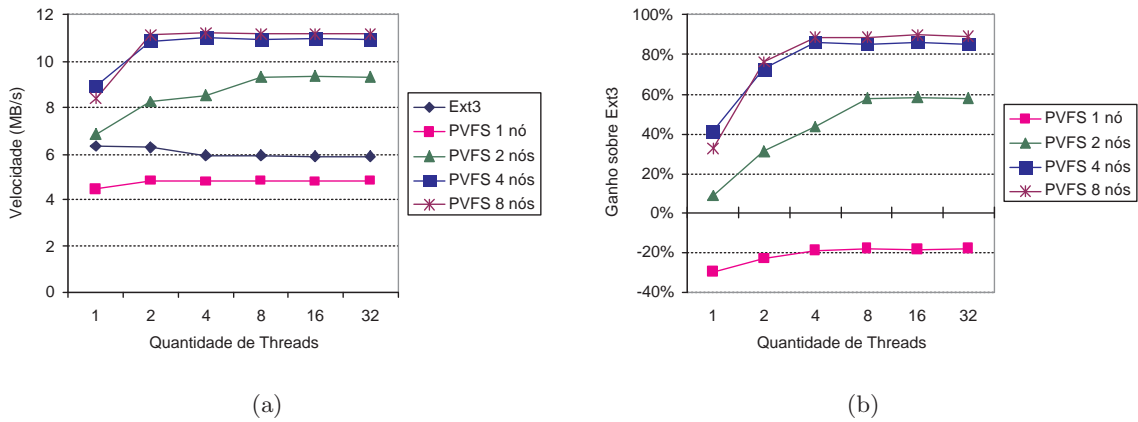


Figura 5.10: Desempenho (a) e ganho percentual sobre o Ext3 (b) ao aumentar concorrência no acesso, lendo 1GB de dados, bloco de leitura de 64KB, modo PIO 2

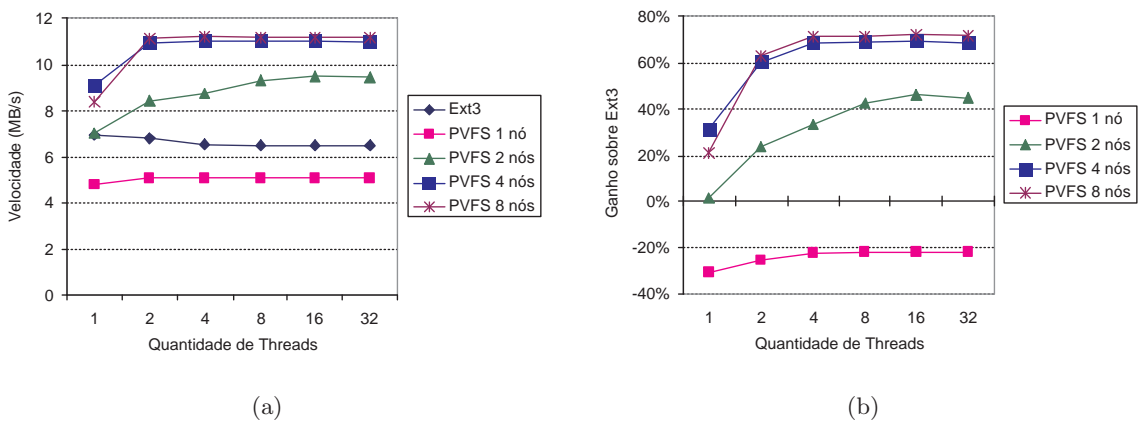


Figura 5.11: Desempenho (a) e ganho percentual sobre o Ext3 (b) ao aumentar concorrência no acesso, lendo 1GB de dados, bloco de leitura de 64KB, modo PIO 3

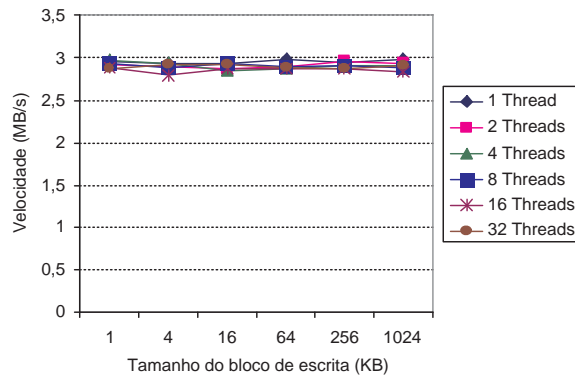


Figura 5.12: Desempenho do Ext3 variando o tamanho do bloco de escrita, para 1GB de dados, modo PIO 0

#### 5.4.1 Variação do Tamanho do Bloco de Escrita

Assim como foi observado nos testes de leitura, o tamanho do bloco de dados usados para escrita também não influenciaram o desempenho do Ext3 e nem do PVFS2, conforme nota-se nos gráficos das figuras 5.12 e 5.13, respectivamente. Dessa forma, para simplificar a análise dos resultados, todos os testes usam tamanho de bloco de escrita igual a 64KB.

#### 5.4.2 Variação do Tamanho do Arquivo

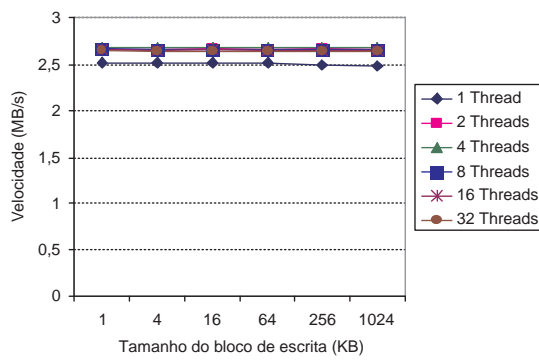
Ao variar o tamanho do arquivo a ser escrito (isto é, a quantidade de dados gravadas no sistema de arquivos), nota-se que quanto maior a quantidade de dados, melhor o desempenho do PVFS2 (veja gráficos da figura 5.15). Já o Ext3 possui um desempenho que varia muito quando a quantidade de dados é pequena e que vai se estabilizando conforme ela aumenta (veja gráfico 5.14).

Essa variação brusca de desempenho do Ext3 ocorre devido ao comportamento do próprio sistema de arquivos, que antes de enviar os dados ao disco, procura deixá-los em memória, para tornar o sistema como um todo mais rápido. Posteriormente, os caches que estiverem “sujeitos” são enviados ao disco para persistir suas informações.

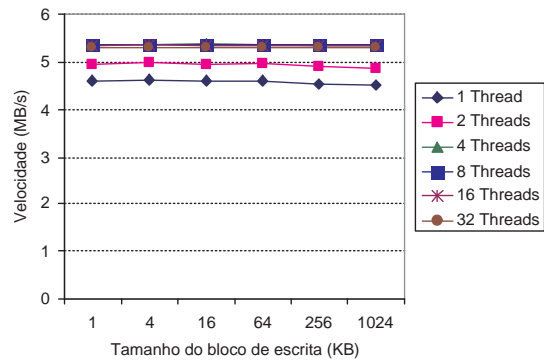
Uma forma de se evitar essa otimização seria a chamada da função de sistema `fflush` para um determinado arquivo, que forçaria a gravação dos dados no sistema de armazenamento local. Porém, realizamos testes chamando essa função após cada escrita, mas não alterou os resultados. Pesquisando em algumas listas de discussão<sup>3</sup> foi identificado que funções como `fflush` e `fsync` não foram implementadas no núcleo do Linux 2.6 (isto é, elas existem, mas não fazem nada), devido ao fato da implementação da escrita dos dados não persistí-los na ordem em que as chamadas de escrita ocorreram. Isso explica a semelhança entre os resultados.

Quando muitos dados são escritos, o efeito do cache é reduzido, pois não existe espaço suficiente na memória para alocar todos os dados escritos. Isso explica a perda de desempenho quando se

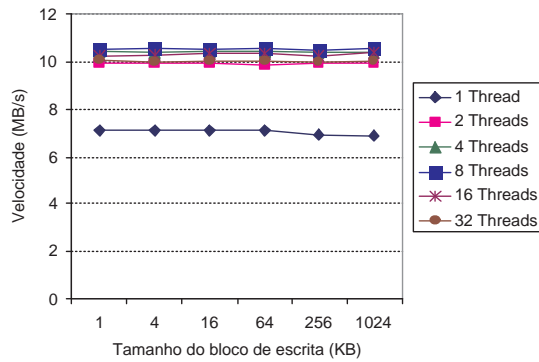
<sup>3</sup>[http://groups.google.com.br/group/linux.kernel/browse\\_frm/thread/d343e51655b4ac7c/](http://groups.google.com.br/group/linux.kernel/browse_frm/thread/d343e51655b4ac7c/), março de 2004



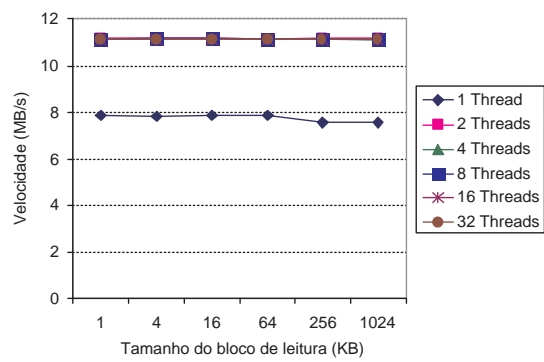
(a)



(b)



(c)



(d)

Figura 5.13: Desempenho do PVFS2 usando (a) 1, (b) 2, (c) 4 e (d) 8 nós de dados, variando o tamanho do bloco de escrita, considerando 1GB de dados gravados, modo PIO 0

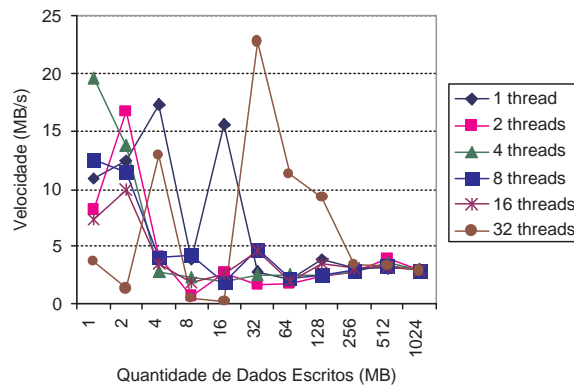


Figura 5.14: Desempenho variando a quantidade de dados gravados, para bloco de leitura de 64KB, sistema de arquivos Ext3, modo PIO 0

tem mais de 512MB de dados para serem escritos.

Nos servidores do PVFS2 ocorre a mesma situação, pois o armazenamento local dos dados é realizado utilizando-se do Ext3. Mas, esse ganho de desempenho é limitado pela velocidade da rede, pois os dados a serem gravados só serão colocados nos caches de escrita dos servidores.

Nos gráficos 5.16 nota-se que o Ext3 varia muito o desempenho para gravar menos do que 64MB e, conforme a quantidade de dados aumenta, seu desempenho vai caindo, enquanto que o PVFS2 vai melhorando. Nota-se também que o aumento do nível de concorrência não afeta muito nenhum dos sistemas de arquivos considerados.

Assim, optou-se por realizar os demais testes utilizando-se sempre de 1GB de dados a serem gravados, o que está coerente com o foco de nossa pesquisa, que envolve análise de desempenho para uma quantidade grande de dados. Isso simplifica a análise, além de evitar que o cache do sistema de arquivos local interfira demais nos resultados, gerando assim uma comparação mais equilibrada.

### 5.4.3 Variação do Número de Servidores

Aumentar o número de servidores de dados do PVFS2 melhora o desempenho da escrita no sistema de arquivos, da mesma forma como foi observado na leitura e pelas mesmas razões (aumento da velocidade agregada dos discos, aumento da quantidade de memória cache dos servidores, entre outros).

O gráfico 5.17(a) mostra que ao aumentar a quantidade de servidores de dados, o PVFS2 melhora seu desempenho com relação ao sistema de arquivos local, mesmo tendo apenas um cliente o acessando.

Porém, assim como na leitura, com apenas um servidor de dados o Ext3 é mais eficiente. Isso acontece pela mesma razão já apresentada, isto é no acesso a sistemas de arquivos remotos tem-se dois principais gargalos: velocidade do disco e banda disponível na rede. Já com dois ou mais servidores, a banda de saída de dados aumenta, sendo o resultado da soma das velocidades dos

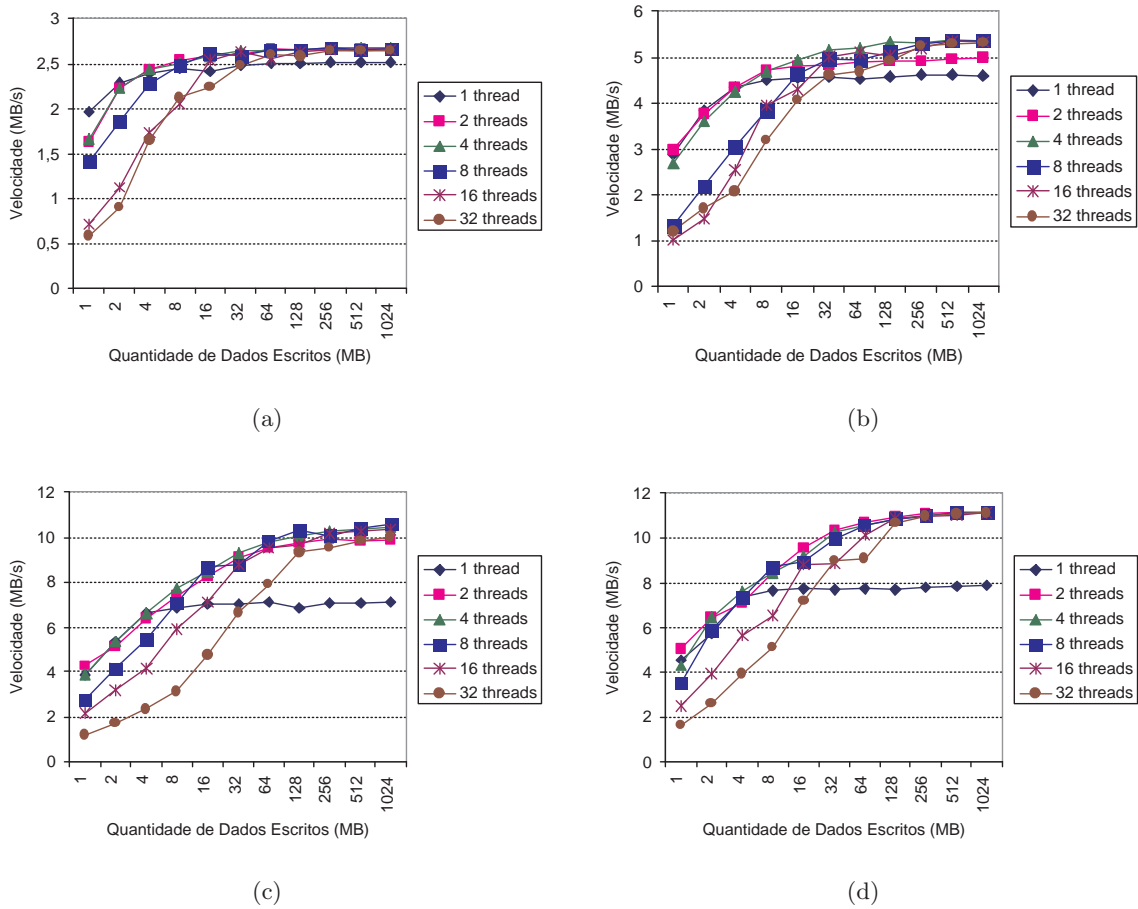
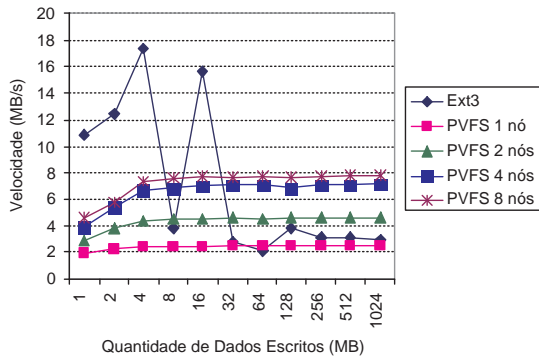
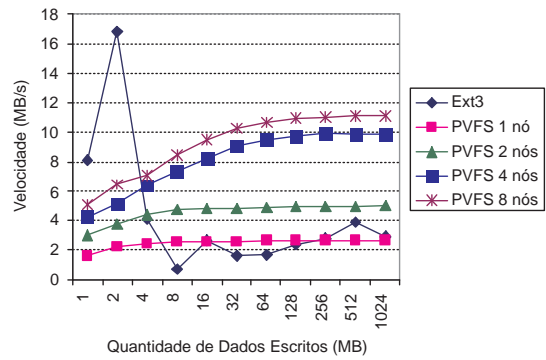


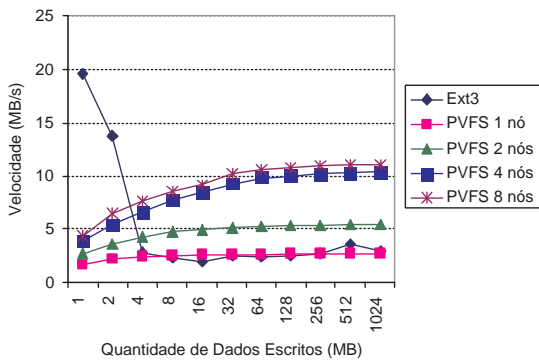
Figura 5.15: Desempenho variando a quantidade de dados gravados, para bloco de escrita de 64KB, usando sistema de arquivos PVFS2 com (a) 1, (b) 2, (c) 4 e (d) 8 nós de dados, modo PIO 0



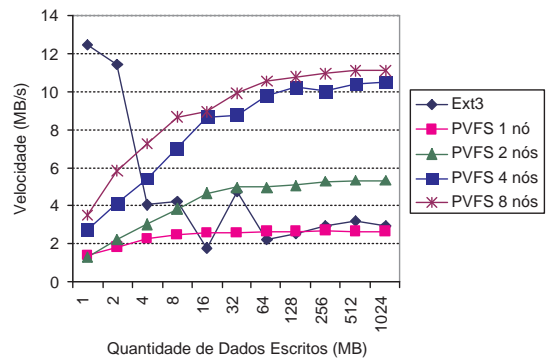
(a)



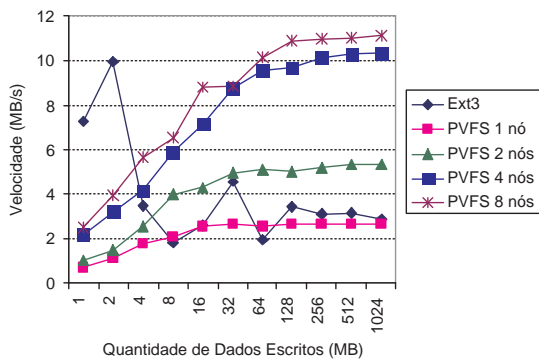
(b)



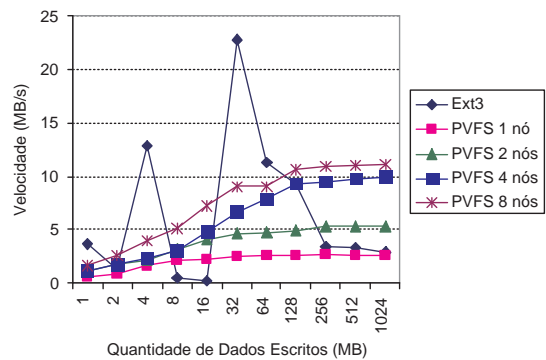
(c)



(d)



(e)



(f)

Figura 5.16: Desempenho variando a quantidade de dados gravados, para bloco de escrita de 64KB, usando (a) 1, (b) 2, (c) 4, (d) 8, (e) 16 e (f) 32 threads, modo PIO 0

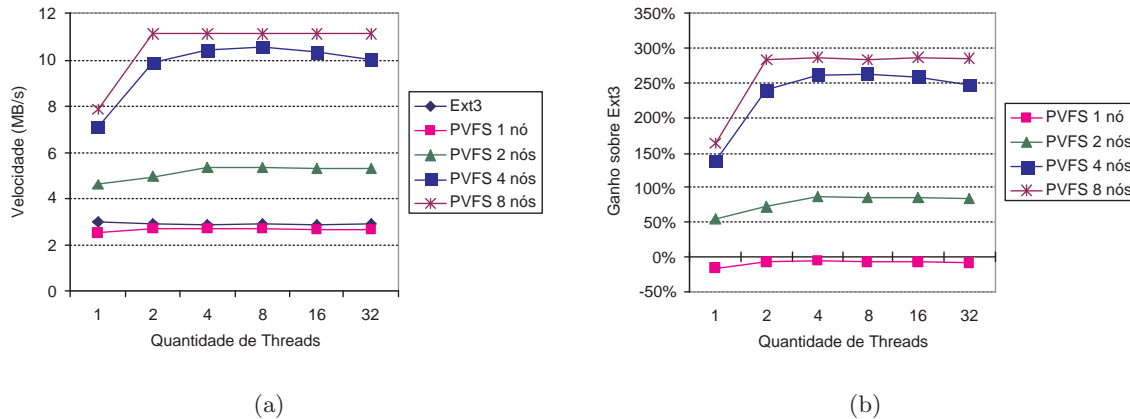


Figura 5.17: Desempenho (a) e ganho percentual sobre o Ext3 (b) ao aumentar concorrência no acesso, escrevendo 1GB de dados, blocos de 64KB, modo PIO 0

discos dos servidores.

O gráfico 5.17(b) mostra o ganho de desempenho na escrita usando o PVFS2 em comparação com o Ext3. Com apenas um servidor, o PVFS2 fica abaixo do desempenho do Ext3 (de 8% a 16% mais lento), mas a partir de dois servidores a situação se inverte e o PVFS2 passa a ser de 55% a 84% mais rápido. Com quatro servidores ele é de 139% a 264% mais rápido e com oito o ganho passa a ser de 164% a 287%.

#### 5.4.4 Variação do Grau de Paralelismo

Ao aumentar o acesso concorrente, o PVFS2 mostrou que lida melhor com os vários pedidos de escrita do que o Ext3. Nota-se no gráfico 5.17(a) que de 1 a 4 threads o desempenho do PVFS2 aumenta, se mantendo a partir daí. Já o Ext3 sofre uma pequena degradação em seu desempenho conforme se aumenta a quantidade de threads.

Novamente, o mesmo resultado apresentado durante os testes de leitura se repete para a escrita, mostrando que pode-se contar com esse sistema de arquivos paralelo também para o caso de escrita concorrente em massa.

#### 5.4.5 Apenas Um Processo, Sem Acesso Concorrente

O acesso concorrente ao PVFS2 na leitura é muito mais eficiente que no Ext3. Isso também é notado na escrita de muitos dados. Porém, nota-se que quando o PVFS2 possui mais de 2 servidores de dados, a escrita de mais de 32MB de forma não-concorrente (ou mono-tarefa) possui desempenho muito melhor que o Ext3.

Observa-se essa constatação no gráfico 5.16(a). Veja que ao aumentar a quantidade de servidores, o desempenho para apenas 1 thread aumenta, ficando acima do Ext3. As razões para que esse resultado ocorresse são:

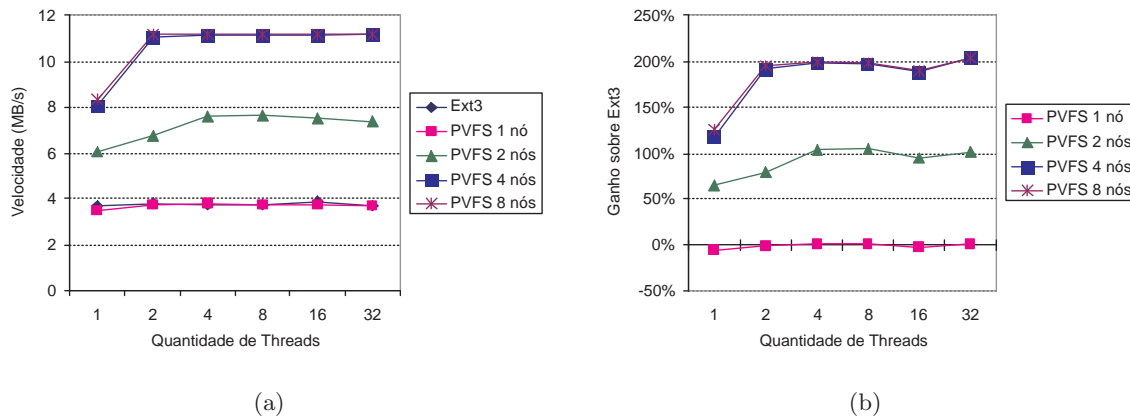


Figura 5.18: Desempenho (a) e ganho percentual sobre o Ext3 (b) ao aumentar concorrência no acesso, escrevendo 1GB de dados, blocos de 64KB, modo PIO 1

1. A existência do cache de escrita do Ext3 dos servidores de dados;
2. O envio de requisições de escrita assíncronamente para os vários servidores de dados.

Isso é confirmado pela documentação de arquitetura do PVFS2 [Tea03], que utiliza a mesma estratégia de se realizar tarefas em paralelo e assincronamente, tanto para leitura quanto para escrita. No caso da escrita, não há transação, mas o PVFS2 nos garante que a gravação dos dados é atômica, evitando que um pedido de leitura retorne dados inconsistentes.

#### 5.4.6 Variação na Velocidade dos Discos

Na escrita observa-se o mesmo comportamento que na leitura quando se varia a velocidade média dos discos. Nos gráficos 5.18, 5.19 e 5.20 têm-se o desempenho do PVFS2 e do Ext3, além do ganho que o PVFS2 teve sobre o Ext3, para as suas várias combinações de servidores e velocidades de disco.

Note que o PVFS2 e o Ext3 melhoram o desempenho conforme a velocidade dos discos aumenta, de forma proporcional, sendo que o PVFS2 está limitado à banda disponível da rede. O comportamento já discutido nas seções anteriores se manteve, independentemente do aumento da velocidade dos discos.

### 5.5 Conclusões Gerais

A partir das análises individuais de cada teste, realizadas nas seções anteriores, é possível chegar a algumas conclusões gerais que nos ajudarão a atingir nossa proposta, descritas em cada uma das seções a seguir.

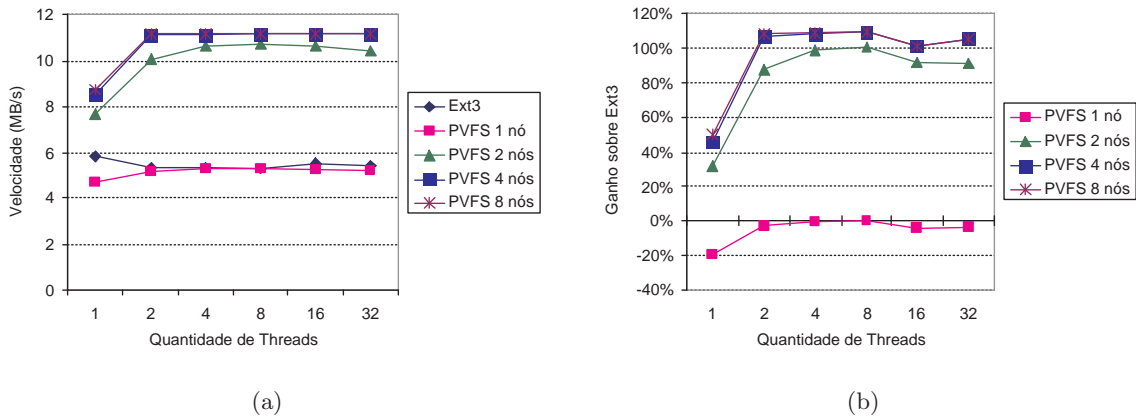


Figura 5.19: Desempenho (a) e ganho percentual sobre o Ext3 (b) ao aumentar concorrência no acesso, escrevendo 1GB de dados, blocos de 64KB, modo PIO 2

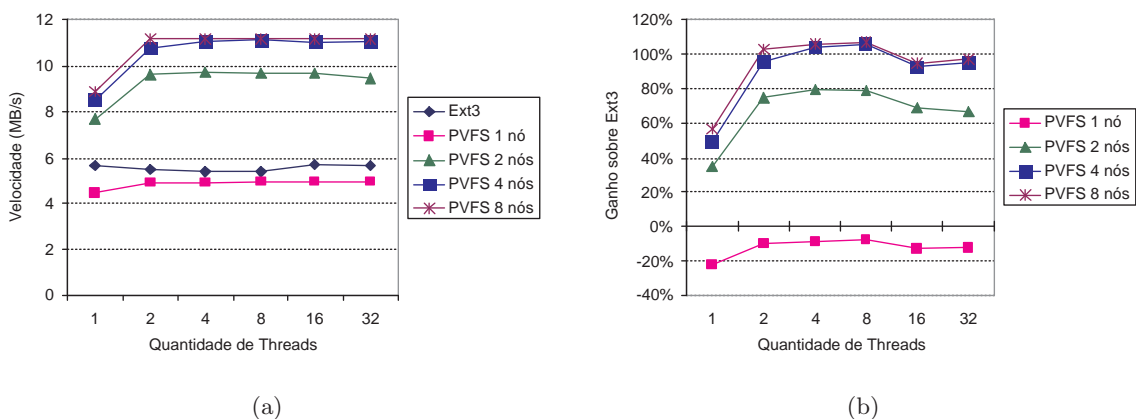


Figura 5.20: Desempenho (a) e ganho percentual sobre o Ext3 (b) ao aumentar concorrência no acesso, escrevendo 1GB de dados, blocos de 64KB, modo PIO 3

### 5.5.1 Rede Rápida É Mais Importante Que Discos Rápidos

Com base nos resultados apresentados nas seções anteriores, verificou-se que o PVFS2 é mais eficiente que o Ext3 quando se tem uma rede suficientemente mais rápida que os discos, especialmente quando o acesso concorrente aumenta, conforme pode-se verificar nos gráficos da figura 5.21. Nela verifica-se também que quando se tem uma quantidade grande de servidores de dados, o ganho sobre o Ext3 aumenta, mostrando que a velocidade média agregada dos discos deixa de ser um gargalo. Porém, com apenas um servidor o sistema de arquivos local é mais rápido que o PVFS2.

Além disso, conforme a velocidade dos discos aumenta, o PVFS2 não consegue acompanhar esse crescimento de forma proporcional, perdendo rendimento até ser limitado pela velocidade da rede, conforme constatado ao analisar os gráficos 5.22. Note também que o PVFS2 com 8 nós tem seu desempenho claramente limitado pela rede, até mesmo com os discos na sua velocidade média mais baixa (modo PIO 0) a partir de 2 threads.

Já o Ext3 melhora seu desempenho linearmente com o aumento da velocidade dos discos, não sendo tão afetado pelo nível de concorrência no seu acesso. A razão dessa diferença de comportamento é que no PVFS2 os discos dos servidores não conseguem dar vazão suficiente para as requisições dos clientes através da rede, o que não ocorre com o Ext3, que por ser um sistema de arquivos local não usa a rede, sendo limitado apenas à interface com o dispositivo de armazenamento.

Para a escrita, embora exista um cache de escrita que torna os testes um pouco menos precisos, observa-se que valem as mesmas constatações que foram feitas para a leitura, conforme nota-se nos gráficos 5.23 e 5.24.

Dessa forma, concluímos que a velocidade da rede é fundamental para que o PVFS2 possa ser mais eficiente que o Ext3. Além disso, para uma quantidade suficiente de servidores, a velocidade média dos discos não chega a ser muito preocupante para seu desempenho.

### 5.5.2 Aproveitamento da Banda dos Discos Não É Linear

Conforme comentado na seção anterior, aumentar o número de servidores de dados do PVFS2 significa aumento de desempenho, desde que a rede não se torne um gargalo (conforme verifica-se nos gráficos da figura 5.22). Porém, simplesmente aumentar o número de servidores e esperar que o desempenho dobre não é realista.

A análise das seções 5.3.3 e 5.4.3 mostra que conforme o número de servidores PVFS2 aumenta, a velocidade média de transferência dos dados entre servidores e cliente não cresce na mesma proporção. Além disso, a análise das seções 5.3.6 e 5.4.6 indica que o aumento da velocidade dos discos (através do modo PIO) não é refletido diretamente no PVFS2, mas somente no Ext3.

Essa constatação fica mais clara nos gráficos das figuras 5.25 e 5.26, que mostra o aproveitamento que esses sistemas de arquivos, em cada uma de suas configurações testadas, obtiveram sobre a velocidade agregada dos discos (conforme tabela da figura 5.2 na página 62).

Note que o PVFS2 não consegue manter o aproveitamento conforme se aumenta a quantidade de servidores e velocidade dos discos. Porém, pelos gráficos das figuras 5.27 e 5.28 (que nada mais são que outra visão dos dados apresentados nos gráficos 5.25 e 5.26 respectivamente) percebe-se

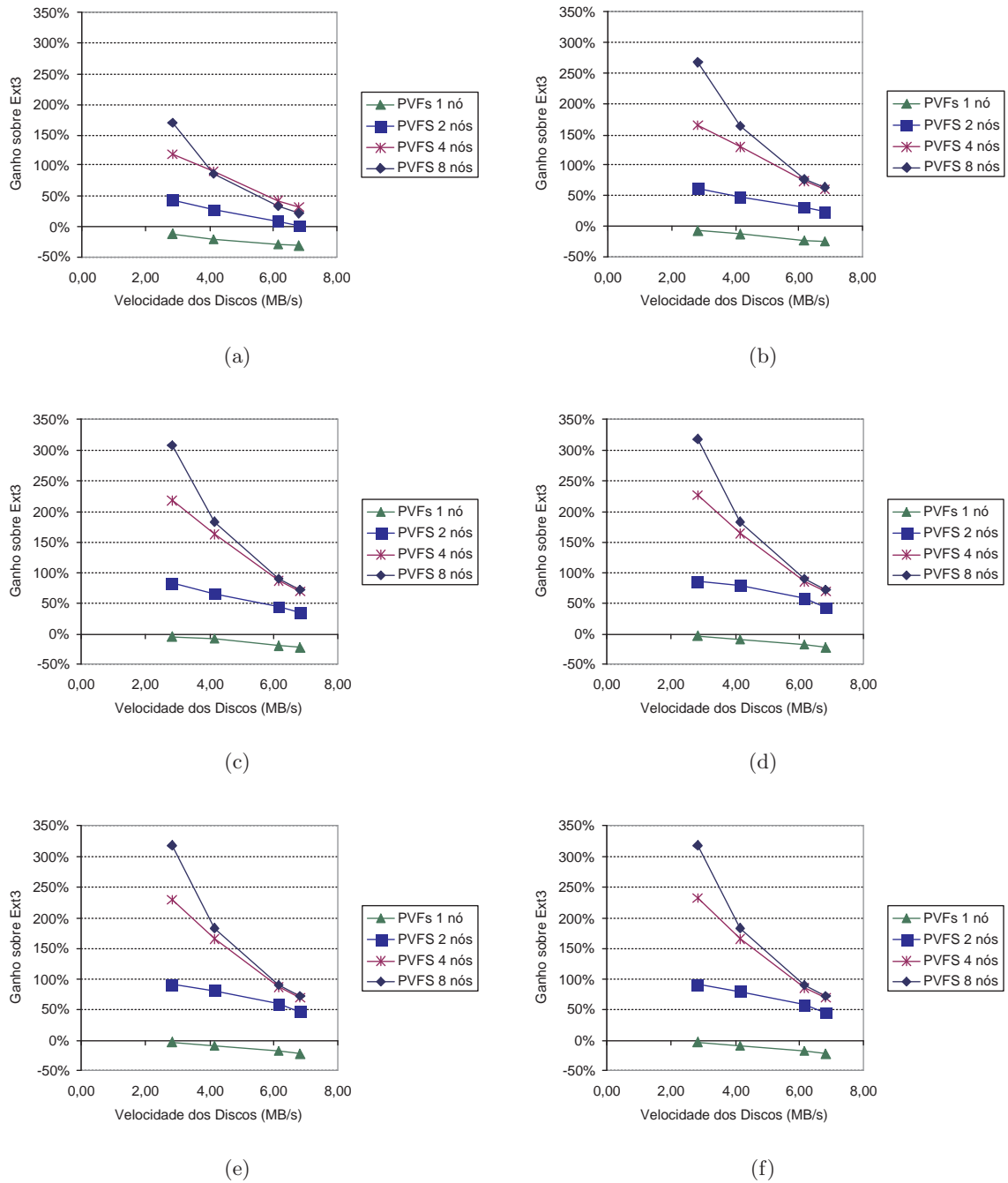


Figura 5.21: Ganho percentual do PVFS2 sobre o Ext3 para os vários modos PIO, lendo 1GB de dados, blocos de 64KB, usando (a) 1, (b) 2, (c) 4, (d) 8, (e) 16 e (f) 32 threads

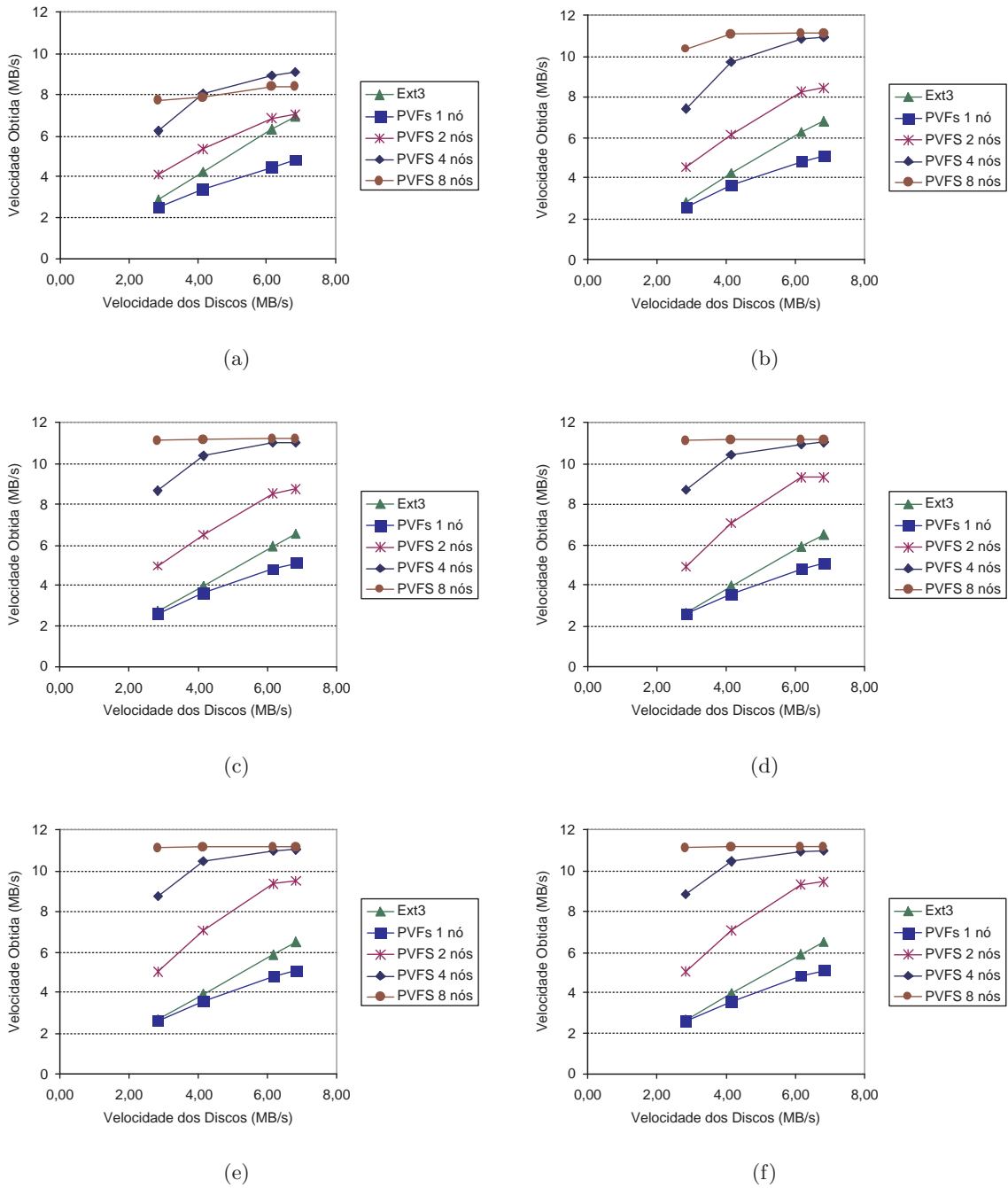


Figura 5.22: Velocidade do PVFS2 e do Ext3 para os vários modos PIO, lendo 1GB de dados, blocos de 64KB, usando (a) 1, (b) 2, (c) 4, (d) 8, (e) 16 e (f) 32 threads

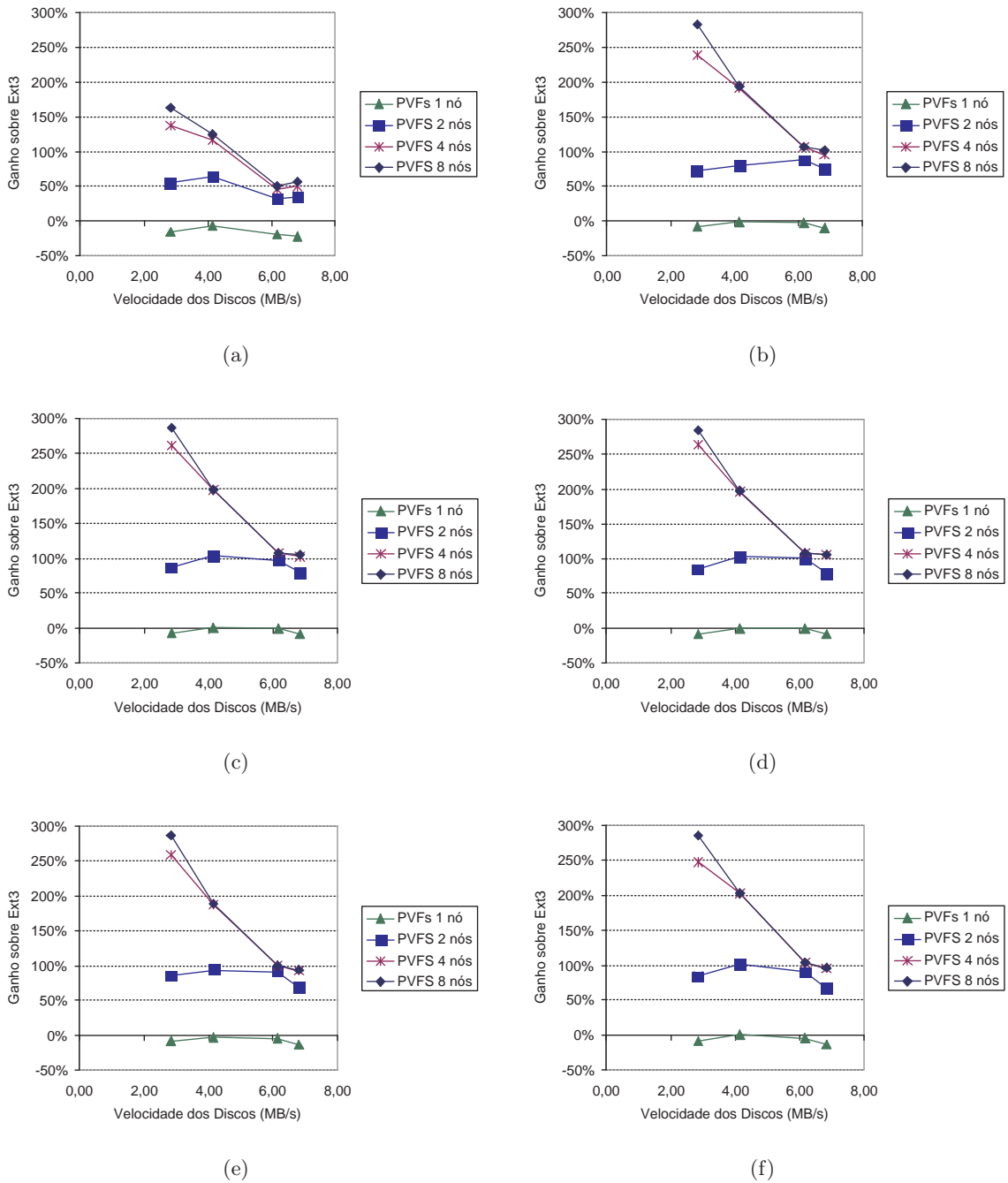


Figura 5.23: Ganho percentual do PVFS2 sobre o Ext3 para os vários modos PIO, escrevendo 1GB de dados, blocos de 64KB, usando (a) 1, (b) 2, (c) 4, (d) 8, (e) 16 e (f) 32 threads

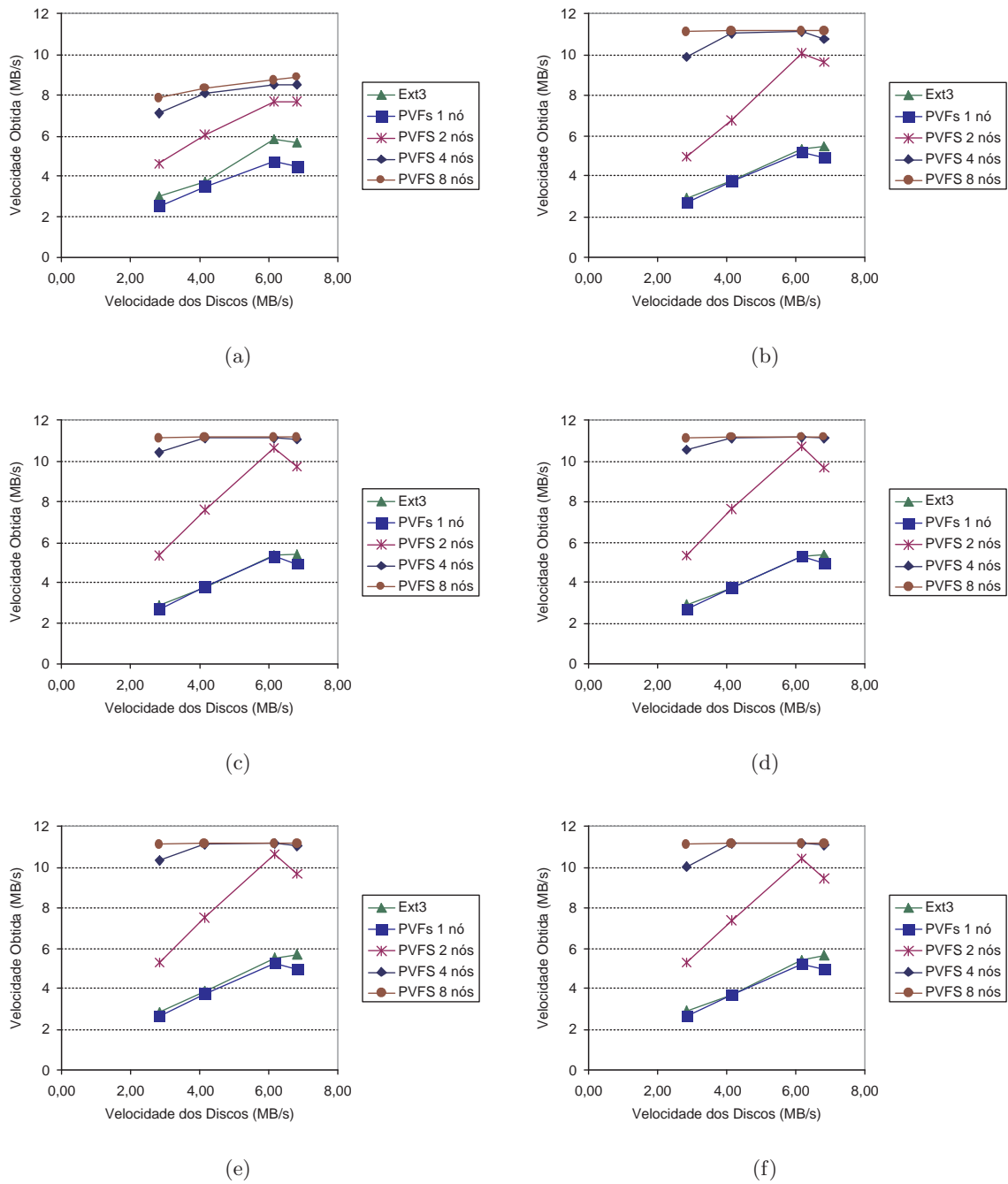


Figura 5.24: Velocidade do PVFS2 e do Ext3 para os vários modos PIO, escrevendo 1GB de dados, blocos de 64KB, usando (a) 1, (b) 2, (c) 4, (d) 8, (e) 16 e (f) 32 threads

que conforme a concorrência aumenta, há uma melhora no aproveitamento do PVFS2, enquanto que o Ext3 tem uma pequena queda.

Perceba que o aproveitamento da velocidade agregada dos discos dos servidores é um pouco maior na escrita do que na leitura. Isso é explicado pelo fato da escrita não utilizar diretamente o disco sempre, deixando os dados em memória cache para serem persistidos mais tarde, o que não ocorre na leitura, pois os caches foram limpos antes de se executar cada teste. Até mesmo a escrita realizada pelo Ext3 ficou mais rápida que seus resultados dos testes de leitura, razão à qual alguns resultados ultrapassam os 100% de aproveitamento.

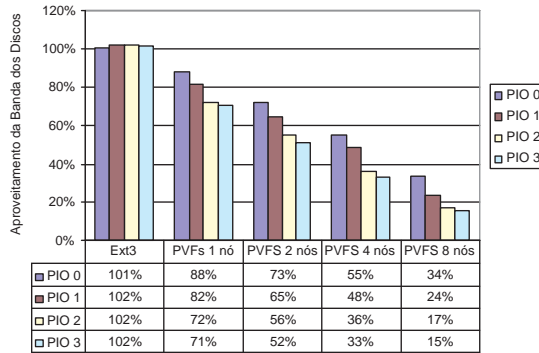
### 5.5.3 PVFS2 É Mais Rápido Que Ext3 Em Uma Rede Ethernet 1000Base-T

A partir das conclusões da seção anterior, conclui-se que o PVFS2 não utiliza toda a banda da rede para tráfego de dados. No caso dos testes desse capítulo, mesmo tendo 12,5MB/s de banda disponível, ele utilizou no máximo 92% dela (11,5MB/s com 8 servidores e 32 threads).

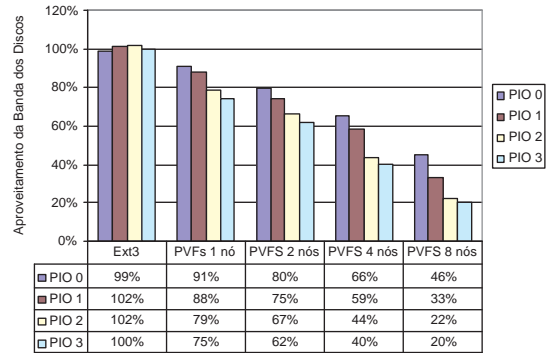
Assim, caso haja muito acesso concorrente a um sistema de arquivos a partir de uma mesma máquina, e caso sua rede seja pelo menos 8% mais rápido que seus discos (já que o PVFS2 não conseguiu utilizar mais do que 92% dela), vale a pena utilizar o PVFS2 no lugar do Ext3. Note que isso é apenas uma estimativa que considera as conclusões das seções anteriores, isto é, o desempenho do Ext3 acompanha a velocidade do disco linearmente, enquanto que o PVFS2 está limitado pela rede. A precisão desse cálculo pode variar dependendo de fatores não considerados, como latência da rede, sobrecarga no protocolo de comunicação, entre outros.

Dessa forma, pode-se estimar grosseiramente, através de uma simples regra de três, que em uma rede Ethernet 1000Base-T (125MB/s), os discos não poderiam ter mais do que 115MB/s de velocidade média, para assim valer a pena usar o PVFS2 no lugar do Ext3.

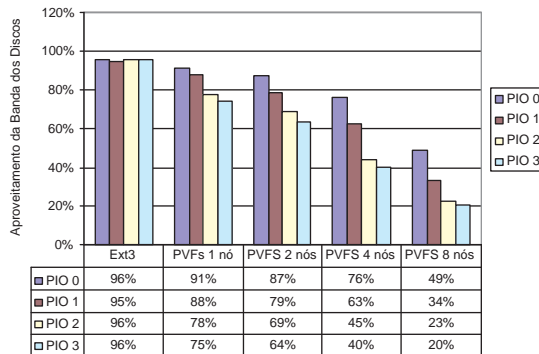
Realizando uma estimativa linear a partir dos resultados aqui apresentados, concluímos que se a rede aumentar sua velocidade de 12,5MB/s para 125MB/s (ou de 100Mbits/s para 1Gbits/s), os resultados vistos com discos de 2,8MB/s (análogo ao modo PIO 0 dos nossos testes) seriam muito similares se usássemos discos de 28MB/s. Ou seja, usando 8 servidores de dados, o PVFS2 teria uma velocidade próxima a  $28 \times 8 \times 49\%$  (de aproveitamento) = 109MB/s. E usando discos de 68MB/s (análogo ao modo PIO 3 dos nossos testes) teria-se uma velocidade de  $68 \times 8 \times 20\%$  = 109MB/s. Nesse caso, os resultados seriam idênticos devido à limitação da velocidade da rede.



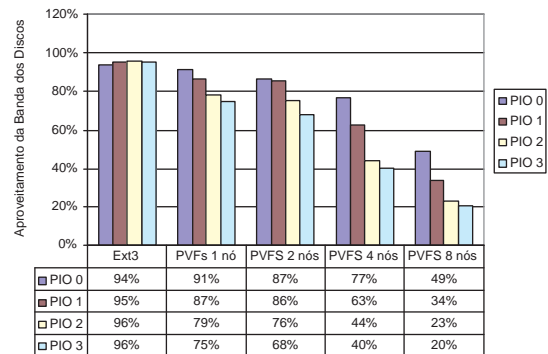
(a)



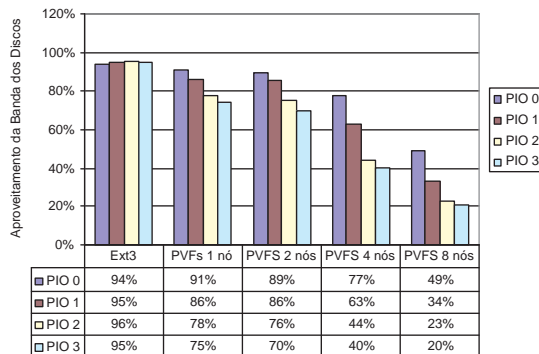
(b)



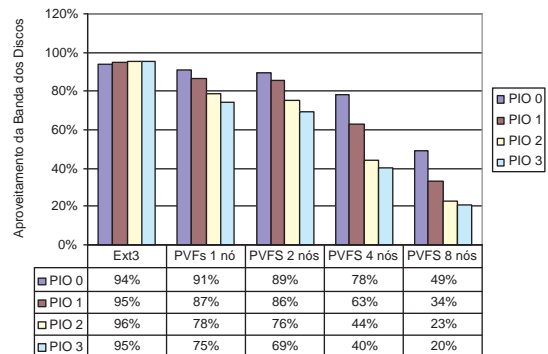
(c)



(d)

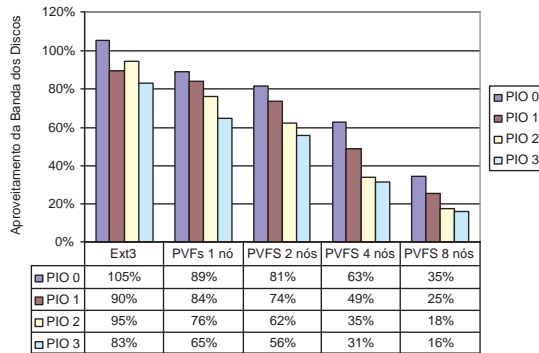


(e)

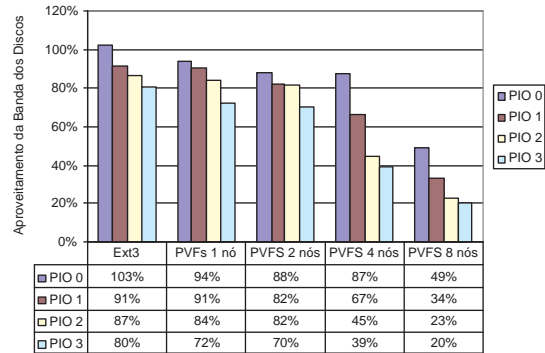


(f)

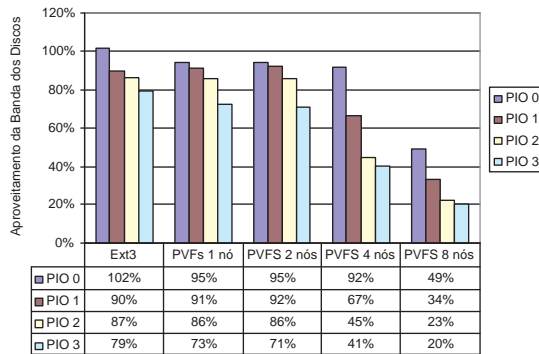
Figura 5.25: Aproveitamento que o Ext3 e o PVFS2 tiveram, relativo à velocidade média agregada disponibilizada pelos discos dos servidores, para os vários modos PIO, lendo 1GB de dados, blocos de 64KB, usando (a) 1, (b) 2, (c) 4, (d) 8, (e) 16 e (f) 32 threads



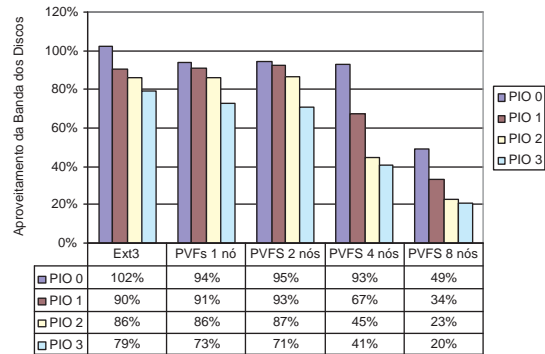
(a)



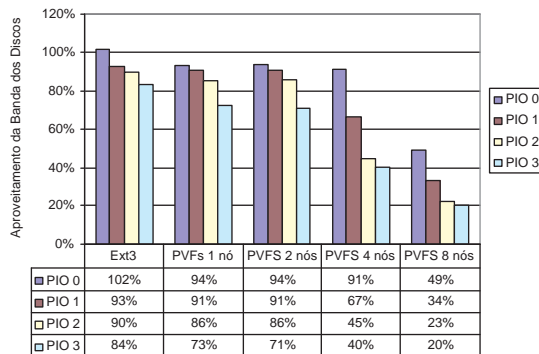
(b)



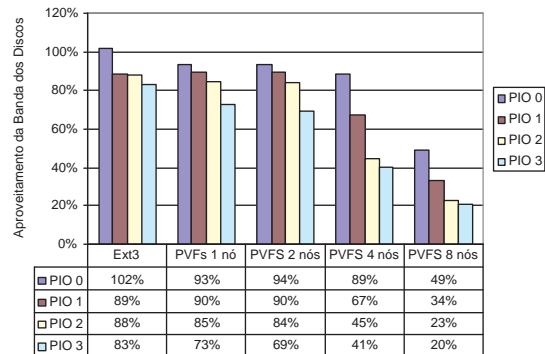
(c)



(d)

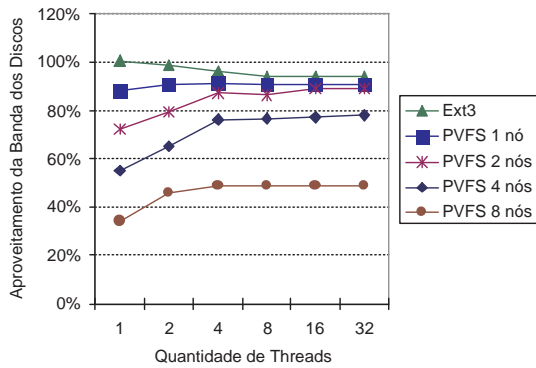


(e)

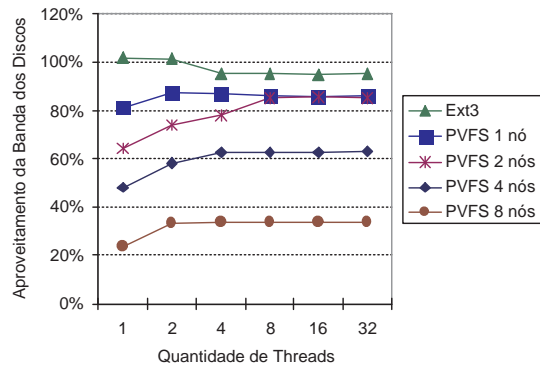


(f)

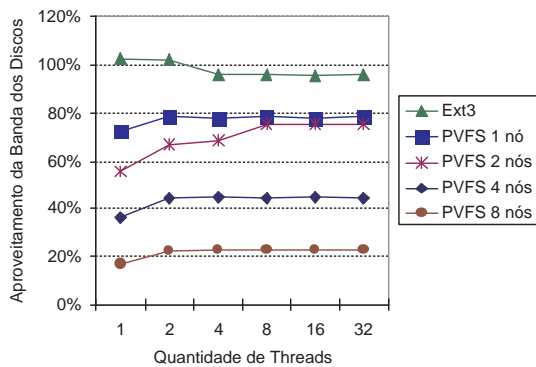
Figura 5.26: Aproveitamento que o Ext3 e o PVFS2 tiveram, relativo à velocidade média agregada disponibilizada pelos discos dos servidores, para os vários modos PIO, escrevendo 1GB de dados, blocos de 64KB, usando (a) 1, (b) 2, (c) 4, (d) 8, (e) 16 e (f) 32 threads



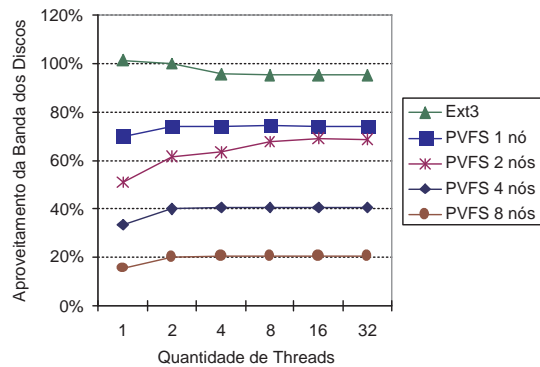
(a)



(b)

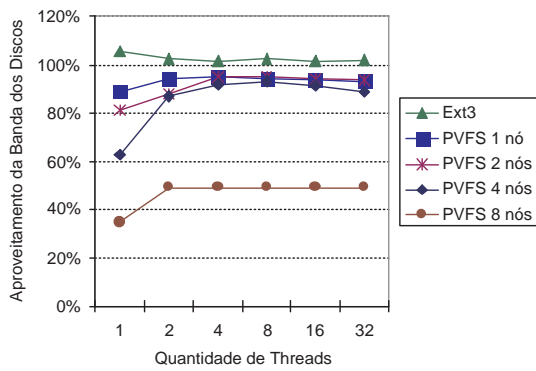


(c)

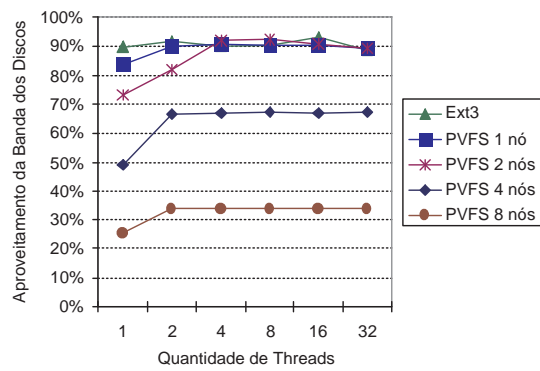


(d)

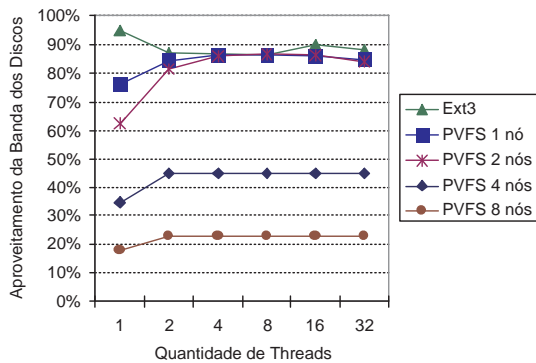
Figura 5.27: Aproveitamento dos sistemas de arquivos relativo à velocidade média agregada disponibilizada pelos discos dos servidores, para os modos PIO (a) 0, (b) 1, (c) 2 e (d) 3, lendo 1GB de dados, blocos de 64KB



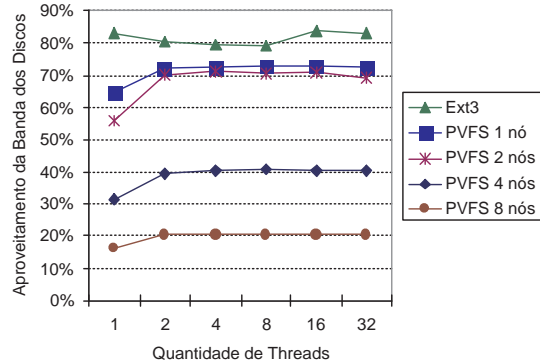
(a)



(b)



(c)



(d)

Figura 5.28: Aproveitamento dos sistemas de arquivos relativo à velocidade média agregada disponibilizada pelos discos dos servidores, para os modos PIO (a) 0, (b) 1, (c) 2 e (d) 3, escrevendo 1GB de dados, blocos de 64KB



## Capítulo 6

# Conclusão

Os sistemas de arquivos paralelos (SAPs) têm como principal propósito o alto desempenho, deixando de lado algumas transparências essenciais para usuários comuns, mas dispensáveis para aqueles que necessitam de uma ferramenta veloz, com a finalidade de reduzir o gargalo causado pela evolução mais lenta dos discos em comparação com processador, memória e rede. Para tanto, os SAPs utilizam-se justamente desses recursos, cuja evolução foi mais acentuada, para enfrentar a lentidão no acesso aos arquivos.

Os sistemas de arquivos distribuídos (SADs) são considerados um super-conjunto dos SAPs, pois provêm as transparências que os usuários comuns necessitam para trabalhar, de forma a não deixar-se notar que estão lidando com arquivos remotos. Um exemplo dessas transparências é o nome do arquivo, que não deve indicar a sua localização física, mas somente a lógica.

Além disso, alguns SADs possuem qualidades específicas, como por exemplo serviço de tolerância a falhas, que evita que dados sejam perdidos ou que pelo menos permita recuperá-los de alguma forma caso alguma falha no sistema de armazenamento aconteça. Outro serviço interessante é a alta disponibilidade, que permite que o usuário não perceba uma falha e continue acessando seus arquivos como se nada tivesse acontecido. Embora seja muito similar à tolerância a falhas, pode não permitir que se recupere os dados em caso de falha muito grave. Alguns sistemas, como o CODA, levam essa transparência ao extremo ao permitir que os usuários possam se desconectar da rede e, ainda assim, continuar acessando seus arquivos através do cache local de sua máquina.

Embora um SAD possa ser simples e confiável para o usuário, o mesmo não se pode dizer de um SAP. Como seu objetivo principal é o alto desempenho, facilidade no uso deixa de ser considerada uma premissa. Um exemplo é o GFS, ou Google File System, que por ser projetado para atender a casos específicos, deixa alguns problemas importantes para as aplicações clientes resolverem. O mais grave deles é a escrita de dados, que embora seja replicada entre os servidores, em caso de falha apenas avisa o cliente que algum dos servidores não conseguiu realizar a escrita com sucesso, não desfazendo a operação nos demais, tudo em favor do desempenho. Os clientes que forem ler esses dados devem levar essa característica em consideração, ficando a cargo deles o tratamento dessa falha.

Porém existem SAPs projetados para serem fáceis de usar, ao mesmo tempo em que proporcionam alto desempenho. O PVFS foi criado com esse conceito, que por seguir os princípios do VFS

(Virtual File System) do Linux, acaba sendo transparente para os usuários desse sistema operacional. Para proporcionar alto desempenho, o PVFS distribui seus dados entre os servidores, que são acessados diretamente pelos clientes, após consultarem um servidor de meta-dados, que informa aos clientes onde os dados estão armazenados.

Embora o PVFS por si só não implemente alguns serviços interessantes em SADS, como replicação de dados ou tolerância a falhas, alguns pesquisadores realizaram algumas modificações em seu código-fonte (que é aberto) a fim de demonstrarem suas teorias e pesquisa ao mesmo tempo em que mostram o quanto flexível esse SAP é. O NFSP tem por objetivo permitir que qualquer cliente NFS possa acessar esse PVFS modificado, tornando-o acessível não somente pelo Linux, mas também através de outros sistemas operacionais como Windows, Solaris, HP-UX, entre alguns exemplos. Já o CEFT-PVFS mostra que é possível ganhar alguma tolerância a falhas ao fornecer duplicação de dados, no intuito de melhorar o acesso concorrente aos dados.

Devido ao trabalho bem sucedido, altamente disseminado, com ótima qualidade e muito bem visto pela comunidade científica, a equipe desenvolvedora do PVFS resolveu rever a sua arquitetura para permitir que ele fosse mais facilmente e amplamente utilizado, tanto como sistema de arquivos, como também base para o desenvolvimento de novos SAPs. Dessa forma, surgiu o PVFS2, que embora em nossos testes não tenha apresentado melhora significativa com relação ao seu antecessor, se mostrou tão confiável e rápido quanto, embora não tenha sido possível observar as novidades prometidas para essa versão, pois ainda não haviam sido desenvolvidas. Como o PVFS e o PVFS2 não apresentaram grandes diferenças, resolvemos adotar o PVFS2 para a realização de nossos testes de desempenho.

## 6.1 Principais Contribuições

Nossa proposta foi mostrar como é possível reduzir o gargalo provocado pelo acesso ao disco ao utilizar-se de recursos que evoluem mais rapidamente, como memória, processador e rede. Para atingí-la, era necessário comparar o PVFS2 com o Ext3 em um ambiente onde os discos de cada máquina fossem mais lentos que a rede. Como não foi possível obter acesso a uma rede com essa característica, a solução adotada foi usar uma rede conectada a 100MBits/s, com discos de última geração (cerca de 3 vezes mais rápidos que a rede), porém com a velocidade original reduzida para serem mais lentos que ela, através de configurações da interface de comunicação dos mesmos.

Durante os testes, constatamos fatos interessantes, como o de que o tamanho do bloco de dados, tanto para leitura como para escrita, não influenciou nossos resultados. Percebemos também que a quantidade de dados lidos influenciou o desempenho do PVFS2, degradando quando se tem poucos dados e muita concorrência, mas melhorando nos outros casos. Já o Ext3 não teve essa influência. Notamos também que conforme o nível de concorrência (a quantidade de threads usadas para acessar os sistemas de arquivos) aumenta, para muitos dados lidos o PVFS2 melhora seu desempenho, enquanto que o Ext3 degrada levemente.

Foi confirmado que ao aumentarmos a quantidade de servidores, o desempenho do PVFS2 melhora consideravelmente, mostrando que o uso da velocidade agregada dos discos é aproveitada, mesmo que não na sua totalidade, tornando-o mais rápido que o Ext3. Até mesmo ao se utilizar apenas uma thread para o acesso aos arquivos, o PVFS2 com mais de um servidor foi mais eficiente

que o Ext3, mostrando seu bom desempenho também com aplicações mono-tarefas. Esses resultados se mantiveram também nos testes de escrita de dados.

Porém, notamos que ao aumentar a velocidade dos discos, o desempenho do PVFS2 não aumentava linearmente, enquanto que no Ext3 isso acontecia. Isso ocorreu devido ao aproveitamento da velocidade agregada dos discos pelo PVFS2 estar limitado à velocidade da rede. Ou seja, conforme a velocidade individual dos discos aumentava, menor era o aproveitamento, principalmente quando a quantidade de servidores era maior, pois gerava uma maior banda agregada com discos, enquanto que a banda da rede se mantinha limitada à única placa de rede do cliente.

Notamos que o PVFS2 atingiu 92% de utilização da banda disponível pela rede nas suas configurações mais agressivas, de onde conclui-se que para o PVFS2 ser mais eficiente que o Ext3, a velocidade média do disco precisa ser 8% mais lenta que a velocidade máxima da rede. Além disso, a partir da análise do aproveitamento que o PVFS2 fez da velocidade agregada dos discos, foi possível construir uma tabela com o percentual de aproveitamento de cada configuração do PVFS2 utilizada, permitindo assim uma aproximação, através de uma simples regra de três, da velocidade esperada do PVFS2 para outras configurações de velocidades de rede e disco.

Por fim, as principais colaborações do nosso trabalho se referem: à atualização do trabalho [Kon], onde adicionamos informações atualizadas sobre a evolução de alguns dos sistemas de arquivos citados; à inclusão de sistemas de arquivos paralelos neste estudo, mostrando como os SADS evoluíram e se especializaram, criando uma nova área para a pesquisa de alto desempenho; e à comparação detalhada entre um SAP (no caso, PVFS2) e o acesso ao disco local (Ext3), mostrando que é possível minimizar a falta de desempenho dos discos a partir do alto desempenho das redes de dados atuais.

## 6.2 Trabalhos Futuros

Como não foi possível utilizar uma rede realmente mais rápida que os discos, realizamos apenas uma estimativa da velocidade do PVFS2 utilizando-se dos mesmos discos usados nos testes, em sua plena capacidade, mas com uma rede Ethernet 1000Base-T (Gigabit). Deixamos como trabalho futuro a confirmação de nossa estimativa em uma rede realmente mais rápida, onde poderemos concluir que o uso de um sistema de arquivos paralelo pode realmente reduzir o gargalo no acesso ao sistema de arquivos.



## Apêndice A

# PSplit: Parallel Split

O PSplit foi desenvolvido especificamente para a tomada de tempos de leitura e escrita paralela de nossos testes, utilizando-se de apenas um processo com várias threads, onde cada uma lê um trecho distinto, contínuo e de tamanho aproximado de um arquivo.

Embora seja possível usar o PSplit como uma ferramenta para a quebra de um arquivo em várias partes, seu principal foco é a simulação do acesso concorrente ao sistema de arquivos. Pode-se optar por somente leitura paralela de um arquivo, leitura e escrita das partes de um arquivo ou somente escrita das partes, que serão constituídas de dados aleatórios, gerados no momento da execução.

### A.1 Parâmetros da Linha de Comando

O PSplit é extremamente flexível quanto às diversas necessidades que o usuário possa apresentar. É possível especificar o tamanho de cada parte a ser gerada ou a quantidade de partes que se quer gerar, o arquivo de entrada, o prefixo do nome do arquivo de saída, o tamanho do bloco, o formato das informações sobre tempo de processamento, entre outros. Cada uma dessas opções é descrita a seguir:

- h, --help Exibe os parâmetros da linha de comando descritos a seguir.
- c, --csv Exibe os resultados de desempenho, como quantidade de threads, tamanho do arquivo lido ou gravado, velocidade de leitura ou escrita, em formato CSV, ou seja, separados por vírgula, para fácil importação em ferramentas de geração de gráficos. Essa opção está desativada por padrão.
- i, --input <nome> Especifica o nome do arquivo a ser usado na leitura. Se não for especificado, o PSplit assumirá que os dados devem ser gerados aleatoriamente e que o tamanho de cada parte a ser gerada deve ser informada através da opção **size**, assim como a quantidade de partes através da opção **pieces**.
- o, --output <nome> Especifica o prefixo do nome do arquivo a ser usado para escrita das partes. Será acrescentado o sufixo “.0000” para a primeira parte, “.0001” para a segunda e assim por

diante. Caso não seja especificado esse parâmetro, o PSplit assumirá que não se quer saída, apenas um teste de desempenho de leitura paralela, não gerando, assim, qualquer tipo de escrita.

- n, --pieces <partes> Especifica a quantidade de partes a serem geradas. Esse valor também indica a quantidade de threads ativas a serem usadas durante o processamento, ou seja, o grau de paralelismo. Se não for especificada, deve-se especificar o nome do arquivo de entrada e o tamanho de cada parte a ser gerada (opção **size**).
- s, --size <tamanho> Especifica o tamanho de cada parte a ser gerada. Caso não seja especificada, o nome do arquivo de entrada deve ser especificado, assim como a quantidade de partes a serem geradas (opção **pieces**). Esse valor pode ser informado utilizando-se de multiplicadores como K, M ou G. Por exemplo, pode-se usar 64K para indicar o valor 65536.
- f, --flush Indica que a cada operação de escrita dos dados, deve-se chamar a função `fflush`, forçando os dados a serem escritos no sistema de armazenamento, ao invés de deixá-los no cache do sistema para uma escrita posterior.
- b, --block <tamanho> Especifica o tamanho do bloco de leitura ou escrita, a ser usado por cada uma das threads internas do PSplit. O tamanho do bloco influencia na quantidade de iterações necessárias para se realizar a leitura completa dos dados (ou escrita). Essa opção também aceita os multiplicadores citados na opção **size**.

## A.2 O Processamento Paralelo

O PSplit realiza a leitura e escrita dos dados de forma paralela, a partir da criação de threads. Tais threads são criadas e manipuladas usando-se as funções POSIX, disponíveis no ambiente Linux.

A quantidade de threads a serem criadas pode ser especificada explicitamente ou calculada internamente, da seguinte forma:

- Caso seja especificado o parâmetro **pieces**, indicando para quebrar o arquivo em  $n > 0$  partes, serão criadas  $n - 1$  threads, sendo que a última parte será criada usando-se a thread principal do processo. Cada thread lerá  $\frac{S}{n}$ , onde  $S$  é o tamanho do arquivo. Caso essa divisão não seja exata, o tamanho que cada thread lerá será aumentado em 1 byte, exceto para a última thread, que lerá menos dados. Isso garante que não ultrapassará a quantidade de partes a serem geradas.
- Caso seja especificado o parâmetro **size**, indicando que cada parte do arquivo deve ter tamanho fixado em  $s$  bytes, será calculada a quantidade de pedaços a serem gerados, baseado no tamanho do arquivo,  $S$ . Serão criadas até  $\frac{S}{s}$  threads, e o resto dessa divisão será o tamanho do último pedaço a ser gerado, que será processado pela thread principal, caso seja diferente de zero.
- Caso não seja especificado um arquivo de entrada, deve-se especificar o tamanho de cada parte a ser gerada e a quantidade de partes a serem geradas, isto é, os parâmetros **pieces**

e **size**. Serão criadas uma thread para cada parte, onde cada uma gerará um arquivo do tamanho especificado.

Cada thread irá abrir o arquivo e se posicionar numa data posição do arquivo antes de começar a leitura, através da função `fseek`. Após isso, será realizada repetidamente a leitura dos dados usando-se o tamanho de bloco de leitura especificado na linha de comando. Caso seja especificado, os dados lidos serão gravados em outro arquivo, utilizando-se do mesmo tamanho de bloco, dentro da mesma iteração.

Ao terminar de ler sua região do arquivo, os arquivos lidos e gravados serão fechados e a thread encerrará suas atividades. A thread principal aguardará até que todas as threads filhas tenham terminado suas tarefas antes de continuar e calcular o tempo gasto pelo processo, além de informar sua velocidade média de leitura ou escrita de dados.

### A.3 Interpretando os Resultados Apresentados

Ao final do processamento, serão apresentadas algumas informações passadas na linha de comando e outras obtidas ou calculadas durante o processamento, que são:

- Tamanho de cada parte gerada (em bytes);
- Tamanho da última parte gerada em bytes (importante caso a divisão dos dados entre as partes não seja exata);
- Total de partes geradas;
- Tamanho do bloco de leitura ou escrita usado (em bytes);
- Total de dados lidos ou escritos (mas não a soma deles) em bytes e em MBytes;
- Tempo total gasto, em segundos, com precisão até milissegundos;
- Velocidade de leitura ou escrita (mas não ambas) em bytes por segundo e MBytes por segundo.

Caso especifique-se a opção `csv`, os dados apresentados serão serapados por vírgula, na seguinte ordem:

1. Quantidade de partes ou threads criadas;
2. Tamanho do bloco de leitura ou escrita, em bytes;
3. Quantidade de dados lidos ou escritos (mas não a soma), em bytes;
4. Quantidade de dados lidos ou escritos (mas não a soma), em MBytes;
5. Tempo total gasto durante a execução, em segundos, com precisão até microssegundos;
6. Velocidade de leitura ou escrita, em bytes por segundo, com precisão até a sexta casa decimal;
7. Velocidade de leitura ou escrita, em MBytes por segundo, com precisão até a sexta casa decimal.

## A.4 Exemplos de Uso

Para deixar claro como é o funcionamento do PSplit, a seguir encontra-se um exemplo:

```
psplit -i teste.1G -n 16 -b 64K -c
```

Nesse exemplo, será processado o arquivo teste.1G, utilizando-se de 16 threads, bloco de 64KB e resultado exibido em formato CSV. Não será gerado nenhum arquivo de saída, ou seja, é apenas um teste de desempenho de leitura.

```
psplit -i teste.1G -n 16 -b 64K -o saida.1G -c
```

Já no exemplo acima, serão gerados 16 arquivos, de nome saida.1G.0000 até saida.1G.0015, cada um sendo uma parte subsequente do arquivo teste.1G.

```
psplit -n 16 -s -b 64K -o saida.1G -c
```

E no exemplo acima, serão gerados os mesmos arquivos citados no exemplo anterior, porém com dados aleatórios, cada um com tamanho de 64MB.

## Apêndice B

# Introdução aos Discos Magnéticos

Os discos magnéticos [Koz01a] possuem hoje o melhor custo/benefício para leitura e escrita de dados dentre outras formas de armazenamento para um sistema de arquivos. Também são o meio mais comum usado desde desktops até servidores de médio porte.

Também chamados de discos rígidos, são compostos de cabeças de leitura e discos de armazenamento magnético. Tais discos são mapeados em **cilindros**, **trilhas** e **setores** (veja figura ao lado). Uma trilha é uma circunferência de certo raio em uma dada face do prato. Um prato é um disco magnético que faz parte do disco rígido como um todo. Ele pode conter vários pratos. Um cilindro é composto por todas as trilhas de mesmo raio do disco rígido. Por exemplo, se tiver 4 pratos, então cada cilindro conterá 8 trilhas. Um setor é um arco de uma trilha. Todos os setores em um mesmo disco rígido têm o mesmo tamanho.

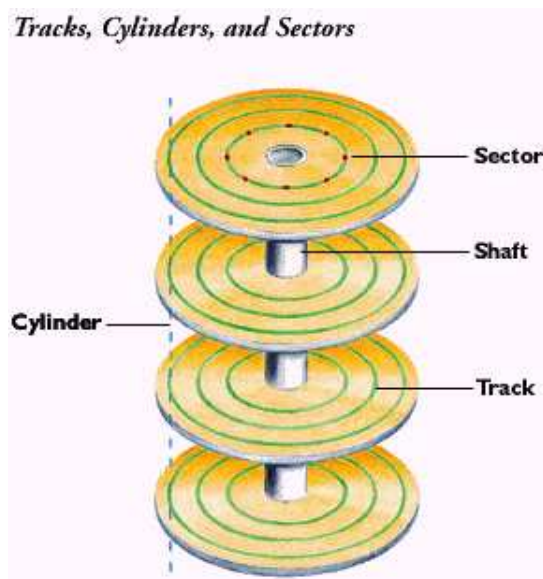


Figura B.1: Organização lógica de um disco rígido

### B.1 Fatores Para Análise de Desempenho

Para ler ou escrever algum dado, o controlador do dispositivo pede para o mesmo mover o braço de leitura para um determinado cilindro, pedindo para ativar a cabeça de leitura de uma determinada face, de um determinado prato (ou seja, para um determinada trilha), esperando então que o disco gire até que o setor fique sob a cabeça, que iniciará a operação de transferência de dados. Para se realizar todas essas tarefas, leva-se um determinado tempo, dado que grande parte delas são mecânicas. Abaixo seguem os tempos de maior relevância:

- **Tempo de busca:** Corresponde ao tempo necessário para que a cabeça de leitura mova-se da

trilha em que ela está naquele instante até a trilha que contém o setor a ser lido ou gravado. Esse tempo, em média, varia de 8 a 10 ms<sup>1</sup>. Além do tempo médio, existem duas outras medidas: tempo entre trilhas adjacentes (*track-to-track*), que costuma ser de 1 ms, e tempo entre trilhas opostas do disco (*full stroke*), correspondendo ao tempo de se passar da trilha de menor raio à de maior raio do disco. Esse tempo varia entre 15 e 20 ms. Tais valores, assim como os próximos a serem discutidos, são referentes à tecnologia disponível em 2001, que não tem melhorado muito nos últimos anos;

- **Tempo de assentamento:** Quando a cabeça de leitura chegar à trilha desejada, levará um certo tempo até que ela se estabilize e possa ler ou gravar os dados com segurança. Em geral isso demora cerca de 0,1 ms. Costuma-se agregar esse valor no tempo de busca;
- **Tempo de reação:** É o tempo que o drive demora para dar início à busca pelo setor requisitado. Em geral isso demora 0,5 ms. Para se entender melhor, uma analogia é quando alguém está dirigindo e o semáforo muda para vermelho. Seria o tempo que o motorista demora para perceber que deve pisar no freio;
- **Latência de rotação:** Após chegar à trilha procurada, a cabeça de leitura deve esperar que o setor que se quer ler ou escrever chegue à ela. O tempo gasto nessa espera está intimamente relacionado com a velocidade de rotação do disco e pode ser, em geral, calculado pela fórmula (30000 / Velocidade de rotação em RPM), com resultado em ms. Esse tempo é importante para a melhora no desempenho de acessos randômicos no disco, pois em acessos contínuos a cabeça de leitura sempre estará na posição ideal para realizar suas tarefas.

O **tempo de acesso** é calculado como sendo a soma desses quatro tempos acima definidos. Como o tempo de assentamento e o de reação são muito baixos, somente o tempo de busca e o de latência são preocupantes para o desempenho. Nos últimos anos, o tempo de busca não tem mudado muito e o tempo de latência depende muito da velocidade de rotação dos discos. No ano de 2000, os discos mais rápidos giravam a 15000 RPM (SCSI), enquanto que aqueles que possuem melhor custo/benefício estavam na faixa dos 5400 a 7200 RPM.

Além desses tempos, existem outros não muito comentados, mas que influenciam o desempenho do tráfego interno dos dados nos discos:

- **Tempo de transferência de dados internos:** É o tempo gasto para ler ou gravar os dados no disco magnético. Em 2000 esse valor estava em 500 Mbit/s. Vale lembrar que ele é referente apenas a transferência de dados internamente no disco, sem chegar a máquina em si;
- **Tempo de troca de cabeça:** É o tempo gasto entre a desativação de uma cabeça de leitura de uma superfície de um prato e a ativação de outra. É um processo puramente eletrônico, sem envolver partes mecânicas nessa operação. Costuma demorar entre 1 e 2 ms (desde a parada da leitura/escrita de uma cabeça até o início da leitura/escrita da outra);

---

<sup>1</sup>1ms =  $\frac{1}{1000}$  segundos

- **Tempo de troca de cilindro:** É o tempo gasto entre a parada da leitura/escrita em um cilindro para o início da leitura/escrita em outro. Ele demora cerca de 2 a 3 ms. Apenas definindo novamente, mas de outra forma, um cilindro é composto por todas as trilhas que podem ser acessadas a partir da mesma posição das cabeças de leitura entre os pratos.

## B.2 Interfaces Entre o Disco e o Computador

Após os dados serem lidos pelo disco, é necessário enviá-los para a memória da máquina, a fim de permitir que sejam usados pelo processador. Para isso, é necessário existir uma interface ligando o disco rígido ao barramento de dados da máquina. Existem fatores de desempenho relativos a essa interface, que dependem muito de como foi desenvolvida, envolvendo tanto hardware como software.

Abaixo seguem algumas dessas interfaces, suas velocidades de transmissão de dados (pico) e quando surgiram:

- **SCSI-1:** Aprovado pela ANSI em 1986, permite velocidades de até 5 MB/s para transferência de dados. Isso foi na mesma época do Intel 80386, de 33 MHz;
- **SCSI-2:** Enviado para aprovação em 1985 e só aprovado pela ANSI em 1994, permite velocidades de até 10 MB/s;
- **EIDE:** Termo concebido pela Western Digital como sendo a extensão para o IDE (*Integrated Drive Electronics*). Apareceu em 1994, na época dos processadores Pentium de 100MHz [Koz01b, Exe02]. Infelizmente foi um padrão proprietário e que possuía muitas limitações, como a barreira dos 504 MB de limite de espaço disponível e a velocidade de transferência de 11 MB/s;
- **ATA (IDE):** *AT Attachment*, ou mais popularmente *IDE (Integrated Drive Electronics)*, esse padrão foi aprovado pela ANSI em 1994 (sendo que foi enviado para aprovação em 1990) e permite velocidades de transferência entre 3 e 8 MB/s;
- **ATA-2:** Aprovado pela ANSI em 1996, época em que a Intel anunciava os Pentium 200MHz, esse padrão permite velocidades de até 16 MB/s;
- **SCSI-3 (SPI):** Lançado em 1996, essa especificação permite velocidades de até 20 MB/s;
- **ATA-3:** Anunciado em 1997, com velocidades de até 33 MB/s (embora esse fosse o valor de pico, poucos discos chegavam a essa velocidade). Já havia processadores Pentium II 400 MHz a venda no mercado;
- **ATA-4:** Publicado em 1998, esse padrão realmente atingia velocidades de 33,3 MB/s. É chamado também de Ultra-DMA/33;
- **ATA-5:** Publicado em 1999, permite velocidades de até 66 MB/s. É chamado também de Ultra-DMA/66. Somente no início de 2000 apareceram discos com esse padrão. Nesse mesmo ano a AMD lançava o Athlon 1 GHz;

- **SCSI-3 (SPI-2):** Lançado em 1999, essa especificação permite velocidades de até 80 MB/s para canais 16-bit;
- **SCSI-3 (SPI-3):** Lançado no início de 2001, essa especificação permite velocidades de até 160 MB/s;
- **ATA-6:** Prometendo atingir 100 MB/s (Ultra-DMA/100), esse padrão chegou em 2001, quando os processadores já chegavam a 2 GHz;
- **ATA-7:** Também conhecido por Ultra-DMA/133, promete atingir 133 MB/s. Esse padrão chegou em 2002;
- **SCSI-3 (SPI-4):** Disponível em 2002, essa especificação permite velocidades de até 320 MB/s.
- **SCSI-3 (SPI-5):** A especificação ficou pronta no início de 2003, segundo [WSC05], permitindo velocidades de até 640 MB/s.
- **SATA150:** Primeira geração da interface Serial ATA [WSA05], apareceu em 2003. A velocidade atingida por essa interface é pouco maior que a da ATA-7.
- **SATA300:** Também conhecido como Serial ATA 300, essa interface permite velocidades de até 300MB/s [WSA05]. Surgiu em 2004.
- **SATA600:** Ainda em desenvolvimento, espera-se velocidades de até 600MB/s. A previsão de lançamento para 2007.

Sobre as tecnologias acima discutidas, fatores como o tamanho do bloco lido pelo disco (isto é, mesmo que o sistema operacional não requisite, o bloco inteiro vai para o cache do disco, agilizando requisições subseqüentes), os modos DMA e Ultra-DMA (usado para acesso direto à memória pelo disco, evitando sobrecarga no processador), os modos PIO, etc, também influenciam o desempenho de um disco.

O modo PIO [WPM05] (*Programmed Input/Output*) foi o método original para a transferência de dados entre o disco e o processador. Existem 5 modos que definem velocidades máximas de transmissão de dados entre os dispositivos. Caso não seja explicitamente especificado o modo PIO através de configurações na interface IDE, cabe ao processador determinar a velocidade ótima para a transferência dos dados do disco.

Devido ao fato de haver uma grande sobrecarga no processador para configurar cada uma das transações para transferência de dados, a interface DMA e UDMA foram criadas para melhorar o desempenho nessa tarefa.

### B.3 Latência

A latência do disco influencia fortemente o desempenho do mesmo, por representar um grande percentual na composição do tempo de acesso (juntamente com o tempo de busca). Ela é inversamente proporcional a velocidade de rotação do disco e, com base na evolução dessa última,

conseguimos definir a evolução da redução do tempo da latência do disco, que é encontrada a seguir. Vale lembrar que esses são os valores médios da época e não os melhores, dado que elas variam muito mais por fabricante que por tecnologia.

- **8,3 ms:** Até o início da década de 90 os discos rodavam a 3600 RPM.
- **6,7 ms:** No início da década de 90 apareceram discos que rodavam a 4500 RPM.
- **5,6 ms:** Pouco tempo depois (aproximadamente em 1992) apareceram os discos que rodavam a 5400 RPM. Estes se tornaram padrão por muito tempo.
- **4,2 ms:** Por volta de 1997, apareceram discos de 7200 RPM.
- **2 ms:** No ano de 2000 apareceram discos de 15000 RPM.
- **1 ms:** Discos de 30000 RPM ainda estão em desenvolvimento, mas acredita-se que não valha a pena o custo para se ter um equipamento desses, que necessita de altíssima precisão, para ganhar apenas 1 ms no tempo de acesso.

A partir das taxas médias de transferência dos sistemas de armazenamento, juntamente com a evolução do tempo de acesso no decorrer do mesmo período, é possível construir um gráfico contendo a curva de crescimento de desempenho dos discos magnéticos (veja figura [B.2](#)). Comparando-se essa curva com a do crescimento de desempenho dos processadores, memória e rede, percebe-se que a evolução do disco não acompanha as demais tecnologias na mesma taxa de crescimento.

O que por um lado é desanimador, por outro é uma dica para tentar se utilizar das demais tecnologias como ferramentas para driblar a ineficiência do disco, criando, assim, novas áreas de estudo, como a de sistemas de arquivos paralelos.

## B.4 Evolução dos Discos vs. Processadores e Rede

As informações contidas neste capítulo foram obtidas a partir de [Koz01a, WSC05] (dados sobre a evolução dos discos), [Tec99, Wik03, MNE03, Myr03] (informações importantes sobre a evolução de algumas tecnologias de transmissão via rede) e [Koz01b, Exe02, Pos02, Gav04] (evolução dos processadores).

A latência informada nessa tabela corresponde ao valor média da época, já que tal valor não tem relação direta com a tecnologia de interface usada. Além disso, cada modelo de disco possui um valor específico.

Ano	Interface	(pico)	(latência)	Rede	(pico/canal)	Processador	freq.	(pico)
1986	SCSI-1	(5 MB/s)	8,3 ms	Ethernet 10Base2	(10 Mbit/s)	80386 DX	16 MHz	(63,6 MB/s)
1994	SCSI-2 (Fast SCSI)	(10 MB/s)	5,8 ms	Ethernet 10BaseF	(10 Mbit/s)	80486 DX4	100 MHz	(127,2 MB/s)
	EIDE ATA	(11 MB/s) (3-8 MB/s)				Pentium	100 MHz	(508,6 MB/s)
1996	SCSI-3 (SPT) (Ultra SCSI)	(20 MB/s)		Fast Ethernet	(100 Mbit/s)	Pentium	200 MHz	(508,6 MB/s)
1997	ATA-2	(16 MB/s)	4,2 ms	Myrinet	(1,6 Gbit/s)	6x86	200 MHz	(572,2 MB/s)
	ATA-3	(33 MB/s)				Pentium II	300 MHz	(508,6 MB/s)
1998	ATA-4 (Ultra ATA/33)	(33 MB/s)		Gigabit Ethernet	(1 Gbit/s)	K6-II	400 MHz	(1,06 GB/s)
1999	SCSI-3 (SPL-2) (Wide Ultra2)	(80 MB/s)		Myrinet (TCP/IP)	(1,147 Gbit/s)	Athlon	750 MHz	(1,6 GB/s)
2000	ATA-5 (Ultra ATA/66)	(66 MB/s)	2,0 ms			Athlon	1 GHz	(1,6 GB/s)
2001	SCSI-3 (SPL-3) (Ultra160)	(160 MB/s)		Myrinet-Fiber	(2 Gbit/s)	Pentium IV	2 GHz	(3,2 GB/s)
	ATA-6 (Ultra ATA/100)	(100 MB/s)						
2002	SCSI-3 (SPL-4) (Ultra320)	(320 MB/s)		10 Gigabit Ethernet	(10 Gbit/s)	Pentium IV	2,8 GHz	(4,3 GB/s)
	ATA-7 (Ultra ATA/133)	(133 MB/s)						
2003	SCSI-3 (SPL-5) (Ultra640)	(640 MB/s)				Pentium IV	3,2 GHz	(4,3 GB/s)
	Serial ATA 150	(150 MB/s)						
2004	Serial ATA 300	(300 MB/s)				Pentium IV	3,8 GHz	(5,3 GB/s)

Tabela B.1: Comparativo da evolução do disco rígido e do processamento de dados

A figura B.2 representa a tabela B.1 de forma normalizada e partindo do mesmo patamar, ou seja, supondo que em 1986 todas as tecnologias apresentadas sobre disco rígido, rede e processador estavam no mesmo nível de evolução. Dessa forma, as linhas desse gráfico mostram o quanto tais tecnologias evoluíram em relação às outras numa escala logarítmica conforme o tempo, levando-se em conta somente os dados apresentados anteriormente.

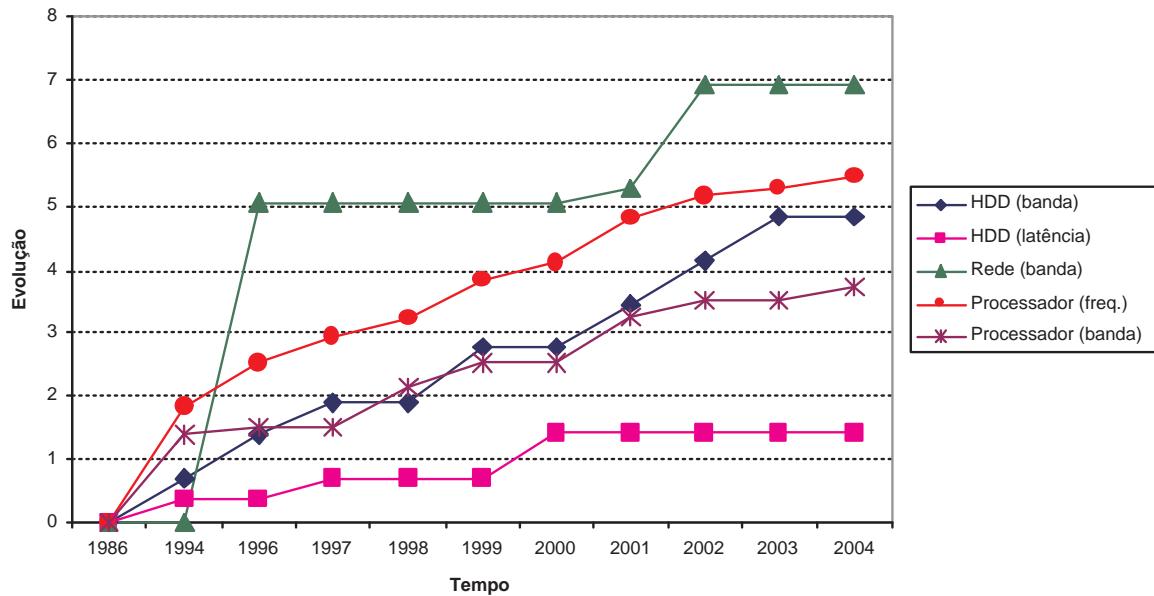


Figura B.2: Evolução tecnológica normalizada



# Referências Bibliográficas

- [Aas05] Josh Aas. Understanding the Linux 2.6.8.1 CPU Scheduler. Technical report, Silicon Graphics, Fevereiro 2005. Acessado em junho de 2005: [http://josh.trancesoftware.com/linux/linux\\_cpu\\_scheduler.pdf](http://josh.trancesoftware.com/linux/linux_cpu_scheduler.pdf).
- [AEK96] Ahmed Amer and Amr El-Kadi. Beating bottlenecks in the design of distributed file systems. *login.*, 21(6):14–23, Dezembro 1996.
- [CIRT00] Philip H. Carns, Walter B. Ligon III, Robert B. Ross, and Rajeev Thakur. PVFS: A parallel virtual file system for linux clusters. In *Proceedings of the 4th Annual Linux Showcase and Conference*, pages 317–327, Outubro 2000. (Best Paper Award), Acessado em dezembro de 2005: <http://www.parl.clemson.edu/pvfs/el2000/extreme2000.ps>.
- [CRIW05] Philip H. Carns, Robert B. Ross, Walter B. Ligon III, and Pete Wyckoff. BMI: A network abstraction layer for parallel i/o. In *19th International Parallel and Distributed Processing Symposium (IPDPS 2005)*. IEEE Computer Society, 2005. Acessado em junho de 2005: <http://www.osc.edu/~pw/papers/carns-bmi-ipdps05.pdf>.
- [DSE88] Peter Dibble, Michael Scott, and Carla Ellis. Bridge: A high-performance file system for parallel processors. In *Proceedings of the Eighth International Conference on Distributed Computer Systems*, pages 154–161, 1988. Acessado em dezembro de 2005: [http://www.cs.rochester.edu/u/scott/papers/1988\\_ICDCS\\_Bridge.pdf](http://www.cs.rochester.edu/u/scott/papers/1988_ICDCS_Bridge.pdf).
- [Eis99] M. Eisler. *NFS Version 2 and Version 3 Security Issues and the NFS Protocol's Use of RPCSEC\_GSS and Kerberos V5*, Junho 1999. Acessado em dezembro de 2005: <http://rfc-ref.org/RFC-TEXTS/2623/chapter2.html>.
- [Exe02] ExeCRaBLE. *Prozessor History*, 2002. Acessado em novembro de 2003: <http://www.execrable.de/hardware/history.html>.
- [Gal00] Doreen L. Galli. *Distributed Operating Systems*. Prentice Hall, 2000.
- [Gav04] Ilya Gavrichenkov. Intel pentium 4 extreme edition 3.46ghz and i925xe express: 1066mhz bus in action! Technical report, Xbit Laboratories, 2004. Acessado em junho de 2005: <http://www.xbitlabs.com/articles/cpu/display/p4xe-346-5.html>.

- [GGL03] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 29–43, New York, NY, USA, 2003. ACM Press. Acessado em fevereiro de 2004: <http://www.cs.rochester.edu/sosp2003/papers/p125-ghemawat.pdf>.
- [GST02] Elizabeth Garbett, Andrew Scheferman, and Albert Tse. Virtual Disk - It's Not Just For Mainframes Anymore. Technical report, StorageTek, Junho 2002. <http://www.storagetek.com/>.
- [Had00] Ibrahim F. Haddad. Pvfs: A parallel virtual file system for linux clusters. *Linux J.*, 2000(80es):5, 2000. Acessado em dezembro de 2005: <http://www.linuxjournal.com/article/4354>.
- [HKM<sup>+</sup>88] John H. Howard, Michael L. Kazar, Sherri G. Menees, David A. Nichols, M. Satyanarayanan, Robert N. Sidebotham, and Michael J. West. Scale and performance in a distributed file system. *ACM Trans. Comput. Syst.*, 6(1):51–81, 1988. Acessado em dezembro de 2005: <http://www.cs.cmu.edu/afs/cs/project/coda/Web/docdir/s11.pdf>.
- [Ins02] Tech Insider. An interactive look at how ethernet has evolved. Technical report, Network World, 2002. Acessado em novembro de 2003: <http://www.networkworld.com/techinsider/2002/1014ethernettime.html>.
- [JFC94] M.-C. Jih, Li-Chi Feng, and Ruei-Chuan Chang. The design and implementation of the pasda parallel file system. In *Proceedings 1994 International Conference on Parallel and Distributed Systems*, pages 142–147, 1994.
- [JKY00] T. Jones, A. Koniges, and R. K. Yates. Performance of the IBM general parallel file system. In *Proceedings of the International Parallel and Distributed Processing Symposium*, pages 673–683, 2000. [http://www.llnl.gov/icc/lc/siop/papers/GPFS\\_performance.doc](http://www.llnl.gov/icc/lc/siop/papers/GPFS_performance.doc).
- [Kon] Fabio Kon. Distributed file systems past, present and future a distributed file system for 2006. Acessado em dezembro de 2005: <http://choices.cs.uiuc.edu/~f-kon/DFSPaper.ps.gz>.
- [Kon94] Fabio Kon. Sistemas de arquivos distribuídos. Master's thesis, Instituto de Matemática e Estatística da Universidade de São Paulo, 1994. Acessado em dezembro de 2005: <http://choices.cs.uiuc.edu/~f-kon/thesis/kon-master.ps.gz>.
- [Koz01a] Charles M. Kozierok. Hard disk drives. Technical report, The PC Guide, 2001. Acessado em novembro de 2003: <http://www.pcguide.com/ref/hdd/>.
- [Koz01b] Charles M. Kozierok. *The Processor*. The PC Guide, 2001. Acessado em novembro de 2003: <http://www.pcguide.com/ref/cpu/>.

- [LD01] Pierre Lombard and Yves Denneulin. NFSP: A distributed NFS server for cluster of workstations. <http://ka-tools.sourceforge.net/publications/nfsp-ipdps01.pdf>, 2001.
- [Lob03] Olivier Lobry. Evaluation des systèmes de fichiers PVFS et NFSP. Technical report, Institut d'Informatique et Mathématiques Appliquées de Grenoble, Abril 2003. Acessado em novembro de 2003: [http://www-id.imag.fr/Laboratoire/Membres/Lobry\\_Olivier/PVFS\\_NFSP/PVFS\\_NFSP.html](http://www-id.imag.fr/Laboratoire/Membres/Lobry_Olivier/PVFS_NFSP/PVFS_NFSP.html).
- [MK93] Ethan L. Miller and Randy H. Katz. RAMA: A file system for massively-parallel computers. In *Proceedings of the Twelfth IEEE Symposium on Mass Storage Systems*, pages 163–168, 1993. Acessado em dezembro de 2005: <http://ssrc.cse.ucsc.edu/~elm/Papers/mss93.pdf>.
- [MK97] Ethan L. Miller and Randy H. Katz. RAMA: An easy-to-use, high-performance parallel file system. *Parallel Computing*, 23(4–5):419–446, 1997. Acessado em dezembro de 2005: <http://ssrc.cse.ucsc.edu/~elm/Papers/pc97.pdf>.
- [ML03] Tom McNeal and Chuck Lever. *Linux NFS FAQ*, 2003. Acessado em novembro de 2003: <http://nfs.sourceforge.net>.
- [MNE03] Myricom. *News & Events Archive*, 2003. Acessado em novembro de 2003: <http://www.myricom.com/news/>.
- [Myr03] Myricom. Performance measurements. Technical report, Myricom, 2003. Acessado em novembro de 2003: <http://www.myricom.com/myrinet/performance/index.html>.
- [NLM98] IBM. *System Management Guide: Communications and Networks*, 1998. Acessado em junho de 2005: [http://publib16.boulder.ibm.com/pseries/en\\_US/aixbman/commadmn/nfs\\_netlock.htm](http://publib16.boulder.ibm.com/pseries/en_US/aixbman/commadmn/nfs_netlock.htm).
- [NWO87] M. Nelson, B. Welch, and J. Ousterhout. Caching in the Sprite network file system. In *SOSP '87: Proceedings of the eleventh ACM Symposium on Operating systems principles*, pages 3–4, New York, NY, USA, 1987. ACM Press. Acessado em dezembro de 2005: <http://www.soe.ucsc.edu/~sbrandt/290S/sprite.pdf>.
- [Ous96] John Ousterhout. A Brief Retrospective on the Sprite Network Operating System, 1996. Acessado em novembro de 2003: <http://www.cs.berkeley.edu/projects/sprite/retrospective.html>.
- [Pos02] Jef Poskanzer. Bandwidth. Technical report, ACME, 2002. Acessado em novembro de 2003: [http://www.acme.com/build\\_a\\_pc/bandwidth.html](http://www.acme.com/build_a_pc/bandwidth.html).
- [Ram02] Harish Ramachandran. Design and implementation of the system interface for PVFS2. Master's thesis, Clemson University, Dezembro 2002. Acessado em dezembro de 2005: <ftp://ftp.parl.clemson.edu/pub/techreports/2002/PARL-2002-008.ps>.

- [RLC02] Rob Ross, Walt Ligon, and Phil Carns. Parallel Virtual File System, 2002. <http://www.parl.clemson.edu/pvfs2/sc2002-whitepaper-pvfs.pdf>.
- [SCR<sup>+</sup>03] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, and D. Noveck. *Network File System (NFS) version 4 Protocol*, Abril 2003. Acessado em dezembro de 2005: <http://rfc-ref.org/RFC-TEXTS/3530/index.html>.
- [Shi96] Ken Shirriff. *The Sprite Operating System*, 1996. Acessado em novembro de 2003: <http://www.cs.berkeley.edu/projects/sprite/sprite.html>.
- [SKM<sup>+</sup>93] Mahadev Satyanarayanan, James J. Kistler, Lily B. Mummert, Maria R. Ebling, Punnet Kumar, and Qi Lu. Experience with disconnected operation in a mobile computing environment. In *Proceedings of the 1993 USENIX Symposium on Mobile and Location-Independent Computing*, pages 11 – 28, Cambridge, MA, Junho 1993. Acessado em junho de 2005: <http://www-2.cs.cmu.edu/afs/cs/project/coda/Web/docdir/mobile93.pdf>.
- [SM88] Inc. Sun Microsystems. *RPC: Remote Procedure Call Protocol Specification Version 2*, Junho 1988. Acessado em dezembro de 2005: <http://rfc-ref.org/RFC-TEXTS/1057/>.
- [SRO96] Steven R. Soltis, Thomas M. Ruwart, and Matthew T. O’Keefe. The Global File System. In *Proceedings of the Fifth NASA Goddard Conference on Mass Storage Systems*, pages 319–342, College Park, MD, 1996. IEEE Computer Society Press. Acessado em dezembro de 2005: <http://www.diku.dk/undervisning/2003e/314/papers/soltis97global.pdf>.
- [Tan92] Andrew S. Tanenbaum. *Modern Operating Systems*. Prentice Hall, 1992.
- [Tea03] PVFS2 Development Team. *Parallel Virtual File System, Version 2*. Clemson University, Setembro 2003. Acessado em março de 2005: <http://www.pvfs.org/pvfs2/pvfs2-guide.html>.
- [Tec99] TechFest. Ethernet technical summary. Technical report, TechFest, 1999. Acessado em novembro de 2003: <http://www.techfest.com/networking/lan/ethernet4.htm>.
- [Tha03] Rajeev Thakur. *ROMIO: A High-Performance, Portable MPI-IO Implementation*, Janeiro 2003. Acessado em dezembro de 2005: <http://www-unix.mcs.anl.gov/romio/>.
- [vH03] Bill von Hagen. *Exploring the Ext3 Filesystem. What is Journaling?* Linux Planet, 2003. Acessado em novembro de 2003: <http://www.linuxplanet.com/linuxplanet/reports/4136/3/>.
- [Wik03] Wikipedia. *10 Gigabit Ethernet*, 2003. Acessado em novembro de 2003: [http://www.wikipedia.org/wiki/10\\_Gigabit\\_Ethernet](http://www.wikipedia.org/wiki/10_Gigabit_Ethernet).
- [WPM05] Wikipedia. *PIO Mode*, 2005. Acessado em agosto de 2005: [http://en.wikipedia.org/wiki/Programmed\\_input/output](http://en.wikipedia.org/wiki/Programmed_input/output).

- [WSA05] Wikipedia. *Serial ATA*, 2005. Acessado em agosto de 2005: [http://en.wikipedia.org/wiki/Serial\\_ata](http://en.wikipedia.org/wiki/Serial_ata).
- [WSC05] Wikipedia. *SCSI*, 2005. Acessado em maio de 2005: <http://en.wikipedia.org/wiki/Scsi>.
- [ZJQ<sup>+</sup>03] Yifeng Zhu, Hong Jiang, Xiao Qin, Dan Feng, and David R. Swanson. Improved read performance in CEFT-PVFS: Cost Effective, Fault-Tolerant Parallel Virtual File System. In *Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2003)*, pages 730–735, Maio 2003. Acessado em junho de 2005: <http://www.cs.nmt.edu/~xqin/pubs/ccgrid03.pdf>.