UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

RAFAEL BOHRER ÁVILA

Uma Proposta de Distribuição do Servidor de Arquivos em Clusters

Tese em co-tutela apresentada como
requisito parcial para a obtenção do grau de
Doutor em Ciência da Computação

Prof. Dr. Philippe Olivier Alexandre Navaux
Orientador brasileiro

Prof. Dra. Brigitte Plateau
Orientadora francesa

Porto Alegre, junho de 2005

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Ávila, Rafael Bohrer

Uma Proposta de Distribuição do Servidor de Arquivos em Clusters / Rafael Bohrer Ávila. – Porto Alegre: PPGC da UFRGS, 2005.

156 f.: il.

Tese (doutorado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2005. Orientadores: Philippe Olivier Alexandre Navaux; Brigitte Plateau.

1. Sistemas de arquivos. 2. NFS. 3. Gerenciamento de metadados. 4. Lazy Release Consistency. 5. Computação baseada em clusters. I. Navaux, Philippe Olivier Alexandre. II. Plateau, Brigitte. III. Título.

“If I have seen farther than others,
it is because I stood on the shoulders of giants.”
— SIR ISAAC NEWTON

Este trabalho é dedicado aos meus avós,
Lêda, Emmanuel, Ivone e Nelson

AGRADECIMENTOS

Bom, se na dissertação já não foi fácil, imaginem agora 6 anos depois... :-)

Aos amigos (em ordem alfabética!): Barreto, Cadinho, Cris, Edson, Elgio, Jú, Maurício, Mozart, Nico, Pati, Piccoli, Pilla, Tati "Doidinha", Tati Pillon, pelos muitos momentos de diversão, cafés, conversas descontraídas, conversas sérias, jantares, conselhos, churrascos, sinucas, corridas (a.k.a. *coopers*), puxões de orelha, crèpes, vinhos, entre outros :-); um agradecimento especial ao "professeur de français" Nico e sua infinita paciência, pelas tantas explicações, particularmente na reta final para o fechamento do resumo;

Aos colegas e amigos de laboratório (por ordem de velheza no grupo): Clarissa, Sanger, Diego, Caciano, Ennes, Kassick, Everton, pelo convívio, pelo ambiente descontraído, pela dedicação, pelas várias horas de observação das luzinhas do switch tentando entender o que estava acontecendo, pelas milhares de (re(re))instalações de Linux, pelos anoiteceres correndo pra fechar aquele artigo... valeu!;

Aos "papudos" Pete Sampras, Thierry, Zé Gota e assemelhados (não estou autorizado a revelar suas verdadeiras identidades...) pelas várias cevas e várias horas de "reunião" no Kripton, Beer Street, Variettà, Master, etc., sempre planejando os próximos artigos;

Aos amigos conhecidos e/ou reencontrados em Grenoble: Pierre, Adrien, João, Carine, Pepê, Cris, por terem sido nossos companheiros nessa aventura incrível de viver longe de casa, e em especial ao casal Pillon por terem nos "adotado" (não só na chegada :-));

Aos funcionários do Instituto, Luis "LO" Otávio, Silvânia, Sula, Cláudia, Marcelo, Lourdes, Jorge, Cristina, obrigado por estarem sempre correndo pra quebrar "aquele" galho pra gente!;

Ao pessoal da administração da rede, Carissimi, Margareth, Leandro, Elgio, Átila, Vinícius, pela qualidade do serviço que nos oferecem e pela paciência de agüentar as nossas reclamações eventuais :-);

À Eli da portaria, Seu Astro e demais seguranças por nos permitirem viajar nas nossas maluquices um pouco menos preocupados com o mundo selvagem lá fora;

À CAPES, pela concessão das bolsas de doutorado no país e de doutorado-sanduíche para a estadia na França, e à HP Brasil, pela bolsa do final do curso;

Aos professores do Instituto, pelas diversas aulas e conhecimentos transmitidos, e em especial ao Prof. Tiarajú Diverio, com seu lado “paizão” sempre pronto a conseguir um ônibus pra levar a gurizada rumo ao alto desempenho :-);

Ao Prof. Navaux, pela orientação desde o Mestrado, pela paciência de me agüentar desde lá :-), pela experiência, pelos conselhos, por todas as oportunidades e principalmente pela amizade ao longo de todos esses anos de UFRGS;

Aos professores Yves Denneulin e Brigitte Plateau, pela co-orientação, pela acolhida no ID e em seus grupos de pesquisa e por toda a ajuda durante a estadia na França;

Aos meus pais, José Carlos e Eveline, e à minha irmã, Letícia, por tudo o que me ensinaram e que ainda ensinam, pelos conselhos, pelo apoio e pelo carinho de sempre, pelo incentivo, e por estarem sempre do nosso lado (mesmo quando só queremos incomodar um pouco (ou um muito));

Finalmente, a essa pessoinha incopiável, insubstitucionável que há alguns anos agüenta o “enjoadinho” 24 horas por dia, aceitando “serenamente” os mais variados momentos de chatice, stress, enjoadez, maluquice, emburradice ou mesmo simplesmente de completa embasbaquice de ficar admirando-la... para a minha “chouchoua ERB” esposa Mônica, obrigado pelo $(\text{amor e carinho})^{\infty}$, *jetememonamú*!

A todos, não há palavras para agradecer a amizade e o convívio de vocês!

A Deus, por todas as alternativas anteriores :-)

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS 11

LISTA DE FIGURAS 13

LISTA DE TABELAS 15

RESUMO 17

ABSTRACT 19

1 INTRODUÇÃO 21
1.1 O Modelo Beowulf de Computação Baseada em Clusters 21
1.2 Escalabilidade dos Sistemas de Arquivos 22
1.3 NFSP: Utilizando NFS Eficientemente em Clusters 25
1.4 O Modelo Proposto: dNFSP 25
1.5 Organização do Texto 26

2 SISTEMAS DE ARQUIVOS PARA COMPUTAÇÃO DE ALTO DESEMPENHO 27
2.1 Terminologia 27
2.2 Sistemas de Arquivos Baseados em Discos Compartilhados 28
2.2.1 XFS 28
2.2.2 Frangipani/Petal 29
2.2.3 GFS 30
2.2.4 GPFS 30
2.3 Sistemas de Arquivos Distribuídos 31
2.3.1 NFS – *Network File System* 31
2.3.2 xFS 33
2.3.3 PVFS 34
2.3.4 DPFS 35
2.3.5 Lustre 36
2.4 Avaliação dos Sistemas Apresentados 37
2.4.1 Sistemas Baseados em Discos Compartilhados 37
2.4.2 Sistemas Distribuídos 38
2.4.3 Avaliação Geral 40

3 NFSP: NFS PARALELO 43
3.1 O Projeto NFSP 43
3.2 Estrutura do NFSP 44
3.2.1 IODs 44
3.2.2 Meta-servidor 45
3.3 Descrição das Implementações 46
3.3.1 uNFSP 46
3.3.2 kNFSP 47
3.3.3 Execução com Múltiplos Servidores 48
3.4 Atividades Atuais e Futuras 49

4 O MODELO DE DISTRIBUIÇÃO PROPOSTO: DNFSP 51
4.1 Visão Geral da Solução Proposta 52
4.2 Otimização das Operações de Escrita no NFSP: *dNFSP* 52
4.3 Manutenção da Coerência entre os Meta-Servidores 53
4.3.1 LRC: *Lazy Release Consistency* 54
4.3.2 Adaptação do LRC ao dNFSP 55
4.4 Políticas para Atualização de Meta-Arquivos 56
4.4.1 Busca por Vizinhança 56
4.4.2 Busca Hierárquica 57
4.4.3 Busca Aleatória 57
4.5 Considerações sobre o Modelo Proposto 58

5 VALIDAÇÃO DO DNFSP: IMPLEMENTAÇÃO E MEDIDAS EXPERIMENTAIS 59
5.1 Implementação sobre o uNFSP 59
5.1.1 Identificação dos Meta-Servidores 60
5.1.2 Comunicação entre os Meta-Servidores 60
5.1.3 Adaptação sobre Identificação dos Blocos 62
5.2 Medidas Experimentais 63
5.2.1 Medidas Primárias de Desempenho: discos e rede 63
5.2.2 Leitura e Escrita Seqüenciais 64
5.2.3 *Overhead* de Busca de Meta-Arquivos 67
5.3 *Benchmarks* 68
5.3.1 BTIO 68
5.3.2 Andrew Benchmark Distribuído 70
5.4 Aplicações 72
5.4.1 GADGET 72
5.5 Avaliação dos Resultados Obtidos 73

6 CONCLUSÕES E CONSIDERAÇÕES FINAIS 75
6.1 Metas Alcançadas e Contribuições do Trabalho 75
6.2 Publicações 76
6.3 Trabalhos Futuros 77

REFERÊNCIAS 79

ANEXO A PUBLICAÇÕES 85

ANEXO B RESUMO ESTENDIDO EM LíNGUA FRANCESA 115

LISTA DE ABREVIATURAS E SIGLAS

CFD	Computational Fluid Dynamics
GFLOPS	Giga Floating-Point Operations per Second
GNU	GNU’s Not Unix
HPF	High Performance FORTRAN
IETF	Internet Engineering TaskForce
kB	kilobytes (1024 bytes)
LRC	Lazy Release Consistency
MB	megabytes (1024 kilobytes, ou 1048576 bytes)
MFLOPS	Mega Floating-Point Operations per Second
MPI	Message Passing Interface
MPP	Massively Parallel Processors
NAS	Network Attached Storage
NFS	Network File System
NFSP	NFS Parallèle
NPB	NAS Parallel Benchmark
PVM	Parallel Virtual Machine
RAID	Redundant Array of Inexpensive Disks
SAN	Storage Area Network
SGBD	Sistema Gerenciador de Bases de Dados
SPH	Smoothed Particle Hydrodynamics

LISTA DE FIGURAS

Figura 1.1: Exemplo de limitação de desempenho causado pelo uso de um servidor de arquivos centralizado 23

Figura 2.1: Estrutura de um sistema usando NFS; os clientes devem primeiro montar a partição remota (a1) e posteriormente utilizar o manipulador recebido, bem como uma especificação de localização do arquivo e deslocamento dentro deste, para acessar os blocos de dados (a2) . . 32
Figura 2.2: Estrutura de um sistema xFS; clientes e servidores podem co-existir nos mesmos nós de processamento e a operação é completamente distribuída 34
Figura 2.3: Exemplo de operação no PVFS: o gerenciador é inicialmente contactado para a obtenção dos meta-dados; em seguida o servidor de E/S correspondente é diretamente acessado pelo cliente para executar a operação desejada 35
Figura 2.4: Arquitetura do DPFS 36
Figura 2.5: Visão global do funcionamento do Lustre 37

Figura 3.1: Interações entre clientes, meta-servidor e IODs no NFSP. A requisição de um cliente (passo 1) é enviada ao meta-servidor, que a repassa ao IOD correspondente (passo 2), o qual realiza a operação desejada e envia os resultados (passo 3); “a” e “b” são requisições distintas . . 45
Figura 3.2: Vazão agregada obtida com o uNFSP no i-cluster 47
Figura 3.3: Vazão agregada obtida com o kNFSP 48
Figura 3.4: Esquema de múltiplos meta-servidores com montagens e exportações mescladas de NFS e NFSP 49

Figura 4.1: Arquitetura proposta para a distribuição do meta-servidor 52
Figura 4.2: Envio de mensagens de atualização de segmentos compartilhados utilizando protocolos *eager* e LRC 54
Figura 4.3: Etapas do mecanismo de atualização de meta-arquivos 55
Figura 4.4: Duas situações ilustrando a alocação de duas partições segundo a política de busca por vizinhança: (a) alocação ruim; (b) alocação mais adequada 57

Figura 5.1: Exemplo de arquivos `dms.conf`; em um total de 4 meta-servidores, o arquivo *a)* corresponde ao primeiro meta-servidor, de IP 192.168.0.1, e o arquivo *b)* ao terceiro, cujo IP é 192.168.0.3 60
Figura 5.2: Vazão agregada obtida para leitura seqüencial no dNFSP 65

Figura 5.3: Vazão agregada obtida para escrita seqüencial no dNFSP 67
Figura 5.4: *Overhead* de busca de meta-arquivos 68
Figura 5.5: Comparação de desempenho do dNFSP em relação ao PVFS e ao NFS na execução do *benchmark* BTIO 70
Figura 5.6: Comparação de desempenho do dNFSP em relação ao PVFS e ao NFS na execução do *benchmark* Andrew distribuído 71
Figura 5.7: Resultados obtidos com a execução do GADGET 73

LISTA DE TABELAS

Tabela 2.1: Resumo das características dos sistemas de arquivos apresentados . . 41

Tabela 5.1: Medidas de vazão máxima de transferência de dados pela rede e no acesso aos discos nos clusters i-cluster e LabTeC; os valores são expressos em MB/s (megabytes por segundo) 64

RESUMO

A evolução da Computação Baseada em Clusters, impulsionada pelo avanço tecnológico e pelo custo relativamente baixo do hardware de PCs, tem levado ao surgimento de máquinas paralelas de porte cada vez maior, chegando à ordem das centenas e mesmo milhares de nós de processamento. Um dos principais problemas na implantação de clusters desse porte é o gerenciamento de E/S, pois soluções centralizadas de armazenamento de arquivos, como o NFS, rapidamente se tornam o gargalo dessa parte do sistema. Ao longo dos últimos anos, diversas soluções para esse problema têm sido propostas, tanto pela utilização de tecnologias especializadas de armazenamento e comunicação, como RAID e fibra ótica, como pela distribuição das funcionalidades do servidor de arquivos entre várias máquinas, objetivando a paralelização de suas operações. Seguindo essa última linha, o projeto NFSP (*NFS Parallèle*) é uma proposta de sistema de arquivos distribuído que estende o NFS padrão de forma a aumentar o desempenho das operações de leitura de dados pela distribuição do serviço em vários nós do cluster. Com essa abordagem, o NFSP objetiva aliar desempenho e escalabilidade aos benefícios do NFS, como a estabilidade de sua implementação e familiaridade de usuários e administradores com sua semântica de uso e seus procedimentos de configuração e gerenciamento. A proposta aqui apresentada, chamada de dNFSP, é uma extensão ao NFSP com o objetivo principal de proporcionar melhor desempenho a aplicações que explorem tanto a leitura como a escrita de dados, uma vez que essa última característica não é contemplada pelo modelo original. A base para o funcionamento do sistema é um modelo de gerenciamento distribuído de meta-dados, que permite melhor escalabilidade e reduz o custo computacional sobre o meta-servidor original do NFSP, e também um mecanismo relaxado de manutenção de coerência baseado em LRC (Lazy Release Consistency), o qual permite a distribuição do serviço sem acarretar em operações onerosas de sincronização de dados. Um protótipo do modelo dNFSP foi implementado e avaliado com uma série de testes, *benchmarks* e aplicações. Os resultados obtidos comprovam que o modelo pode ser aplicado como sistema de arquivos para um cluster, efetivamente proporcionando melhor desempenho às aplicações e ao mesmo tempo mantendo um elevado nível de compatibilidade com as ferramentas e procedimentos habituais de administração de um cluster, em especial o uso de clientes NFS padrões disponíveis em praticamente todos os sistemas operacionais da atualidade.

Palavras-chave: Sistemas de arquivos, NFS, gerenciamento de meta-dados, Lazy Release Consistency, computação baseada em clusters.

A Proposal for Distributing the File Server on Clusters

ABSTRACT

The evolution of Cluster Computing, boosted by the advances in technology and relatively low cost of PC hardware, has led to the construction of each time larger and larger parallel machines, featuring hundreds or even thousands of compute nodes. One of the main problems with such large clusters is the management of I/O, since centralised file storage solutions like NFS rapidly become the bottleneck of this part of the system. Over the last years, many solutions to this problem have been proposed, both in the directions of making use of specialised hardware for storage and communication, such as RAID and fibre optics, as well as by distributing the functionalities of the file server over a number of machines, so that its operations may be parallelised. Following this second trend, the NFSP project is a proposal for a distributed file system which extends standard NFS in order to improve the performance of read operations by distributing its functions among several nodes of the cluster. By following this approach, NFSP aims at joining performance and scalability with the benefits of NFS, like implementation stability and both users and administrators familiarity with its semantics and management practices. The proposal presented in this thesis, referred to as dNFSP, is an extension of NFSP with the main goal of providing better performance to applications exploring both read and write operations, since the last one is not tackled by the original model. The basis for the proposed system is a distributed metadata management model, which allows for better scalability and reduces the computational cost on the original NFSP metaserver, and also a relaxed mechanism for maintaining metadata coerence based on LRC (Lazy Release Consistency), which enables the distribution of the service without incurring in costly data sinchronization operations. A prototype of the dNFSP model has been implemented and evaluated by means of a series of tests, benchmarks and applications. The obtained results confirm that the model can be used as a cluster file system, effectively providing better performance to applications and at the same time keeping a high degree of compatibility with standard tools and procedures that cluster administrators are used to, especially the use of standard, unmodified NFS clients which are available on practically all of today's operating systems.

Keywords: File systems, NFS, metadata management, Lazy Release Consistency, cluster computing.

1 INTRODUÇÃO

A sub-área da Computação chamada comumente de *Computação Baseada em Clusters*[1] é uma tendência que vem se desenvolvendo a passos largos nas últimas duas décadas. Tendo tido seu grande impulso no início dos anos 90 (BERKELEY, 1995; STERLING et al., 1995), é uma área que acompanha diretamente o desenvolvimento tecnológico de ponta em equipamentos de processamento, comunicação e armazenamento de dados, e na qual muito se tem investido, tanto em ambientes voltados a pesquisa e desenvolvimento quanto na indústria e no comércio.

As razões para tanto interesse na Computação Baseada em Clusters vêm principalmente de uma característica marcante de grande parte dos clusters implantados no mundo inteiro: o uso de componentes "de prateleira", ou seja, equipamentos amplamente disponíveis comercialmente para o grande público, como os computadores PC, e de software de livre utilização e distribuição, como o sistema operacional GNU/Linux (LINUX, 2005; FREE SOFTWARE FOUNDATION, 2005; OPEN SOURCE INITIATIVE, 2005). Essa característica traz dois grandes benefícios imediatos: o uso de tecnologia atual, pois os componentes de hardware podem ser facilmente substituídos por outros mais atualizados, de melhor desempenho e/ou capacidade, e uma significativa redução de custos, já que nenhuma tecnologia especializada é envolvida e o software é gratuito.

1.1 O Modelo Beowulf de Computação Baseada em Clusters

A abordagem "de prateleira" recém comentada foi utilizada primeiramente pela equipe de Thomas Sterling e Donald Becker, pesquisadores do *NASA's Goddard Space Flight Center*, nos Estados Unidos, em um sistema cujo nome transformou-se praticamente em sinônimo de computação paralela de baixo custo: o cluster *Beowulf* (STERLING et al., 1995; SAVARESE; STERLING, 1999; STERLING et al., 1999; STERLING, 2002). Beowulf é o herói de um antigo poema inglês, no qual é narrada a história de como ele libertou o povo dos Danes do monstro Grendel; a idéia de Sterling e Becker com o cluster Beowulf era libertar os pesquisadores e cientistas da área de Processamento Paralelo e Distribuído do uso de uma classe restrita de máquinas paralelas, que os forçava a um ciclo de desenvolvimento demorado e muitas vezes oneroso financeiramente.

[1] Também freqüentemente referenciada por *Computação Baseada em Agregados* ou *Computação Baseada em Aglomerados*, ou ainda, com a expressão completa em inglês, *Cluster Computing*, *Clustered Computing* ou *Cluster-based Computing*. Na língua portuguesa, as divergências ficam por conta do termo *cluster*, normalmente traduzido para *agregado* ou *aglomerado*. Ao longo deste texto, opta-se pelo uso da palavra não traduzida, cluster, sem o uso do estilo itálico, por entender-se que é um termo já estabelecido entre a comunidade de paralelismo no Brasil e, por isso, identifica mais prontamente ao leitor o tipo de máquina paralela do qual se está falando.

Em 1994, Sterling e sua equipe foram desafiados pela NASA com o projeto de construir uma máquina paralela com 10 Gigabytes de espaço de armazenamento e desempenho de 1 GFLOPS de pico, cujo custo, porém, não deveria ultrapassar um orçamento de cerca de US$50.000,00. Diversos grupos de pesquisa americanos, na época, estavam investindo em máquinas paralelas com nós fracamente acoplados, formadas por redes de estações de trabalho, utilizando software de programação como PVM (GEIST et al., 1994) e o recém surgido padrão MPI (MPI FORUM, 1994). A outra opção, mais tradicional, seria a compra de uma máquina MPP. Em ambos os casos, o orçamento seria ultrapassado em até 10 vezes o valor máximo estipulado devido aos custos das máquinas e dos sistemas operacionais comerciais.

O grupo voltou-se, então, para a já estabelecida nova classe de computadores pessoais, os PCs. Com processadores como o 486 DX4 e o Pentium, já seria possível construir um computador paralelo baseado nesse tipo de máquina que se aproximasse do desempenho necessário. A única peça que faltava para o nascimento do primeiro cluster Beowulf veio pouco tempo depois, com o lançamento da versão 1.0 do Linux.

O primeiro cluster Beowulf era composto de 16 nós com processadores 486 e rede Ethernet 10 Mb/s. Essa máquina era capaz de atingir 42 MFLOPS de desempenho de pico. Embora ainda longe dos 1 GFLOPS solicitados pela NASA, esse desempenho já era comparável ao de máquinas paralelas comerciais da época como a Paragon e a CM-5. Apenas 3 anos depois, com 32 nós de processamento usando processadores Pentium Pro, o desempenho da máquina ultrapassou os 2 GFLOPS, e poucos meses adiante os clusters passaram a figurar na lista TOP500 (TOP500, 2005), que periodicamente relaciona, desde 1993, os 500 computadores mais poderosos do mundo.

Percebe-se, com esse breve relato, o tamanho do impacto que a abordagem da Computação Baseada em Clusters teve na área de Processamento Paralelo e Distribuído; em um intervalo de apenas 4 anos, os clusters passaram do nível de protótipo experimental para concorrentes das máquinas de maior poder computacional do mundo. Desde então, a facilidade de construção e manutenção de clusters do tipo Beowulf vem despertando o interesse de grupos de pesquisa do mundo inteiro, permitindo que praticamente qualquer instituição tenha a disponibilidade de uma máquina paralela para o desenvolvimento de suas pesquisas e execução de aplicações. Essa facilidade, aliada ao rápido progresso tecnológico, levou à construção de clusters cada vez maiores, que hoje chegam facilmente à casa dos milhares de nós, e conseqüentemente originam novos problemas.

1.2 Escalabilidade dos Sistemas de Arquivos

Um dos principais problemas com a construção de clusters de centenas ou mesmo milhares de nós de processamento é o gerenciamento de Entrada/Saída, ou E/S. Em aplicações que manipulam grandes volumes de dados, como previsão meteorológica, simulações químicas e físicas, processamento de imagens, entre outras, é comum a necessidade de troca de dados entre os nós de processamento e destes com um sistema de arquivos, de modo que a informação seja armazenada de forma permanente para posterior recuperação (ex. análise dos resultados calculados). Nesses casos, o desempenho do sistema de arquivos torna-se um ponto-chave para o desempenho da aplicação como um todo. O que freqüentemente ocorre é que a tecnologia empregada no sub-sistema de E/S (discos rígidos, controladoras, cabos) não acompanha a tecnologia dos processadores; em conseqüência, essa parte do cluster torna-se o gargalo do desempenho de muitas aplicações (SCHIKUTA; STOCKINGER, 1999).

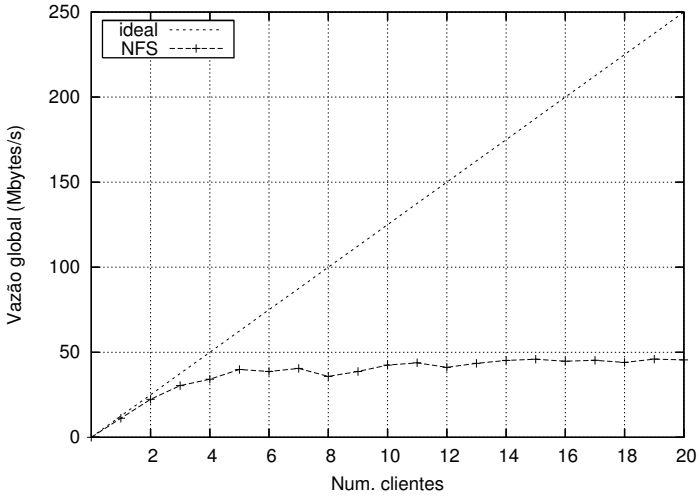

Figura 1.1: Exemplo de limitação de desempenho causado pelo uso de um servidor de arquivos centralizado

Em clusters do tipo Beowulf, o problema é ainda agravado devido ao software não-especializado normalmente empregado. Na maioria dos casos, o sistema de arquivos de um cluster Beowulf é baseado no *Network File System*, ou NFS (SANDBERG et al., 1985). O NFS é o padrão *de facto* para o compartilhamento de arquivos em redes Unix, estando normalmente disponível para uso imediato em sistemas desse tipo e seus derivados, como o GNU/Linux. Assim, o NFS foi a escolha natural para o mundo dos clusters, tanto para armazenamento dos arquivos de usuários quanto para compartilhamento, pelos nós de processamento, de software de sistema como bibliotecas de programação, compiladores, depuradores, entre outros.

Embora o NFS cumpra perfeitamente sua tarefa de prover acesso compartilhado ao sistema de arquivos em uma rede de propósito geral, sua característica centralizada, como apresentado em mais detalhes no próximo capítulo, torna-se facilmente o gargalo do sistema de arquivos em um cluster. Em outras palavras, trata-se de um problema de *escalabilidade*.

Para ilustrar a situação, a Figura 1.1 mostra o resultado de uma experiência com um pequeno programa que, executado em um cluster, faz com que cada nó de processamento crie, simultaneamente com os demais, um arquivo de 64 Mbytes em um sistema de arquivos NFS. O experimento foi executado no cluster LabTeC, uma máquina com 20 nós bi-processados disponibilizada no Instituto de Informática da UFRGS[2]. A curva ilustra o desempenho obtido em termos de vazão global na escrita de dados, medida em Mbytes/s, à medida que se aumenta o número de clientes. A linha tracejada indica o desempenho ideal. Percebe-se facilmente que, a partir de um número ainda pequeno de clientes (entre 5 e 6), a vazão máxima de escrita no sistema de arquivos é atingida, saturando a capacidade do servidor em um ponto muito aquém do desempenho ideal. Pode-se ter uma idéia, assim, do impacto causado por um servidor de arquivos centralizado no desempenho de uma aplicação que use centenas ou milhares de nós.

Esse tipo de problema de escalabilidade não é exclusivo dos clusters Beowulf, e nem

[2]Cada nó apresenta dois processadores Pentium III de 1 GHz, 1 Gbyte de memória RAM, um disco rígido SCSI de 18 Gbytes e duas placas de rede Ethernet 1000 Mb/s, uma delas conectada a um switch 3Com com portas 10/100. O cluster foi implantado no âmbito de um convênio entre o Instituto e a Dell Computers do Brasil.

mesmo dos clusters em geral. Na verdade, qualquer máquina paralela com um grande número de nós pode ser afetada devido às mesmas razões. A diferença, principalmente no caso de máquinas paralelas comerciais, é que os fabricantes já incluem no projeto da máquina um sub-sistema de E/S capaz de suportar adequadamente a carga imposta pelas aplicações ou tipos de aplicações previstas. Assim, praticamente todas as máquinas paralelas comerciais apresentam um sistema de arquivos próprio, cuidadosamente projetado para não restringir demasiadamente o desempenho do sistema. Como mostrado no Capítulo 2, essas soluções utilizam, em geral, algum tipo de tecnologia avançada de comunicação e armazenamento, como fibra ótica e *arrays* de discos do tipo RAID (PATTERSON; GIBSON; KATZ, 1988).

A mesma abordagem, entretanto, nem sempre é válida para um ambiente de clusters, principalmente para a classe Beowulf. Tecnologias mais eficientes de comunicação e armazenamento foram portadas e mesmo desenvolvidas para a arquitetura dos PCs. A segunda metade dos anos 90 marcou o surgimento de placas ISA e PCI para tecnologias de comunicação como Fast Ethernet (INSTITUTE OF ELECTRICAL AND ELECTRONIC ENGINEERS, 1995), Myrinet (BODEN et al., 1995), SCI (INSTITUTE OF ELECTRICAL AND ELECTRONIC ENGINEERS, 1992; HELLWAGNER; REINEFELD, 1999), Gigabit Ethernet (INSTITUTE OF ELECTRICAL AND ELECTRONIC ENGINEERS, 1998) e Quadrics (QSNET HIGH PERFORMANCE INTERCONNECT, 2003), todas muito utilizadas em clusters no mundo inteiro. Controladoras de discos SCSI e RAID para PCs foram também desenvolvidas por empresas como Compaq, HP, IBM e Dell. No entanto, a barreira para a utilização de tais componentes em clusters quase sempre recai no orçamento. Por exemplo, o custo aproximado de implantação de tecnologia Myrinet em um cluster, atualmente, fica em torno de US$1.000,00 por nó de processamento, o que equivale ao custo de um PC completo[3]. Muitos grupos de pesquisa, nesse caso, preferem instalar o dobro de nós ao invés de investir numa tecnologia como Myrinet, e utilizar conexões de rede menos onerosas como a Fast Ethernet, arcando com uma capacidade de comunicação entre nós no mínimo 10 vezes mais lenta.

Ao invés de investir em tecnologias avançadas que, apesar de disponíveis, apresentam custo elevado e até mesmo contrariam a "filosofia" Beowulf, diversos grupos de pesquisa partiram para outra direção na busca de soluções para o problema da E/S em clusters: a distribuição do sistema de arquivos.

Desde os primeiros projetos de máquinas paralelas virtuais sobre redes de estações, como o NOW de Berkeley, protótipos de sistemas de arquivos distribuídos vêm sendo projetados e implementados. A idéia principal é distribuir as funcionalidades do sistema de arquivos entre várias máquinas de uma rede ou de um cluster, de modo que, atuando em paralelo, essas máquinas consigam realizar as tarefas normais de um sistema de arquivos em menos tempo. Outras vantagens diretas são a melhoria na escalabilidade, pela disponibilização de um número maior de pontos de acesso ao sistema de arquivos, e também um melhor aproveitamento de recursos, pois o espaço em disco eventualmente não utilizado dos nós do cluster pode agora fazer parte do espaço global de armazenamento. Atualmente, projetos que seguem essa abordagem começam a atingir um nível de estabilidade e maturidade suficiente para seu uso em clusters de produção. É o caso, por exemplo de sistemas como o PVFS (CARNS et al., 2000) e o Lustre (CLUSTER FILE SYSTEMS, INC., 2002). O Capítulo 2 apresenta tais sistemas e suas características em mais detalhes.

[3]Dados de janeiro de 2005

1.3 NFSP: Utilizando NFS Eficientemente em Clusters

Dentre os projetos de sistemas de arquivos distribuídos da atualidade, é possível identificar um certo número de iniciativas que visam, entre outros objetivos, manter um nível de compatibilidade com o software tradicionalmente empregado na implantação de clusters. Nesse caso, em especial para os clusters Beowulf, vários projetos baseiam sua abordagem sobre extensões ao protocolo NFS padrão. A justificativa para sistemas desse tipo é que, sendo baseados em métodos, protocolos e implementações já existentes, o projeto tende a diminuir o impacto de introdução de uma nova tecnologia, e também pode tirar proveito da maturidade de módulos de software já bem estabelecidos como é o caso do NFS. Um dos projetos que segue esse raciocínio é o NFSP — *NFS Parallèle*.

O NFSP é um projeto originado no *Laboratoire Informatique et Distribution*, ou ID ("idéia", em francês), vinculado ao IMAG — *Institut d'Informatique et Mathématiques Apliquées de Grenoble*, na França — e que vem sendo desenvolvido desde 2001 por alunos de doutorado, DEA (etapa do sistema educacional francês semelhante à especialização) e estagiários do laboratório, sob orientação do Prof. Yves Denneulin. A partir de 2002, em função dos interesses comuns de pesquisa e do longo histórico de cooperações científicas entre os grupos dos dois países, o Grupo de Processamento Paralelo e Distribuído do II/UFRGS passou a cooperar com o desenvolvimento do NFSP, com o estabelecimento da orientação em co-tutela da presente tese. Durante o ano de 2003, o trabalho foi realizado em Grenoble, nas dependências do ID, como estágio-sanduíche, onde foi concretizada a Proposta de Tese e realizados os primeiros experimentos práticos.

O objetivo principal do NFSP é estender o NFS convencional de forma a melhorar seu desempenho e escalabilidade, tornando-o um sistema de arquivos adequado para uso em clusters, mas também mantendo compatibilidade com o restante do software tradicional de um cluster, em especial os clientes NFS, aos quais cabe o papel principal na maior parte dos nós de processamento. No NFSP, os clientes NFS são mantidos de forma intacta, reduzindo significativamente a quantidade de configuração e manutenção do sistema de arquivos.

Várias frentes de pesquisa estão sendo conduzidas, atualmente, no projeto NFSP, abrangendo desde a otimização da comunicação interna do servidor até a adaptação do sistema para ambientes de *grid computing*. Uma dessas frentes é a proposta apresentada neste trabalho.

1.4 O Modelo Proposto: dNFSP

Como exposto anteriormente, o principal problema de uso do NFS em clusters é sua falta de escalabilidade, originada pela existência de um servidor central. O NFSP, conforme apresentado posteriormente no Capítulo 3, resolve boa parte desse problema, permitindo que operações de leitura de dados por parte dos clientes sejam feitas de forma completamente distribuída, e conseqüentemente com melhor desempenho. As operações de escrita, no entanto, ainda são mantidas vinculadas ao servidor centralizado, o que causa novamente o problema de gargalo no caso de ocorrência maciça desse tipo de operação.

Dada essa situação, caracterizou-se o desafio para a elaboração desta tese: desenvolver um modelo de distribuição do NFS, adaptando a abordagem do NFSP, que permita a realização eficiente de operações de acesso ao sistema de arquivos, tanto no caso da leitura como no da escrita, em clusters do tipo Beowulf (ou seja, sem a utilização de tecnologias especializadas), e mantendo as principais diretivas do projeto como a compatibilidade

com os clientes NFS convencionais. Com esse novo modelo, o NFSP torna-se apto a suportar um conjunto maior de aplicações paralelas, adicionando ao esforço já realizado a capacidade de executar uma nova classe de programas. O resultado desse trabalho é, assim, apresentado nas páginas seguintes.

1.5 Organização do Texto

O trabalho desenvolvido é apresentado no restante desta monografia de acordo com a seguinte organização:

Revisão do Estado da Arte e levantamento bibliográfico (Cap. 2) São apresentados diversos sistemas de arquivos para ambientes concorrentes e ressaltadas suas principais características, com o objetivo de identificar técnicas e métodos que beneficiem a obtenção de melhor desempenho para aplicações que realizam E/S em clusters.

Descrição do NFSP (Cap. 3) Este capítulo apresenta o projeto NFSP e expõe as principais características do modelo de distribuição proposto, bem como os resultados obtidos com as implementações realizadas, como forma de embasar o entendimento do trabalho posteriormente desenvolvido.

Apresentação do modelo proposto na tese: dNFSP (Cap. 4) Aqui é apresentada a contribuição principal do trabalho desenvolvido. O modelo dNFSP é detalhado, evidenciando as extensões feitas ao NFSP e justificando a motivação para suas características com base nos sistemas estudados.

Resultados experimentais e validação do modelo dNFSP (Cap. 5) Neste capítulo são apresentados detalhes da implementação de um protótipo do modelo dNFSP, baseado na versão *user-level* do NFSP, e os resultados experimentais obtidos com a execução de testes, *benchmarks* e aplicações utilizando o sistema.

Avaliação do trabalho, conclusões e trabalhos futuros (Cap. 6) Fechamento do texto com uma avaliação do trabalho que foi realizado, as principais conclusões e perspectivas de trabalhos futuros como forma de seguimento às atividades no NFSP.

2 SISTEMAS DE ARQUIVOS PARA COMPUTAÇÃO DE ALTO DESEMPENHO

A busca de soluções para o armazenamento eficiente de dados em máquinas paralelas é um problema que existe praticamente desde o surgimento da computação concorrente, pois problemas com grande volumes de dados a serem analisados (simulações numéricas, astrofísica, modelagem molecular, entre outros) são objetos constantes do mundo científico, e freqüentemente a própria motivação para o desenvolvimento de um novo método. Assim, diversas soluções de otimização dos sistemas de arquivos, empregando variadas técnicas, já foram propostas. Neste capítulo, apresenta-se um panorama de tais soluções, procurando evidenciar as mais influentes na atualidade e que melhor se adaptam ao cenário de computação com clusters no qual o trabalho está inserido.

Como forma de organização da apresentação, os sistemas de arquivos avaliados são classificados em dois grupos:

- *Baseados em Discos Compartilhados:* um ou mais dispositivos de armazenamento são compartilhados por todos os nós de processamento, de forma semelhante à memória compartilhada em uma máquina SMP; o aumento de desempenho é obtido principalmente pelo uso de equipamentos dedicados (ex. redes de fibra ótica) para a comunicação entre os nós de processamento e para a leitura e escrita dos dados nos discos, além do próprio paralelismo nas operações de acesso.

- *Distribuídos:* os dados e funcionalidades do sistema de arquivos são distribuídos entre os processadores, de modo que os acessos são feitos de forma concorrente e, em conseqüência, podem ser realizados em menos tempo; em geral, sistemas deste grupo não exigem tecnologias avançadas de comunicação e/ou armazenamento, embora possam tirar proveito desses recursos quando disponíveis.

Nas seções seguintes são apresentados, de acordo com esta classificação, alguns dos sistemas de arquivos mais freqüentemente referenciados na literatura, os quais reúnem as principais características necessárias para aplicações de alto desempenho. Ao final, apresenta-se um quadro comparativo de tais sistemas.

2.1 Terminologia

A literatura sobre sistemas de arquivos emprega diversos termos específicos, próprios dessa área de pesquisa. Alguns dos mais utilizados são listados aqui para maior clareza.

Dados Referem-se à informação armazenada pelos usuários em seus arquivos; em outras palavras, são os dados visíveis pelos usuários. Pode ser comparado ao *payload* de uma mensagem no contexto do MPI, por exemplo.

Meta-dados São os “dados sobre os dados”, ou seja, informação de controle que, juntamente com os dados, forma a noção completa de um arquivo. Exemplos de meta-dados são o tamanho de um arquivo, identificação de proprietário e grupo, permissões de acesso, data de modificação, entre outros. Os meta-dados normalmente precisam ser lidos antes que os dados de um arquivo possam ser acessados, por exemplo para localizar o bloco de dados no disco, ou verificar se o acesso é válido em relação às permissões; em sistemas de arquivos distribuídos, os meta-dados também indicam em que máquina na rede os dados estão armazenados e de que forma eles estão distribuídos.

Meta-arquivo Alguns sistemas de arquivos distribuídos armazenam os meta-dados de arquivos remotos sob forma de arquivos no sistema local; tais arquivos são normalmente chamados de meta-arquivos.

Servidor de E/S (*I/O daemon*) Também freqüentemente usados por sistemas de arquivos distribuídos, são servidores dedicados a ler e armazenar blocos de dados nos discos locais. Normalmente, vários servidores de E/S são utilizados em paralelo para otimizar os tempos de acesso a disco nos sistemas que os empregam.

Servidor de meta-dados De forma complementar ao item anterior, é comum a implantação de servidores dedicados a prover os meta-dados.

Striping Refere-se à técnica de dividir os dados de um arquivo em blocos, ou “fatias”, as quais são distribuídas em servidores de E/S. Além da otimização dos tempos de leitura e escrita dos dados, esta técnica também favorece a implementação de mecanismos de tolerância a falhas (por exemplo, pela replicação do mesmo bloco de dados em mais de um servidor).

NAS (*Network-Attached Storage*) Este termo é empregado para designar unidades de armazenamento permanente que apresentam capacidade de comunicar-se diretamente através da rede, por exemplo através de uma porta Ethernet em uma rede TCP/IP.

SAN (*Storage Area Network*) Denominação utilizada quando um conjunto de discos (normalmente possuindo recursos de NAS) é empregado em um mesmo sistema de forma a prover uma área de grande capacidade de armazenamento; uma SAN pode ser vista por um sistema como um único dispositivo lógico de armazenamento cuja capacidade equivale à soma das capacidades individuais.

2.2 Sistemas de Arquivos Baseados em Discos Compartilhados

Neste grupo de sistemas, o aumento em desempenho é obtido pelo uso de uma tecnologia dedicada de comunicação de dados, como fibra ótica, e/ou de armazenamento, como RAID, no acesso a um ou mais discos compartilhados entre os nós de processamento. Alguns dos mais representativos sistemas de arquivos deste grupo são apresentados a seguir.

2.2.1 XFS

O XFS (SWEENEY et al., 1996) é um sistema de arquivos desenvolvido para o IRIX, da Silicon Graphics — SGI — como sucessor do EFS, e vem sendo disponibilizado desde 1994.

O objetivo principal da SGI com o XFS é o suporte a arquivos de grande porte, da ordem dos petabytes, além de alto desempenho e escalabilidade. A base para a obtenção dessas características vem do uso de árvores B+ como estrutura de dados principal do sistema, o que permite eficiência na manipulação de blocos de dados, em especial na gerência de espaço livre. O suporte a arquivos e diretórios de grande porte é obtido com o uso de ponteiros de 64 bits. O sistema apresenta ainda um esquema de registro (*log*) de transações semelhante ao utilizado em bases de dados, o que permite que falhas eventuais não deixem a estrutura de diretórios e arquivos em disco em um estado inconsistente, e também propicia uma forma rápida de recuperação.

Embora seja um sistema de arquivos de propósito geral, para ser usado tanto em servidores quanto estações de trabalho comuns, o XFS inclui características que favorecem aplicações de alto desempenho. Uma delas é a estrutura modularizada da implementação, que permite acesso concorrente por vários processos a praticamente todas as funções do sistema de arquivos (somente o gerenciador de transações é centralizado), o que é especialmente desejável em máquinas SMP. Outra característica é o suporte a E/S direta (*direct I/O*). Esse recurso permite que as aplicações tenham acesso de forma direta e eficiente aos discos, o que pode representar um aumento significativo de desempenho na transferência de grandes volumes de dados e/ou quando a unidade de armazenamento permanente é composta de vários discos em *array* (ex. RAID).

O XFS foi recentemente portado para o Linux e com isso tem sido distribuído no núcleo padrão desse sistema.

2.2.2 Frangipani/Petal

Embora não exija diretamente a utilização de alguma tecnologia avançada para seu funcionamento, o sistema Frangipani/Petal (THEKKATH; MANN; LEE, 1997) é incluído neste grupo devido à sua estrutura de funcionamento. Distinguem-se duas camadas; a inferior, Petal, provê um serviço distribuído de armazenamento com capacidade de escalabilidade e tolerância a falhas; a superior, Frangipani, implementa o sistema de arquivos propriamente dito, criando as abstrações de arquivos e diretórios sobre a área de armazenamento da camada inferior, apresentando também capacidade de tolerância a falhas.

A camada Petal implementa um serviço de *disco virtual*. Assim como um disco físico, o disco virtual é um espaço de armazenamento no qual são escritos e lidos blocos de dados. Entretanto, um disco virtual é composto de vários discos físicos localizados em máquinas distintas. Servidores executando em cada uma dessas máquinas se comunicam através da rede de modo a coordenar a distribuição das operações de leitura e escrita. Opcionalmente, a camada também pode ser configurada para realizar a replicação de dados, como mecanismo de tolerância a falhas, e na geração eficiente de *backups*.

Na camada superior, o sistema Frangipani oferece às máquinas cliente uma visão indistinta de um conjunto de arquivos e diretórios, utilizando o espaço de armazenamento do Petal como se fosse um único dispositivo de armazenamento. Múltiplos servidores Frangipani podem ser executados nas máquinas da rede/cluster, portanto o sistema implementa um mecanismo distribuído de travas que garante a consistência dos dados.

Uma característica importante do Frangipani é a utilização de registros de transações na escrita de dados, de modo que uma falha em um servidor pode ser recuperada pelos demais, e o sistema pode continuar operando com os servidores que restam. Essa recuperação é feita de forma transparente.

Resultados práticos com o sistema revelam bom desempenho em um cluster com poucos nós de processamento, mas escalabilidade apenas regular. Os autores mostram (1997)

medidas realizadas com *benchmarks* em um cluster com 7 nós, e os resultados obtidos, embora não diretamente constatado pelos mesmos, sugerem que o mecanismo distribuído de travas imponha demasiada complexidade sobre os nós e sobre a rede, reduzindo o aumento de desempenho. Neste caso pode-se imaginar que uma tecnologia de comunicação mais rápida como a Myrinet possa permitir a utilização eficiente de um número maior de nós. O sistema, entretanto, foi um dos primeiros a se aproximar da idéia de SAN, e parece ter inspirado diversos projetos no final dos anos 90, vista a quantidade relativamente grande de citações do referido artigo.

2.2.3 GFS

O *Global File System*, ou GFS (SOLTIS et al., 1998; PRESLAN et al., 1999) é um sistema de arquivos de alto desempenho desenvolvido pela Universidade de Minnesota, tendo sido iniciado em 1995. O objetivo principal do GFS é aproveitar as características de baixa latência e alta vazão de tecnologias de rede como Fibre Channel e Gigabit Ethernet para transformar um conjunto de discos em uma área de memória compartilhada acessível diretamente pelos clientes. Em outras palavras, o GFS elimina o papel de um servidor central, compondo o que os autores chamam de "sistema multi-cliente simétrico".

Os dispositivos de armazenamento no GFS tem a característica NAS, de modo que cada dispositivo (ex. disco rígido, volume RAID) é diretamente acessível através da rede IP, formando uma SAN (*Storage Area Network*). Esses dispositivos formam o que os autores chamam de *Network Storage Pool*, uma união lógica dos dispositivos que implementa a visão de espaço de memória compartilhado. Cada cliente pode, portanto, acessar o sistema de arquivos de forma independente dos demais. Com o uso de uma rede de alto desempenho, e pela distribuição dos dados nos diversos discos, os acessos são otimizados.

Uma das metas que orientou o projeto do GFS foi, também, a capacidade de escalabilidade do sistema. Por isso, mecanismos elaborados de controle foram projetados e incorporados ao longo dos anos. Podem ser citados, dentre os mais importantes, os *device locks*, que permitem ações atômicas de granularidade fina sobre os dispositivos de armazenamento, evitando perda de desempenho por contenção, os *stuffed dinodes*, que consistem na inclusão de dados nos próprios *i-nodes* de metadados, e o *extendible hashing*, um mecanismo eficiente para o registro e a localização de informações sobre os arquivos.

Como resultado dessa combinação de características, o GFS apresenta-se como um sistema de arquivos com efetivo alto desempenho, tendo motivado o surgimento da Sistina Software, empresa que passou a deter os direitos comerciais sobre o sistema a partir de 2001. Um projeto paralelo, chamado de OpenGFS, está dando continuidade à versão livre originada em Minnesota.

O GFS foi inicialmente implementado para a plataforma IRIX. Desde 1998, entretanto, o sistema foi portado para o Linux, no qual o principal eixo de desenvolvimento vem sendo conduzido.

2.2.4 GPFS

O GPFS, ou *General Parallel File System* (SCHMUCK; HASKIN, 2002), é um sistema de arquivos paralelo desenvolvido pela IBM a partir do final dos anos 90, sendo o sucessor do Tiger Shark, concebido para sistemas multimídia. É o sistema de arquivos utilizado no ASCI White. O objetivo principal da IBM com o desenvolvimento do GPFS foi suprir necessidades de vazão, capacidade de armazenamento e confiabilidade exigidas pelos problemas de maior porte e complexidade.

A arquitetura de base do GPFS é um sistema *shared-disk* (disco compartilhado), onde todos os clientes acessam os discos diretamente e todos têm igual capacidade de acesso. Os discos podem ser diretamente conectados à rede, se possuírem recursos de NAS, ou então gerenciados por um servidor local, o qual faz a interface com clientes. Em qualquer dos dois casos, o protocolo de acesso aos discos é definido por funções convencionais de acesso a blocos de dados.

O sistema permite acesso completamente paralelo tanto a um conjunto de arquivos como a blocos distintos de dados dentro de um mesmo arquivo. O tamanho dos blocos é intencionalmente grande (256 kB por padrão, mas configurável entre 16 kB e 1 MB) de modo a minimizar as perdas com atraso de busca (*seek overhead*). Para o caso de arquivos pequenos, também é possível utilizar *sub-blocos*, que podem ter até $1/32$ do tamanho de um bloco normal.

Como cada cliente acessa o conjunto de discos de forma independente, é necessário estabelecer um mecanismo de controle de acesso. Isso é feito através de um sistema distribuído de travas, garantindo a consistência do sistema. Cada nó de processamento executa um gerenciador local de travas, que se comunica com um gerenciador global, executado em um dos nós, para obter *tokens* de travamento.

2.3 Sistemas de Arquivos Distribuídos

Nesta seção, são apresentados os sistemas de arquivos que buscam obter aumento de desempenho pela distribuição de tarefas entre diversas máquinas. Apesar de não ser um sistema de arquivos projetado para o processamento paralelo, o NFS é apresentado inicialmente devido à sua larga difusão e utilização na computação com clusters. Na seqüência, são descritos sistemas projetados especificamente para a computação concorrente.

2.3.1 NFS – *Network File System*

O NFS, ou *Network File System* (CALLAGHAN; PAWLOWSKI; STAUBACH, 1995), é o padrão *de facto* para compartilhamento de arquivos em ambiente Unix, e por conseqüência foi naturalmente absorvido pela comunidade de clusters desde seus primeiros passos (STERLING et al., 1999). O NFS foi desenvolvido pela Sun Microsystems e surgiu em 1985 com o lançamento do SunOS 2.0. Embora tenha sido a primeira versão pública, esta já era numerada 2.0, uma vez que a versão 1 foi utilizada apenas internamente na Sun. A versão em uso atualmente é a 3.0, disponível por padrão em todas as distribuições Linux modernas, e é a normalmente usada em clusters Beowulf. A versão 4 encontra-se em processo de padronização como RFC, tendo a Sun cedido o controle de modificações no NFS à IETF (*Internet Engineering Task Force*) em 1998.

O NFS é, na verdade, um protocolo para acesso remoto transparente de um cliente ao sistema de arquivos de um servidor; seu objetivo primário não é o alto desempenho, e sim permitir que usuários em um sistema distribuído possam ter acesso a seus arquivos pessoais de qualquer ponto do sistema (ex. estações de trabalho), assim como fazem outros sistemas de arquivos distribuídos como AFS (SATYANARAYANAN, 1990) e Coda (SATYANARAYANAN et al., 1990).

A operação é dividida em duas partes: *montagem* e *acesso*. Antes que qualquer operação seja realizada, o sistema de arquivos remoto deve ser *montado*, ou seja, associado a algum ponto do sistema de arquivos local. Esta operação fornece ao cliente um manipulador para acessos subseqüentes.

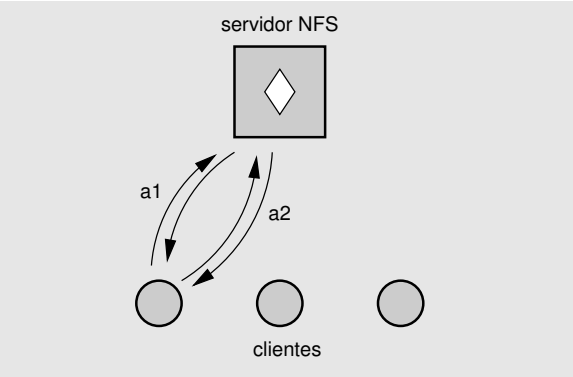

Figura 2.1: Estrutura de um sistema usando NFS; os clientes devem primeiro montar a partição remota (a1) e posteriormente utilizar o manipulador recebido, bem como uma especificação de localização do arquivo e deslocamento dentro deste, para acessar os blocos de dados (a2)

O sistema é projetado de modo a ser sem-estado[1]. Toda requisição de um cliente, após realizar a montagem da partição remota, deve estar completamente identificada com localização do arquivo, deslocamento e tamanho do bloco de dados a ser manipulado (nesse sentido, o servidor NFS pode ser visto como ambos gerenciador de meta-dados e servidor de dados no mesmo processo). A Figura 2.1 ilustra uma operação no sistema. Toda requisição proveniente de um cliente é recebida, tratada e respondida diretamente pelo servidor NFS.

O fato de o NFS ser sem-estado é favorável à tolerância a falhas. Devido a esse modelo de funcionamento, falhas dos clientes não afetam o servidor, e falhas do servidor são toleradas de maneira simples e transparente pelos clientes, sendo o funcionamento retomado normalmente quando aquele volta a operar. Por esta razão, o NFS foi amplamente difundido nos ambientes Unix. Outras características importantes do sistema incluem o suporte a heterogeneidade entre servidores e clientes e uma semântica de acesso própria (ou seja, não específica de um ou outro sistema operacional, favorecendo a portabilidade).

Apesar das mencionadas vantagens, o NFS torna-se um problema potencial de escalabilidade em clusters de grande porte, em função do modelo centralizado. No caso de aplicações com E/S intensa, que fazem uso extensivo do sistema de arquivos, um servidor NFS centralizado limita o desempenho máximo alcançável em pelo menos três níveis: *i*) a conexão de rede do servidor, *ii*) o disco do servidor, e *iii*) a CPU do servidor. Por esta razão, muita pesquisa vem sendo desenvolvida tanto no sentido de aprimorar o NFS como na busca de novas soluções.

O *NFSˆ2* (MUNTZ, 2001) é uma arquitetura de integração de vários servidores NFS sob o mesmo espaço de nomeação. Um conjunto de servidores NFS convencionais é usado como um repositório de partições; entre esse repositório e os clientes é introduzido um balanceador de carga que analisa as requisições recebidas e as direciona para o servidor correspondente, desvinculando a localização visível de um arquivo ou diretório de sua localização física. A vantagem do NFSˆ2 é permitir a adição ou remoção de servidores de forma transparente aos clientes; por outro lado, a arquitetura ainda mantém um ponto de acesso centralizado (o balanceador de carga), não sendo adequado para o processamento paralelo em função da baixa escalabilidade.

D-NFS (LIU et al., 2000), abreviação para *Distributed NFS*, é um projeto desenvol-

[1]O NFS v4 passará a manter estado.

vido na Universidade Tsinghua de Pequim, China. A base do funcionamento do sistema é a distribuição do servidor NFS tradicional em vários processos, cada um executando em uma máquina distinta, onde cada processo atende somente um sub-conjunto do total de clientes. Os vários processos podem trocar informações entre si, por exemplo no caso de um cliente tentar acessar um arquivo que está armazenado em outro servidor. Nesse caso, os meta-dados são buscados do servidor remoto e armazenados na *Name and Attribute Cache*, ou NACache (LIU et al., 2000), de modo a otimizar acessos futuros. A separação do gerenciamento de dados e de meta-dados permite também a replicação e migração dos arquivos. Com essas características, o D-NFS cumpre o objetivo de prover maior área de armazenamento a clientes NFS tradicionais, mas não é adequado para ambientes de alto desempenho como clusters. Por exemplo, no caso de um cliente tentar acessar um arquivo não disponível no servidor ao qual está conectado, é necessário que o conteúdo completo do arquivo seja copiado de um servidor remoto, o que acarreta em uma operação demorada de transferência de dados via rede.

Seguindo uma abordagem diferente, o Bigfoot-NFS (KIM; MINNICH; MCVOY, 1994) opta pela modificação do software cliente em vez de alterar o servidor. O envio de requisições por parte dos clientes é feito através de uma biblioteca de RPCs em vetor, onde o mesmo serviço pode ser enviado a diversas máquinas ao mesmo tempo. O servidor de arquivos, no Bigfoot-NFS, consiste de um grupo de servidores NFS tradicionais, completamente independentes uns dos outros. Portanto, toda a noção de um sistema de arquivos global é implementada no lado do cliente. Assim como no NFSˆ2 e no D-NFS, o Bigfoot-NFS não realiza nenhum espécie de *striping*, de modo que cada servidor armazena arquivos completos. Enquanto essa característica favorece o gerenciamento de meta-dados, pois não há replicação, é possível a ocorrência de gargalos de vários clientes necessitam do mesmo conjunto de arquivos ao mesmo tempo. Assim, o Bigfoot-NFS caracteriza-se também como um sistema para permitir o compartilhamento de um maior volume de dados, e não para ambientes de processamento concorrente.

2.3.2 xFS

O *Berkeley xFS*[2] (ANDERSON et al., 1995) é um tentativa de sistema de arquivos "sem servidor" desenvolvido na Universidade da Califórnia em Berkeley de 1993 a 1995. Ele é baseado em diversos esforços de pesquisa da época, principalmente *RAID*, *LFS(Log-structured File System)*, *Zebra* e *Multiprocessor Cache Consistency* (todos descritos no mesmo artigo de Anderson et al. (1995)). O xFS apresenta uma estrutura completamente distribuída, onde dados e meta-dados encontram-se difundidos entre as máquinas disponíveis (que podem ser a totalidade ou somente parte dos recursos disponíveis) e ainda possuem a capacidade de migração.

O sistema é composto de 4 tipos de entidades: *servidores de armazenamento*, *gerenciadores*, *limpadores* e *clientes*. Os servidores de armazenamento são servidores de E/S, ou seja, responsáveis pela leitura e escrita de blocos de dados nos discos; os gerenciadores são servidores de meta-dados; e os clientes são as entidades que usam o sistema de arquivos. Os limpadores são um tipo de funcionalidade particular do xFS, não encontrado em outros sistemas de arquivos, exercendo a função de desfragmentar e reciclar o espaço de armazenamento de modo a prover espaço para futuras operações de escrita.

Não há restrição quanto a qual funcionalidade deve ser executada em qual máquina; estas podem abrigar os 4 tipos de entidades sem distinção ou tê-las agrupadas por al-

[2]Não confundir com o XFS (com "X" maiúsculo) da SGI.

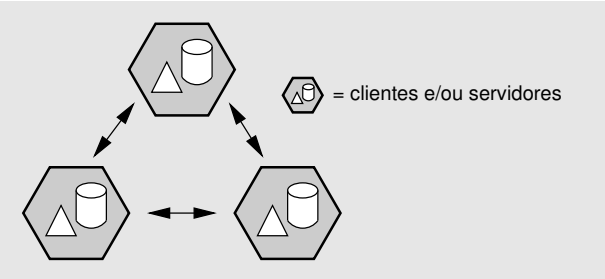

Figura 2.2: Estrutura de um sistema xFS; clientes e servidores podem co-existir nos mesmos nós de processamento e a operação é completamente distribuída

gum critério que se deseje. Na Figura 2.2 é apresentada uma visão simplificada do xFS, ilustrando uma situação onde todas as máquinas possuem igual funcionalidade.

Um mecanismo de cache cooperativa é estabelecido com o objetivo de reduzir o atraso com a busca de dados e meta-dados. O princípio desse mecanismo é que um cliente pode obter informações que já se encontrem disponíveis em outro cliente. Isso forma uma cache de 3 níveis no xFS, com diferentes graus de atraso: um determinado bloco de dados pode estar na cache local de um nó (atraso da cache somente), na cache de outro cliente (atraso de cache e rede) ou ainda nos discos (atrasos de cache, rede e disco).

Um protótipo do xFS foi implementado e testado entre 1994 e 1995 em um cluster de 32 nós SPARCStation, utilizando TCP/IP sobre Myrinet como tecnologia de comunicação. Os resultados obtidos revelaram-se promissores tanto em termos de desempenho como de escalabilidade, exibindo comportamento praticamente linear até o número de nós disponível.

Apesar dos bons resultados, o xFS foi descontinuado com o encerramento do projeto NOW, e não se tem conhecimento de que tenha sido portado para Linux até os dias atuais.

2.3.3 PVFS

PVFS — *Parallel Virtual File System* (CARNS et al., 2000) — é um projeto conjunto entre o Parallel Architecture Research Laboratory, da Universidade Clemson, e o Argonne National Laboratory, ambos nos Estados Unidos. O objetivo do PVFS é prover um sistema de arquivos de alto desempenho para a classe de máquinas paralelas Beowulf, sendo, dentro desta filosofia, especificamente projetado para utilizar componentes (principalmente rede e discos) convencionais.

O sistema pode ser visto como uma versão simplificada do xFS. O acesso ao armazenamento permanente é realizado por servidores de E/S (*I/O daemons*), e um gerenciador único (e portanto central) é responsável por armazenar e difundir informações de meta-dados. Os clientes dispõem de três métodos para acessar o sistema de arquivos: por meio de uma API dedicada, por meio da interface POSIX padrão (uma vez que um módulo VFS do Linux é também disponibilizado, permitindo a montagem direta do sistema) ou ainda por MPI-IO.

Cada arquivo no PVFS é dividido entre múltiplos servidores de E/S através de *striping*. O acesso a um arquivo é dividido em duas etapas, como ilustrado na Figura 2.3. Inicialmente, o cliente contacta o gerenciador (passo 1) para obter uma especificação da localização do arquivo: servidor de E/S inicial, número de divisões, e ID local. Juntamente com o tamanho do arquivo e o tamanho padrão das “fatias” de *striping*, qualquer acesso a um determinado bloco de dados pode ser precisamente especificado. Posterior-

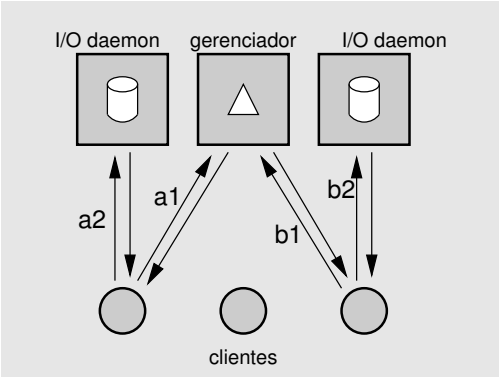

Figura 2.3: Exemplo de operação no PVFS: o gerenciador é inicialmente contactado para a obtenção dos meta-dados; em seguida o servidor de E/S correspondente é diretamente acessado pelo cliente para executar a operação desejada

mente, o cliente contacta os servidores de E/S diretamente de modo a executar a operação desejada. O gerenciador não precisa ser contactado novamente para operações sobre o mesmo arquivo.

Alguns experimentos realizados com o PVFS em um cluster de 256 nós com tecnologias de rede Fast Ethernet e Myrinet revelam resultados muito bons, tanto em termos de desempenho quanto de escalabilidade. Em ambas as tecnologias, o desempenho obtido acompanha os limites do meio de comunicação, com escalabilidade praticamente linear. Um ponto negativo do PVFS é a existência de um único gerenciador centralizado, comprometendo a capacidade de tolerância a falhas do sistema. Uma nova versão, o PVFS 2, está em curso de desenvolvimento, e objetiva tratar essa dentre outras questões.

2.3.4 DPFS

O *Distributed Parallel File System* (SHEN; CHOUDHARY, 2001) é um sistema de arquivos desenvolvido na Northwestern University, nos Estados Unidos, voltado a aplicações científicas de grande porte. O DPFS também é baseado em uma arquitetura cliente/servidor, onde os clientes realizam a leitura e a escrita de dados por meio de troca de mensagens com um conjunto de servidores através de uma rede TCP/IP. O objetivo principal do projeto é permitir o aproveitamento de recursos de armazenamento eventualmente disponíveis em uma rede (ex. os discos rígidos de estações de trabalho comuns), o que possibilita a execução de aplicações que manipulam grandes volumes de dados, e ao mesmo tempo realizar essa tarefa com alto desempenho pela combinação de paralelismo e distribuição.

O DPFS apresenta duas características inovadoras em relação aos outros sistemas de arquivos distribuídos encontrados na literatura: o método de *striping* não é estático, e pode até mesmo receber orientações por parte do usuário final, e o gerenciamento de metadados é implementado através de um SGBD como Postgres ou MySQL. A Figura 2.4 ilustra o sistema. O acesso ao sistema de arquivos por parte dos clientes ocorre, assim como no PVFS, em duas etapas: primeiro o cliente consulta o SGBD para obter os metadados, e posteriormente acessa diretamente os servidores para a leitura/escrita dos dados.

A justificativa para qualificar o DPFS como um sistema voltado a aplicações científicas vem dos chamados *níveis de arquivos* que o sistema define. Três níveis de arquivos são previstos: linear, multi-dimensional e em arranjo. A cada um corresponde um método distinto para a realização do *striping*. No método linear, a divisão do arquivo é feita de

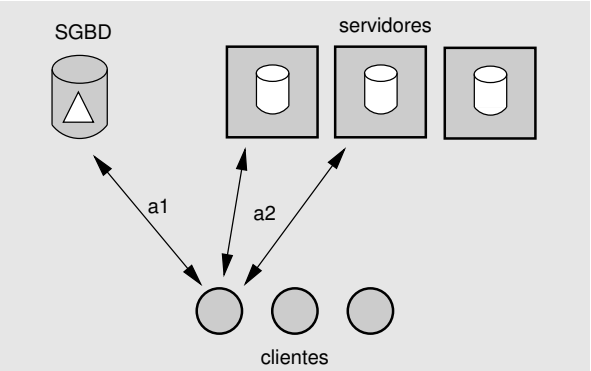

Figura 2.4: Arquitetura do DPFS

forma seqüencial entre os servidores disponíveis (*round-robin*), dividindo o arquivo em fatias de tamanho arbitrário, o que corresponde ao *striping* normalmente encontrado em outros sistemas. No método multi-dimensional, as fatias são agrupadas de modo a corresponder a padrões de acesso tipicamente encontrados em aplicações científicas, como a leitura de colunas inteiras de matrizes. Por último, o método de *striping* em arranjo é projetado especificamente para corresponder aos padrões de acesso do HPF, o que beneficia aplicações implementadas nessa linguagem.

O gerenciamento de meta-dados no DPFS não é feito através de meta-arquivos, mas sim pelo armazenamento de informações em um SGBD. O objetivo é permitir a realização dessas operações através de uma interface de alto nível e confiável, principalmente pela existência dos mecanismos de gerenciamento de transações atômicas nos SGBDs, o que fornece automaticamente a manutenção de consistência do sistema de arquivos.

Para acesso ao sistema, os clientes DPFS devem fazer uso de uma API própria, a qual define funções para a criação, escrita e leitura de dados nos arquivos. É através dessa interface que o usuário pode expressar as orientações necessárias para o mecanismo de *striping*, de modo a obter melhor desempenho dependendo do tipo de arquivo. Uma interface com comandos Unix como *ls*, *cp*, *mkdir*, dentre outros, também é disponibilizada.

2.3.5 Lustre

O Lustre (CLUSTER FILE SYSTEMS, INC., 2002) é um dos mais recentes sistemas distribuídos de arquivos, desenvolvido pela Cluster File Systems, Inc., voltado especialmente para a computação com clusters. O Lustre combina características já conhecidas de sistemas de arquivos distribuídos, como separação de dados e meta-dados, com tecnologias modernas de gerenciamento como LDAP e XML. A primeira versão pública e considerada estável foi lançada em dezembro de 2003.

O sistema de arquivos do Lustre apresenta diversas abstrações projetadas com o objetivo de melhorar o desempenho e a escalabilidade. A Figura 2.5 ilustra uma visão global do sistema. O Lustre trata arquivos como objetos, que são localizados por meio de servidores de meta-dados, ou MDS (*Metadata Servers*). Os MDS suportam operações no nível do espaço de nomeação do sistema de arquivos, como procura (*lookup*), criação e remoção de arquivos e manipulação de atributos (proprietário, grupo, data e hora de modificação, entre outros). As operações de acesso aos blocos de dados propriamente ditos são redirecionadas a *objetos de armazenamento*, ou OST (*Object Storage Targets*), os quais gerenciam o armazenamento de informações fisicamente situadas em *discos baseados em objetos*, ou OBD (*Object-Based Disks*).

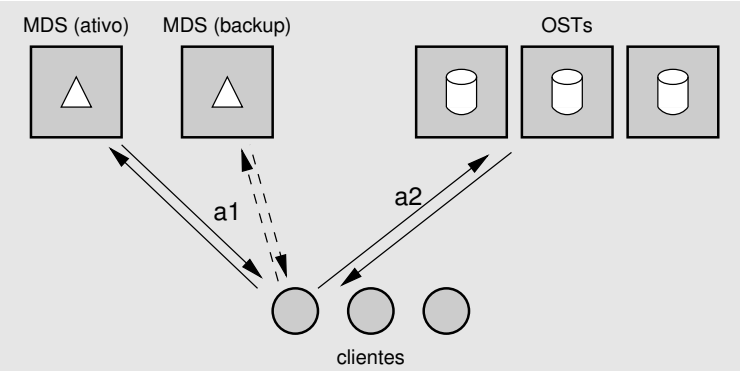

Figura 2.5: Visão global do funcionamento do Lustre

Os MDS mantêm um registro de transações sobre mudanças nos meta-dados e no status do cluster, de modo a tolerar e recuperar eventuais falhas na rede e/ou no hardware. Em operação normal, o MDS possui uma réplica que assume suas funções (modelo primário/backup) em caso de falha do servidor. Essa passagem de operação para o segundo MDS é feita de forma automática. Além do registro de transações, o Lustre faz uso de sistemas de arquivos locais com *journaling* como Ext3 e Reiserfs para garantir a consistência dos dados.

O Lustre suporta heterogeneidade de tecnologias de rede por meio de uma abstração chamada de NAL — *Network Abstraction Layer*. Essa abstração é uma camada de software que padroniza as funções de acesso à rede, atuando como intermediária entre o restante do sistema e a tecnologia utilizada. Atualmente, já existem implementações da NAL para os protocolos TCP e Quadrics. A troca de mensagens entre os diversos componentes do sistema é realizada com uma API chamada Portals.

2.4 Avaliação dos Sistemas Apresentados

Pode-se perceber uma diversidade de características nos diversos sistemas apresentados. Mesmo dentro dos dois principais grupos, cada sistema utiliza-se de uma determinada abordagem em busca do favorecimento de um ou outro aspecto do desempenho do sistema de arquivos. Nesta seção, apresenta-se uma avaliação e comparação dessas características, principalmente em termos de gerenciamento de dados e meta-dados, tolerância a falhas e adequação ao modelo Beowulf, procurando relacioná-las com o trabalho desenvolvido na tese.

2.4.1 Sistemas Baseados em Discos Compartilhados

Uma análise das características dos sistemas de arquivos deste grupo torna-se relativamente difícil de ser realizada, pelo fato de serem sistemas que exigem, em sua maioria, alguma tecnologia especial, não disponível normalmente para testes. Adicionalmente, a documentação encontrada é geralmente de cunho comercial, e não técnico. Os poucos indicativos de desempenho encontrados revelam quase sempre vazão e escalabilidade muito bons, o que pode ser realmente esperado dado o alto investimento nesses produtos. Tais informações podem, por outro lado, ser tendenciosas, pois são divulgadas normalmente pelo próprio fabricante. O que se pode comprovar, entretanto, é que são sistemas efetivamente usados em máquinas paralelas comerciais (como o GPFS, usado no ASCI White) e assim normalmente fechados (com exceção do XFS e de uma variação do GFS), de li-

cença comercial, e que portanto não se encontram disponíveis para uso pela comunidade científica.

A característica comum desses sistemas, que origina a classificação é a definição de uma área global de armazenamento, seja por um único disco, um dispositivo do tipo RAID ou mesmo por discos em servidores independentes. Os blocos de dados e as informações de meta-dados dos arquivos dos usuários encontram-se espalhados pela integridade do espaço de armazenamento, podendo ser explicitamente replicados a fim de obter-se melhor tolerância a falhas, como faz o Frangipani/Petal, ou, no caso mais geral, automaticamente replicados por dispositivos de hardware como as controladoras RAID, o que neste caso é independente de sistema.

2.4.2 Sistemas Distribuídos

Os sistemas de arquivos enquadrados neste grupo são aqueles mais próximos do objetivo deste trabalho, por isso cabe, neste ponto, uma avaliação mais detalhada de suas características. Como item principal, ressalta-se o fato de nenhum deles exigir o uso de alguma tecnologia especializada, podendo portanto ser empregados diretamente em clusters de forma geral.

A avaliação é feita pela análise individual de algumas características que podem ser observadas nos referidos sistemas, como apresentado a seguir.

2.4.2.1 Gerenciamento de Dados

Os sistemas expostos apresentam diferentes formas de lidar com o conteúdo dos arquivos, ou dados propriamente ditos. No NFS convencional e sistemas nele baseados, os dados de um arquivo são manipulados de forma integral, ou seja, não existe uma separação dos dados em forma de *striping* ou mecanismo semelhante. Isso é uma característica herdada do próprio NFS, e quando mantida favorece a simplicidade do sistema, pois não é necessário controlar a localização das fatias. Adicionalmente, observa-se que o *striping* é implementado principalmente para permitir acesso concorrente aos diversos blocos de dados de um arquivo, favorecendo o desempenho no acesso, o que não é o objetivo principal de sistemas como o D-NFS e o Bigfoot-NFS.

Outra possibilidade para o gerenciamento de dados é a efetiva utilização de *striping*, como fazem o xFS, o PVFS, o DPFS e o Lustre. No caso do xFS e do Lustre, existe ainda a possibilidade de implementação de RAID em software, de forma a garantir a integridade dos dados. A divisão do conteúdo dos arquivos em fatias, como já colocado, objetiva principalmente o melhor desempenho no acesso, pois vários processadores podem acessar blocos distintos em paralelo, desde que suportado pelas conexões de rede, e assim realizar a leitura ou escrita de dados em menos tempo. Outro aspecto importante é que essa técnica minimiza a contenção de acesso quando vários processadores precisam manipular dados do mesmo arquivo, pois cada um tem a possibilidade de trabalhar em partes distintas de cada vez. Pode-se considerar, portanto, que o *striping* de dados é uma característica fundamental para o bom desempenho de um sistema de arquivos distribuído.

2.4.2.2 Gerenciamento de Meta-Dados

De forma semelhante aos dados, o gerenciamento de meta-dados encontra abordagens diferentes entre os sistemas estudados, agrupadas em duas situações: gerenciamento centralizado e gerenciamento distribuído. No caso centralizado encontram-se a maioria dos sistemas, devido à sua maior facilidade de implementação, e também pelo fato de

que a maior parte das operações de E/S realizadas em uma aplicação paralela com essa característica tem a ver com os dados, e não com os meta-dados; portanto, o custo de um gerenciamento centralizado de meta-dados é mascarado pela ocorrência de um maior número de operações de dados.

O NFS convencional não faz distinção, em termos de processos, entre o gerenciamento de dados e de meta-dados; ambos são gerenciados pelo processo *nfsd*, que é o servidor propriamente dito. Desta forma, pode-se considerar que o gerenciamento é centralizado, uma vez que o servidor é único[3]. No caso do NFSˆ2 e do Bigfoot-NFS, essa tarefa continua exatamente igual, pois tratam-se de vários servidores NFS convencionais individuais.

Sistemas como o PVFS, o DPFS e o Lustre implementam uma entidade dedicada, no sistema, para o gerenciamento dos meta-dados. No PVFS e no Lustre essa entidade é um processo em uma das máquinas servidoras, enquanto que no DPFS é um SGBD. Em todos os casos, o servidor de meta-dados é único, existindo apenas um *backup* no caso do Lustre. Nessa abordagem, o acesso ao sistema de arquivos começa por um contato com o servidor de meta-dados, o qual informa a localização dos blocos de dados, e posteriormente os servidores de E/S são contactados diretamente. Essa técnica é de implementação simples e, como já exposto, o fato de centralizar as operações de meta-dados não chega a se tornar um gargalo para o sistema.

Nos casos do D-NFS e do xFS, assim como planejado para a versão 2 do PVFS, a gerência de meta-dados é distribuída. Isso significa que as informações sobre localização dos blocos de dados de um arquivo encontram-se possivelmente espalhadas em mais de uma máquina na rede, ou mesmo replicadas. A vantagem desta abordagem é, além da eliminação do ponto central de acesso a meta-dados, a possibilidade de tolerância a falhas se for permitida a replicação de informações. Por outro lado, a complexidade de gerenciar esse mecanismo é maior, pois é necessário garantir que nenhum cliente obtenha informação inválida, o que o leveria, por exemplo, a acessar um bloco de dados já em uso por outro arquivo. Tal complexidade de gerenciamento pode afetar o desempenho do sistema. No D-NFS, o objetivo maior não é o desempenho, portanto o controle é implementado sem essa preocupação. No xFS observa-se um mecanismo realmente complexo, mas que aparentemente permite a obtenção de bom desempenho. Infelizmente, pela descontinuidade do projeto, não se pode avaliar o sistema nos ambientes de execução atuais.

Em resumo, os sistemas existentes mostram uma tendência de centralização da gerência de meta-dados sem afetar o desempenho global do sistema. Entretanto, o xFS indica que seria possível realizar essa gerência de forma distribuída, o que favoreceria a tolerância a falhas pela possibilidade de replicação de meta-dados entre máquinas distintas.

2.4.2.3 Tolerância a Falhas

A tolerância a falhas nos sistemas relacionados ocorre em diferentes níveis. Ela pode estar restrita aos meta-dados, abranger dados e meta-dados, e ainda suportar falhas permanentes ou somente temporárias. Este é o caso do NFS e seus derivados; tradicionalmente, o NFS, por ser projetado de forma a não manter estado sobre as conexões aos clientes, pode tolerar falhas temporárias como interrupções nos cabos de rede ou períodos de ocupação intensa de CPU no servidor. No caso do NFSˆ2 e do Bigfoot-NFS, onde os clientes não acessam o servidor da maneira convencional, não fica claro, na literatura, se o mesmo

[3]Na verdade, o *nfsd* pode disparar cópias adicionais de si mesmo através de *fork()* para melhor atender as requisições dos clientes, mas todas as cópias trabalham sobre o mesmo conjunto de meta-dados, portanto pode-se ainda considerá-lo como centralizado.

nível de tolerância é suportado.

O DPFS tolera a ocorrência de falhas no gerenciamento de meta-dados de forma indireta, devido ao uso de um SGBD para essa função. Se houver uma interrupção no meio de uma operação desse tipo, o próprio mecanismo de transações atômicas de um SGBD como o Postgres ou o MySQL trata de retornar as informações de meta-dados a um estado consistente.

Nos casos do xFS e do Lustre, tanto a gerência de dados quanto a de meta-dados dispõem de mecanismos de tolerância a falhas. O xFS distribui completamente a funcionalidade do servidor de arquivos, estando as informações replicadas em RAID sobre várias máquinas. No Lustre, o armazenamento de dados também se beneficia de técnica RAID por software. Já a gerência de meta-dados conta com um servidor *backup*, de modo que, se o servidor principal falhar, aquele assume seu lugar de forma automática.

Adicionalmente a essas características, muitos dos sistemas de arquivos da atualidade, mesmo os de propósito geral, já empregam técnicas de *journalling* que permitem um certo nível de tolerância a falhas no caso, por exemplo, de interrupções de energia elétrica. Em conseqüência, muitos dos sistemas descritos, inclusive aqueles do grupo baseado em discos compartilhados, têm um benefício indireto advindo dessas técnicas.

2.4.2.4 Adequação ao Modelo Beowulf

Este é um item bastante subjetivo, mais relacionado às facilidades de gerenciamento do que ao funcionamento do sistema em si. Por outro lado, dada a grande difusão de sistemas baseados em software livre e componentes não-especializados, um sistema de arquivos que se encaixa no modelo Beowulf pode ser uma vantagem considerável.

Na verdade, o único sistema a não se encaixar no modelo Beowulf é o xFS, em função de não contar com implementações atuais (que rode em Linux, por exemplo). O restante dos sistemas se distingüe principalmente pela questão de disponibilidade do software. Poucos sistemas encontram-se em estado maduro e possuem um ciclo estável de desenvolvimento e manutenção. Dentre esses, pode-se colocar o PVFS, o Lustre e o próprio NFS. Sob esse aspecto, essas seriam as escolhas para o sistema de arquivos de um cluster da atualidade.

2.4.3 Avaliação Geral

Pode-se observar, nos sistemas apresentados, uma grande quantidade de características que favorecem a computação de alto desempenho em vários aspectos. Um quadro comparativo resumindo essas características é apresentado na Tabela 2.1.

Pode-se concluir que os sistemas mais adequados para a computação baseada em clusters são aqueles que se baseiam na distribuição das funcionalidades, principalmente pela disponibilidade do software e pela não exigência de tecnologias específicas. Em termos de técnicas empregadas, percebe-se o *striping* como uma das principais características para o alto desempenho. A tolerância a falhas é encontrada em diversos sistemas e representa maior confiabilidade no armazenamento de informações, portanto é uma característica desejável. Na comunidade, nota-se uma especial atenção para o PVFS, por ser originário de parceiros da NASA e pelo próprio desempenho reportado; o Lustre foi lançado recentemente, e também tem atraído a atenção dos grupos de pesquisa.

No capítulo seguinte, apresenta-se outro projeto baseado no NFS padrão, chamado de *NFSP*, o qual apresenta uma combinação de diversas das características recém apresentadas, e que serviu de base para o desenvolvimento da tese.

Tabela 2.1: Resumo das características dos sistemas de arquivos apresentados

Sistema	Gerência de Dados	Gerência de Meta-Dados	Tolerância a Falhas[a]	Característica Beowulf
XFS	SAN	SAN	M	plena
Frangipani/Petal	SAN	SAN	D/M	média
GFS	SAN	SAN	D	média
GPFS	SAN	SAN	D/M	não
NFS	integral	centralizada	temporária	plena
NFSˆ2	integral	centralizada	?	média
D-NFS	integral	distribuída	temporária	média
Bigfoot-NFS	integral	centralizada	?	média
xFS	RAID	distribuída	D/M	não
PVFS	*striping*	centralizada	não	plena
DPFS	*striping*	centralizada	M	média
Lustre	RAID	centralizada	D/M	plena

[a]D = dados; M = meta-dados

3 NFSP: NFS PARALELO

No capítulo anterior foram apresentados diversos dos sistemas de arquivos atuais voltados ao processamento paralelo. Dentre os sistemas expostos, pode-se perceber um subconjunto que tem como característica comum a utilização do protocolo padrão NFS como base de funcionamento. A motivação para essa característica é justificada de várias formas, sendo a principal o fato de o NFS ser um protocolo bem estabelecido, com implementações bastante refinadas e conseqüentemente com pouca tendência a instabilidades.

Nessa mesma linha de pensamento, surgiu no ano de 2000 o projeto NFSP — NFS Paralelo — do *Laboratoire Informatique et Distribution* (ID) de Grenoble, França, como parte de um projeto de implantação de um cluster de grande porte. Tendo motivado desde então diversos trabalhos científicos de alunos e estagiários do ID, o NFSP é base para a tese aqui apresentada, e portanto se faz necessária uma exposição de suas principais características, conforme a seguir.

3.1 O Projeto NFSP

O ID, no ano de 2000, tendo sido recém criado e com diversos projetos novos em implantação, fechou uma parceria com a HP para a construção de um cluster de grande porte, chamado de *i-cluster*. Em sua instalação final, o i-cluster apresentava 225 nós de processamento, sendo cada nó dotado de um processador Pentium III 733 MHz, com 256 MB de memória RAM, disco rígido de 15 GB e equipado com placa de rede Fast Ethernet 10/100 Mb. Diversos novos desafios para os grupos de pesquisa do ID se apresentaram com a implantação do i-cluster, dentre eles a escolha de um sistema de arquivos adequado para o porte da máquina, visto que se tornou evidente, desde a concepção do projeto, que o uso de um servidor NFS tradicional, com problemas inerentes de escalabilidade, seria inviável.

A equipe coordenada pelo Prof. Yves Denneulin iniciou assim uma fase de avaliação dos diversos sistemas de arquivos paralelos disponíveis na época. Uma série de critérios foi estabelecida para a escolha do sistema a ser usado no i-cluster, dentre elas a exigência de que o mesmo introduzisse o mínimo de modificações em uma instalação convencional de um cluster (visto que ferramentas de gerenciamento de clusters eram também um dos tópicos de pesquisa do ID), além de dever ser um sistema aberto e disponível em plataforma Linux.

Uma primeira possibilidade foi considerada sobre o AFS, porém esta opção foi descartada pelo fato de a granularidade de compartilhamento no AFS ser muito grossa (nível de arquivo e não de bloco de dados), além da existência de mecanismos de segurança que impunham perdas consideráveis de desempenho e que se mostravam desnecessários em um ambiente isolado e dedicado ao processamento de aplicações paralelas como o de um

cluster. O grupo então voltou-se para o PVFS, o qual foi instalado e submetido a uma série de testes de desempenho. Os resultados obtidos foram desanimadores; ainda no início de seu desenvolvimento, o PVFS apresentava comportamento bastante instável, e trazia o inconveniente, segundo os critérios adotados, de exigir modificações no kernel dos nós de processamento.

Assim, não tendo encontrado uma solução existente que contemplasse todas características desejadas para o i-cluster, o grupo decidiu iniciar uma nova implementação. Avaliando os critérios estabelecidos, concluiu-se que o NFS representava uma escolha equilibrada entre todas as características desejadas, e assim surgiu o NFSP. O modelo e protótipo iniciais foram resultado da tese de Doutorado de Pierre Lombard (2003), defendida em dezembro de 2003.

3.2 Estrutura do NFSP

O modelo estrutural do NFSP é inspirado no PVFS. O objetivo principal é paralelizar ao máximo as operações de acesso a blocos de dados, porém sem exigir demasiadas modificações nos procedimentos convencionais de instalação e funcionamento do protocolo. Basicamente, duas possibilidades são oferecidas: alterar o código dos clientes ou alterar o código do servidor. Como em geral o número de clientes é maior do que o de servidores, a escolha da segunda opção resulta em menor número de modificações, e por isso essa foi a solução adotada. Em conseqüência, a primeira característica importante do NFSP é o fato de os clientes utilizarem o código padrão de um cliente NFS regular (ou seja, o código disponível no kernel Linux original). Em outras palavras, os nós clientes de um cluster usando o NFSP não precisam ser alterados em nenhum aspecto. Essa característica, desejável desde o início do projeto, abre espaço para duas importantes constatações: ferramentas de gerenciamento já existentes podem ser utilizadas normalmente, e o cluster permanece compatível com soluções comerciais apresentadas por empresas como a MandrakeSoft e a Red Hat, o que favorece instalações de cunho mais profissional onde se deseja contar com suporte técnico por parte do distribuidor do software.

O NFSP, portanto, obtém ganho em desempenho pela paralelização das funções do servidor NFS. Este último é dividido em duas entidades: o *meta-servidor* e os *servidores de E/S*, ou, como usualmente chamado, *IOD* (do inglês *I/O daemon*). A Figura 3.1 ilustra a interação entre clientes, meta-servidor e IODs.

3.2.1 IODs

Como freqüentemente encontrado em sistemas de arquivos paralelos, o NFSP faz o uso de IODs como processos independentes, executados em diversos nós do cluster, como mecanismo de armazenamento de dados. O uso dessa técnica é bastante difundido, pois apresenta uma série de vantagens: o espaço em disco de vários nós pode ser aproveitado, o mecanismo facilita a introdução de técnicas de replicação de dados para tolerância a falhas, e, principalmente, diversos IODs operando em paralelo aumentam significativamente a vazão de dados lidos e/ou escritos pelo sistema de arquivos.

No NFSP, assim como em outros sistemas como o xFS e o PVFS, o armazenamento do conteúdo de arquivos nos IODs é feito com a técnica de *striping*, ou seja, o arquivo é dividido em blocos menores (e portanto "listrado") e cada bloco é armazenado em um IOD distinto, continuando de forma circular se o número de blocos ultrapassa o número de IODs disponíveis. O tamanho dos blocos pode ser variável ou fixo, dependendo do sistema. No NFSP, são utilizados blocos de tamanho fixo correspondentes ao tamanho

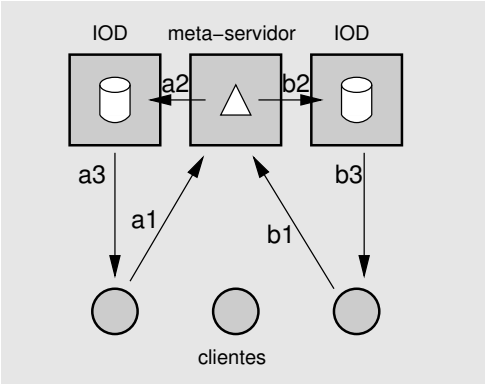

Figura 3.1: Interações entre clientes, meta-servidor e IODs no NFSP. A requisição de um cliente (passo 1) é enviada ao meta-servidor, que a repassa ao IOD correspondente (passo 2), o qual realiza a operação desejada e envia os resultados (passo 3); “a” e “b” são requisições distintas

dos blocos usados no protocolo NFS (geralmente 8192 bytes).

3.2.2 Meta-servidor

O meta-servidor é um *daemon*, executado em apenas um dos nós, que aparece para os clientes como o servidor NFS convencional. Isso significa que as operações de montagem e acesso ao sistema de arquivos, por parte dos clientes, são direcionadas ao meta-servidor, como se fosse o servidor NFS de uma instalação usual. Essa característica reforça, mais uma vez, o fato de um cliente NFSP não necessitar de qualquer modificação em seu funcionamento.

Assim como no NFS, o meta-servidor exporta uma ou mais hierarquias de diretórios (geralmente especificadas no arquivo `/etc/exports`), as quais são mantidas no sistema de arquivos local do servidor. À diferença do NFS, entretanto, essa hierarquia não contém os próprios arquivos exportados, e sim um conjunto de *meta-arquivos*, onde cada meta-arquivo contém informações sobre a localização dos dados. Se fosse feita uma listagem da hierarquia local de arquivos exportados em um servidor NFSP, seria constatado que a listagem é idêntica à de um servidor NFS normal, com exceção dos tamanhos dos arquivos: enquanto que no caso do NFS normal os arquivos apresentariam seu tamanho real, no NFSP cada arquivo teria somente alguns poucos bytes, suficientes para determinar a localização dos dados. Note-se que diretórios são tratados de forma idêntica nos dois sistemas.

O ponto-chave do NFSP é o tratamento de requisições dos clientes. Ao receber uma requisição, por exemplo de leitura de bloco de dados, o meta-servidor consulta o meta-arquivo correspondente (tomando as devidas providências iniciais em relação à existência do arquivo, permissões, etc.) e a repassa ao IOD correspondente. Este último, por sua vez, obtém o bloco desejado a partir do disco (ou de sua cache, se o bloco já foi lido anteriormente), e o transmite ao cliente *como se fosse o servidor*. No caso em que diversas requisições semelhantes são emitidas pelos clientes ao mesmo tempo, obtém-se ganho em desempenho pelo funcionamento em paralelo dos diversos IODs.

3.3 Descrição das Implementações

Ao longo de três anos de atividades, alguns protótipos do NFSP foram implementados por alunos do ID, e diversos testes foram realizados, com a finalidade de investigar alternativas quanto às possibilidades práticas do modelo proposto. Apresenta-se a seguir uma breve descrição das implementações realizadas e níveis de desempenho obtidos.

3.3.1 uNFSP

O primeiro protótipo do NFSP foi implementado com base na versão em nível de usuário (*user-space*) do NFS, mais precisamente a versão 2.2beta47. A escolha dessa versão é fundamentada no fato de que permitia um ciclo rápido de modificações e testes (se comparada à versão de kernel, a ser apresentada na seqüência), e também porque a versão 3 do servidor apresentou problemas de instabilidade. Em conseqüência, a versão é chamada de uNFSP (de *user-space*).

As alterações no código do NFS original consistiram, basicamente, na interceptação de requisições advindas dos clientes e, quando necessário, tratamento de acordo com a funcionalidade desejada para o NFSP. Algumas funções como a criação e remoção de diretórios, conforme exposto anteriormente, podem ser realizadas diretamente no sistema de arquivos local do meta-servidor; as demais, entretanto, devem ser encaminhadas aos respectivos IODs. Naturalmente, também foi necessária a implementação *from-scratch* desses últimos.

O armazenamento dos blocos de dados nos IODs é realizado utilizando o sistema de arquivos local dos próprios. Para fornecer correta localização e ao mesmo tempo minimizar a possibilidade de conflito entre blocos de dados distintos, o nome de cada arquivo que contém um bloco é dado pela combinação do *inode* do meta-arquivo correspondente, no meta-servidor, mais um número gerado aleatoriamente. Assim, um bloco de dados em um IOD qualquer pode ser encontrado sob um nome do tipo `i00000042_s00001234_o00789000`. O número aleatório é armazenado no meta-arquivo para permitir a localização posterior do bloco no IOD correspondente.

Outro aspecto importante da implementação é a necessidade de uso de *UDP spoofing*, técnica de alteração do endereço de origem de um pacote IP. Essa técnica é necessária quando da resposta dos IODs aos clientes, pois estes esperam receber a resposta vinda do servidor e não de outra máquina.

Uma medida preliminar de desempenho do uNFSP foi realizada no i-cluster, configurando o sistema com 16 nós IOD e fazendo os nós clientes (que montam a partição NFSP) ler, em paralelo, um arquivo previamente criado com tamanho de 1 GB. O tempo de execução considerado para o experimento é tomado desde que a aplicação é lançada sobre o cluster até seu encerramento completo; embora esse tempo inclua um pequeno *overhead* devido às fases de disparo e finalização dos processos sobre os nós, a duração do experimento é suficientemente longa (cerca de 90 s no caso mais rápido) para mascará-lo. O desempenho é avaliado sob forma de vazão agregada, ou seja, a vazão de dados que os nós conseguem obter em leitura de forma conjunta, dada formalmente pela fórmula

$$V = \frac{nK}{t}$$

onde V é a vazão agregada, em MB/s, n é o número de clientes, K é o tamanho do arquivo lido, em MB (no caso, 1024 MB ou 1 GB) e t é o tempo de execução, em segundos.

Como forma de comparação, as medidas de desempenho apresentam também uma curva de desempenho “ideal” esperado para um determinado número de clientes. Esse

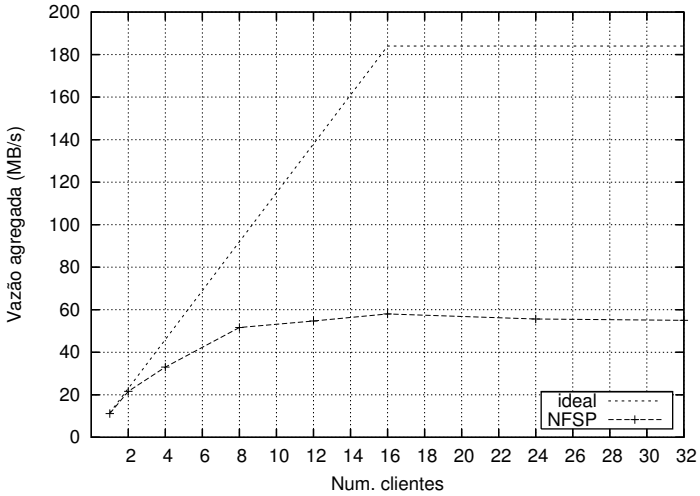

Figura 3.2: Vazão agregada obtida com o uNFSP no i-cluster

desempenho ideal é determinado pela vazão máxima das conexões de rede dos nós envolvidos no experimento (outro fator limitante seria o desempenho dos discos rígidos, porém os discos do i-cluster apresentam vazão superior à da conexão de rede, tornando-se este último fator o limitante principal). A curva de vazão ideal é dada pela fórmula

$$V = \begin{cases} n_C R & \text{se } n_C \leq n_I \\ n_I R & \text{caso contrário} \end{cases}$$

onde V é a vazão agregada, em MB/s, n_C é o número de clientes, n_I é o número de IODs e R é a vazão máxima da rede, em MB/s. A distinção é necessária uma vez que, enquanto o número de clientes é inferior ao número de IODs, não há conexões suficientes para saída de dados, no lado dos clientes, para completar a vazão total permitida pelos IODs. No caso deste experimento, n_I é 16, como já mencionado, e R é fixado em 11,5. Esse valor foi determinado através de um experimento de ping-pong, e obtido como vazão máxima, na prática, da conexão de rede de um nó do i-cluster (o valor máximo teórico, em se tratando de uma placa Fast Ethernet, é de 100 Mb/s ou 12,5 MB/s).

Assim, o gráfico na Figura 3.2 mostra os valores obtidos com o uNFSP, em comparação com o desempenho ideal. Pode-se perceber de imediato que o desempenho alcançado pelo protótipo situa-se muito abaixo do ideal. Após investigação sobre o protótipo, concluiu-se que a implementação em nível de usuário sofre de perdas significativas em função de constantes comunicações entre os processos que implementam o servidor e o núcleo do sistema operacional, principalmente devido a cópias de blocos de dados em memória. O efeito visível é uma carga de praticamente 100% de utilização de CPU no nó onde o meta-servidor é executado quando do uso de um número elevado de clientes.

Em função desses resultados, o grupo decidiu portar a implementação para a versão do servidor NFS disponível no kernel Linux, descrita a seguir.

3.3.2 kNFSP

O porte do NFSP sobre a versão de kernel do NFS foi acompanhado de diversas pequenas extensões ao sistema, já com o objetivo de suportar um primeiro nível de tolerância a falhas por replicação. Assim, uma das mudanças realizadas, além da própria adaptação sobre o NFS de kernel, foi a introdução do conceito de *VIOD*, ou IOD virtual. A idéia é que o meta-servidor comunica-se com o VIOD, que na prática é associado a

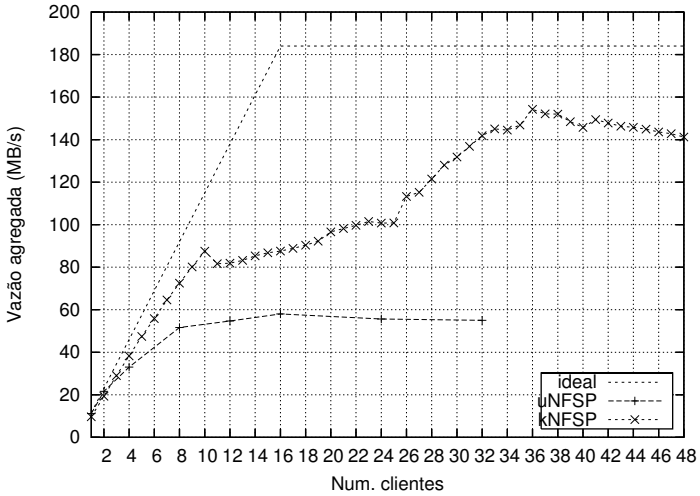

Figura 3.3: Vazão agregada obtida com o kNFSP

um grupo de IODs reais que se revezam na função de armazenar blocos de dados. Embora o grupo ainda não tenha efetivamente implementado uma política de replicação, essa técnica já permite um nível inicial de balanceamento de carga e de utilização de disco entre os IODs. Foi incluída também a possibilidade de mesclar a exportação de partições em modo NFSP ou em modo NFS normal, bastando acrescentar ou não a palavra-chave "nfsp" à linha de configuração no `/etc/exports`. O trabalho foi desenvolvido por Olivier Valentin durante seu período de estágio no ID em junho de 2002.

A implementação em si não apresenta grandes diferenças em relação ao uNFSP, à exceção do mecanismo de VIOD. Uma extensão à chamada `nfsctl()` foi introduzida para permitir a configuração do servidor com as mesmas ferramentas do pacote `nfs-utils`, que também teve que ser alterado. Em sua versão final, o kNFSP consiste portanto de dois grupos de *patches*, um para o kernel Linux e outro para o pacote `nfs-utils`.

Os mesmos testes de desempenho aplicados anteriormente foram repetidos para esta versão do NFSP, com a única diferença de ter sido usado um número maior de nós clientes. Os resultados são mostrados na Figura 3.3, comparados com a versão uNFSP e com o desempenho ideal. O ganho em desempenho em relação à versão anterior é visível, confirmando-se a hipótese de menor *overhead* quando o tratamento é feito inteiramente em espaço de kernel. A irregularidade da curva a partir de um número maior de clientes, conforme averiguado, deve-se ao fato de ocorrerem *timeouts* nos clientes, devido à grande quantidade de requisições enviadas ao meta-servidor, o que diminuiu o desempenho mas efetivamente evita que aquele seja saturado plenamente.

3.3.3 Execução com Múltiplos Servidores

Uma outra tentativa de obtenção de melhor desempenho com os protótipos do NFSP foi a implantação de um esquema com vários meta-servidores, onde cada um exporta uma árvore local de meta-arquivos e re-exporta árvores de outros meta-servidores, sendo estas montadas por NFS normal. A Figura 3.4 ilustra a situação.

O objetivo principal deste esquema, implementado por Adrien Lebre (atualmente doutorando no ID) como parte de seu DEA (LEBRE, 2002), é prover diversos pontos de acesso aos clientes, de modo que um mesmo meta-servidor não seja sobrecarregado por todas as requisições. Cada meta-servidor armazena somente os seus próprios meta-arquivos, mas no caso de um cliente solicitar um meta-arquivo que "pertença" a outro

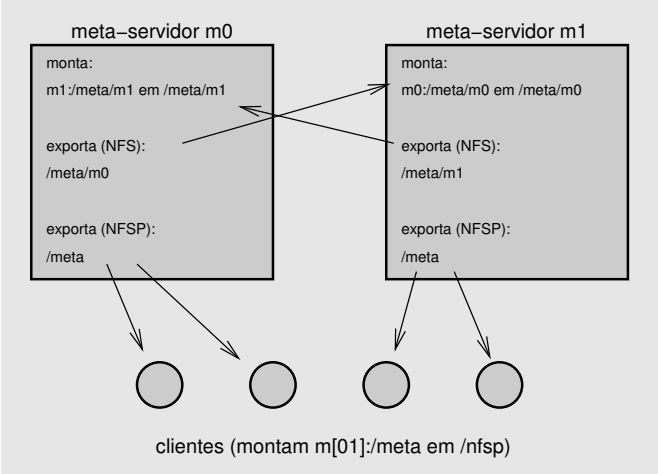

Figura 3.4: Esquema de múltiplos meta-servidores com montagens e exportações mescladas de NFS e NFSP

meta-servidor, o acesso pode ser feito pela montagem NFS.

Esse teste foi realizado com a versão do NFSP em nível de usuário, pois a versão de kernel não permite a exportação de duas hierarquias distintas quando uma está contida na outra, como é o caso ilustrado (diretórios `/meta` e `/meta/m0`, por exemplo). Mesmo sendo utilizada a versão com maior sobrecarga, os resultados confirmaram a expectativa de melhor desempenho. Com a utilização de 4 meta-servidores, foi possível atingir o dobro de vazão em operações de leitura, quando comparada ao protótipo uNFSP original.

Apesar de efetiva em termos de desempenho, a desvantagem desta abordagem é refletida em termos de espaço de nomeação: nesse esquema, cada cliente enxerga duas hierarquias de diretórios, `/meta/m0` e `/meta/m1`, o que acarreta em perda de transparência de nomeação dos arquivos, passando a hierarquia a ser dependente dos servidores m0 e m1.

3.4 Atividades Atuais e Futuras

Este capítulo apresentou uma breve síntese do projeto NFSP, como forma de embasamento para o trabalho desenvolvido na tese, tendo sido expostas as principais características do sistema, bem como as atividades realizadas no projeto até o momento. Além de novos testes e refinamentos das implementações já feitas, a equipe do Prof. Denneulin desenvolve ainda outras atividades:

- Um dos critérios iniciais de projeto do NFSP, além da intrusividade mínima, consiste no armazenamento de dados de forma distribuída. Este é um dos fatores determinantes da escolha do mecanismo de IODs, além dos próprios benefícios inerentes do método. O objetivo do projeto, com essa característica, é permitir não só o acesso paralelo a blocos de dados, mas também a implementação de um mecanismo de troca direta dessas informações *entre* clusters distintos, possivelmente em um ambiente de WAN. Um primeiro passo nesse sentido foi realizado durante o ano de 2003, com a implementação da API GXfer, realizada por Christian Guinet. Essa API define funções para troca de dados diretamente entre dois conjuntos de IODs distintos, de modo que um cluster possa ter acesso ao sistema de arquivos de outro de forma direta e eficiente, sendo os acessos potencialmente paralelizáveis.

- O doutorando Olivier Valentin inicia suas pesquisas no chamado NFSG, um sistema de arquivos baseado nos mesmos princípios do NFSP porém orientado a grids, ou clusters de clusters. A idéia é prover um espaço de nomeação global sobre um conjunto de clusters que utilizem o NFSP, possivelmente fazendo uso da interface GXfer para troca de dados de forma otimizada, e também aplicando conceitos de consistência mais relaxada como é feito no AFS.

- Analisando-se o modelo de distribuição empregado no NFSP, percebe-se que a paralelização dos acessos pelos IODs é realizada de forma efetiva no caso das operações de leitura, uma vez que os dados são enviados diretamente aos clientes, porém o mesmo não ocorre nas operações de escrita. Quando um ou mais clientes realizam a escrita de blocos de dados, todas as informações devem ser necessariamente enviadas ao meta-servidor, pois ele é a única entidade conhecida dos clientes. Tendo o meta-servidor uma única conexão de rede, como é o caso normal, retorna-se ao problema de escalabilidade do NFS original. Com o objetivo de investigar e solucionar esse problema, foi proposta a tese aqui apresentada. O detalhamento da solução proposta e conseqüente validação são apresentados nos capítulos seguintes.

4 O MODELO DE DISTRIBUIÇÃO PROPOSTO: DNFSP

O capítulo anterior apresentou as principais características do projeto NFSP, sobre o qual o trabalho é baseado. Conforme mostrado, o NFSP consegue efetivamente prover um nível diferenciado de desempenho ao sistema de arquivos de um cluster, mas tal benefício, pela arquitetura do sistema, restringe-se às operações de leitura de dados, ficando as operações de escrita enquadradas no mesmo modelo centralizado do NFS original. Enquanto que muitas aplicações cuja maioria das operações é de leitura podem já tirar proveito desse primeiro modelo do NFSP, é também desejável que o sistema permita a realização de escrita de dados de forma otimizada. É com esse objetivo que se desenvolve esta tese, cuja contribuição principal é apresentada neste capítulo.

Com o estudo dos sistemas apresentados no Capítulo 2, diversas técnicas de otimização do desempenho de sistemas de arquivos puderam ser analisadas. Após um período de investigação realizado durante o ano de 2003 junto ao Laboratoire ID, período esse que correspondeu à estadia em Grenoble devido à orientação em co-tutela, concluiu-se que seria possível apresentar uma extensão ao NFSP para permitir a distribuição também das operações de escrita, resultando no modelo apresentado nas seções seguintes. Cabe ressaltar desde já que as modificações introduzidas foram planejadas de forma a manter os objetivos originais do NFSP, principalmente em relação à não introdução de processos demasiadamente intrusivos que implicassem em modificações significativas na instalação convencional de um cluster.

Dentre os principais objetivos da contribuição proposta, podem ser ressaltados:

- apresentar uma extensão do modelo NFSP que permita melhor desempenho e escalabilidade de aplicações paralelas com maior tendência à realização de operações de escrita;

- permitir a obtenção de um nível mais elevado de desempenho em um cluster do tipo Beowulf, sem implicar no uso de tecnologias de maior custo; em outras palavras, permitir que aplicações que não dependem explicitamente do desempenho do sistema de arquivos possam ser beneficiadas pelo uso de um sistema compatível com o NFS padrão, mesmo sem o uso de tecnologias especiais;

- sendo o NFS um protocolo amplamente difundido, investigar uma possibilidade de melhor desempenho desse sistema de modo a beneficar também ambientes, além de clusters, onde seu uso é desejável;

- atingir um equilíbrio adequado entre desempenho e facilidade de uso/integração com instalações já existentes; não se está almejando atingir o melhor desempenho possível para o sistema de arquivos de um cluster, e sim um aumento significativo

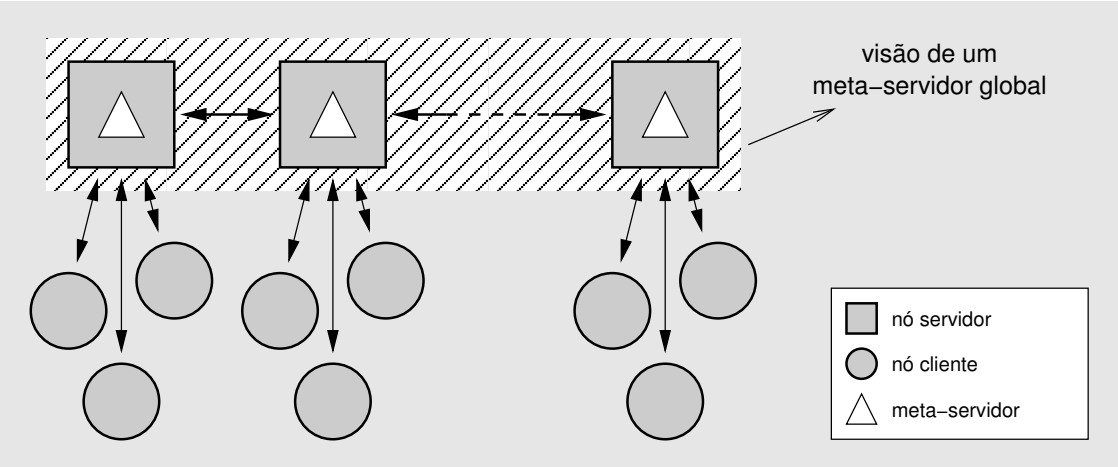

Figura 4.1: Arquitetura proposta para a distribuição do meta-servidor

de desempenho que ainda permita o uso de procedimentos tradicionais de uso e administração do cluster;

- não se deseja provocar uma situação de competição com outros sistemas de arquivos, principalmente pelo fato de que as diversas abordagens existentes podem ser melhor adaptadas a um ou outro tipo de aplicação. Por exemplo, pode-se citar a interface MPI-IO, provavelmente mais adequada do que o NFS para aplicações que precisam de um nível mais fino de paralelismo na manipulação de dados intra-arquivos; em contra-partida, seu uso requer conhecimento da interface específica de programação, ao passo que o NFS é utilizado por meio da interface padrão (POSIX) de acesso ao sistema de arquivos.

4.1 Visão Geral da Solução Proposta

Objetivando principalmente o aumento do desempenho e da escalabilidade nas operações de escrita de dados, a base do modelo proposto é a distribuição do meta-servidor original do NFSP. A Figura 4.1 ilustra a arquitetura planejada. A disponibilização de várias instâncias do meta-servidor permite que os clientes formem grupos independentes, de modo que cada grupo enxerga uma única instância como o servidor NFS. Uma vez que o tamanho de cada grupo é menor que o número total de clientes, a demanda total de vazão sobre cada instância do meta-servidor é menor, reduzindo a carga sobre o mesmo e, em conseqüência, possibilitando um melhor desempenho.

A seguir é apresentado o detalhamento do modelo de extensão ao NFSP proposto.

4.2 Otimização das Operações de Escrita no NFSP: *dNFSP*

Como apresentado no capítulo anterior, o NFSP transforma o servidor NFS tradicional em um meta-servidor, cujas requisições de acesso a blocos de dados são repassadas a servidores dedicados a esse fim. Embora a leitura de dados seja feita por esses servidores, repassando as informações lidas diretamente para os clientes, as operações de escrita ainda devem ser encaminhadas ao meta-servidor, centralizando a transferência de dados e, em conseqüência, tornando-se um gargalo para esse tipo de operação.

A solução mais imediata para esse problema, assim, é disponibilizar diversos pontos

de acesso para os clientes, de modo que haja maior largura de banda para a transferência dos dados; em outras palavras, isso significa disponibilizar várias instâncias dos meta-servidores, formando a noção de um meta-servidor global.

Um dos testes realizados sobre os protótipos do NFSP, como apresentado anteriormente, produz um mecanismo semelhante, apenas com uma indireção, ou *network hop*, a mais (no caso, um servidor NFSP deve repassar a escrita ao servidor NFS associado). Para comprovar a eficácia desse modelo, avaliando ser esta a alternativa que produziria melhores resultados, foi realizado um novo teste sobre o protótipo original, mas desta vez simplesmente disparando processos adicionais (em nós distintos) do meta-servidor, os quais acessavam o mesmo conjunto de IODs. Uma vez que a distribuição não era prevista na implementação original, foi necessário garantir que os diversos meta-servidores não trabalhassem sobre os mesmos arquivos. Assim, para a execução dos testes, cada cliente trabalha com um arquivo individual (nomeado com base no *hostname* do nó cliente).

O experimento foi realizado no i-cluster, sendo configuradas 8 máquinas como IODs, 8 máquinas com meta-servidores e 8 clientes. A execução consistiu em cada cliente criar um arquivo próprio de tamanho 1 GB.

O ganho em desempenho nas operações de escrita pode ser imediatamente percebido, proporcionando uma vazão total bem acima do possível com um único servidor (teoricamente, 12,5 MB/s, que é a capacidade da rede Ethernet 100 Mb/s instalada no cluster), aproximando-se de 70 MB/s.

Analisando os resultados obtidos com este experimento, concluiu-se que a distribuição dos meta-servidores é suficiente para atingir os objetivos propostos para o trabalho em termos de desempenho e filosofia do NFSP. Esse, portanto, é o modelo adotado no trabalho. Para distinção em relação do NFSP original, adota-se a denominação *dNFSP*, de *distributed* NFSP, para o modelo proposto.

Em conseqüência da distribuição dos meta-servidores no dNFSP, abre-se a possibilidade de conflito de dados quando dois ou mais clientes conectados a meta-servidores distintos têm a necessidade de manipular o mesmo conjunto de arquivos. Esse problema, em efeito a manutenção da consistência de meta-dados entre as instâncias, é tratado a seguir.

4.3 Manutenção da Coerência entre os Meta-Servidores

Com a simples distribuição dos meta-servidores no dNFSP, um problema surge de imediato em uma situação bem simples: se um cliente cria um determinado arquivo, o mesmo não será visível a outro cliente que esteja associado a um meta-servidor diferente. Isso ocorre porque os meta-arquivos são elementos presentes no sistema de arquivos local dos meta-servidores, e portanto não visíveis remotamente.

No teste apresentado no capítulo anterior, envolvendo a combinação de servidores NFS e NFSP como forma de distribuição, o mesmo problema não aparece, pois os meta-arquivos são exportados e acessados pelos meta-servidores justamente pela montagem NFS. Em compensação, surge o problema da perda de transparência no espaço de nomeação, como já exposto. Adicionalmente, o mecanismo de consistência fraca típico do NFS pode não ser suficiente para garantir que o caso recém exposto de criação de um arquivo por um cliente qualquer seja sempre imediatamente visível aos demais clientes. Assim, um mecanismo de manutenção de consistência de meta-arquivos deve ser introduzido.

A solução proposta para esse problema no dNFSP foi buscada na literatura de sistemas de memória compartilhada distribuída (DSM). Mais especificamente, a técnica a

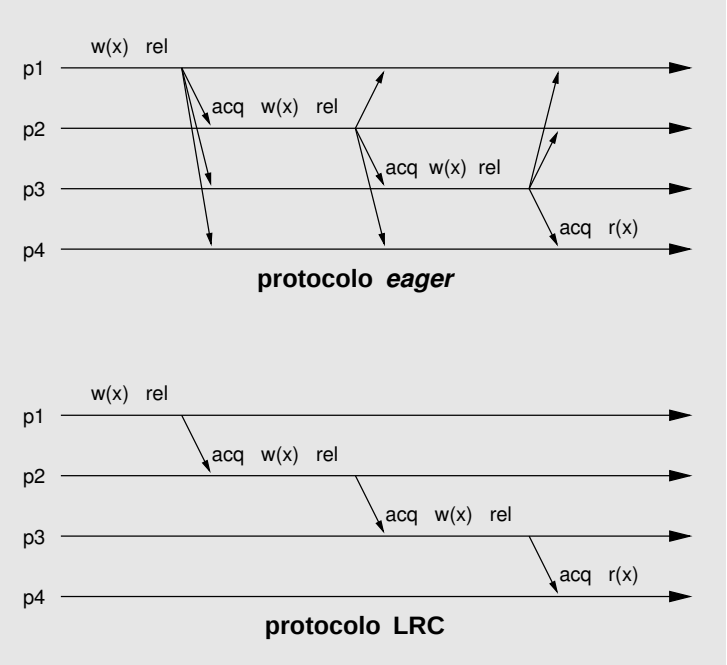

Figura 4.2: Envio de mensagens de atualização de segmentos compartilhados utilizando protocolos *eager* e LRC

ser utilizada é inspirada no mecanismo de *Lazy Release Consistency* (KELEHER; COX; ZWAENEPOEL, 1992), ou LRC, usado no sistema TreadMarks (AMZA et al., 1996).

4.3.1 LRC: *Lazy Release Consistency*

O método de *Lazy Release Consistency* é um mecanismo de consistência de dados proposto por Keleher et al. (1992) para sistemas de memória compartilhada distribuída (ou DSM, *Distributed Shared Memory*), sendo usado, por exemplo, no TreadMarks. O LRC resulta da observação, em sistemas DSM, de que nem sempre é necessário propagar as modificações efetuadas em um segmento de memória compartilhada a todos os processos, e sim somente aos que vão utilizá-lo a curto prazo.

A Figura 4.2 ilustra a troca de mensagens entre processos em um sistema DSM, na primeira situação utilizando um protocolo *eager* (“ansioso”), e na segunda situação utilizando LRC. O segmento compartilhado, *x*, é criado pelo processo *p1*, o qual altera seu conteúdo, e em seguida é acessado pelos processos *p2*, *p3* e *p4*, nesta ordem. A cada acesso, um processo deve sinalizar a obtenção do segmento (*acq*), realizar a operação desejada (*r* ou *w*, leitura ou escrita), e finalmente sinalizar a liberação do segmento (*rel*).

No protocolo *eager*, a liberação de um segmento automaticamente dispara o envio das alterações efetuadas a todos os processos que compartilham o segmento. Desta forma, tendo sido executada pelos três primeiros processos, essa operação gera o envio de $3 * 3 = 9$ mensagens de atualização na rede. No caso do LRC, as atualizações não são enviadas explicitamente pelo processo que terminou de acessar o segmento, e sim são *solicitadas* pelo processo que vai começar a utilizá-lo. Como observado na figura, a mesma seqüência de acessos, no LRC, ocasiona o envio de somente 3 mensagens pela rede.

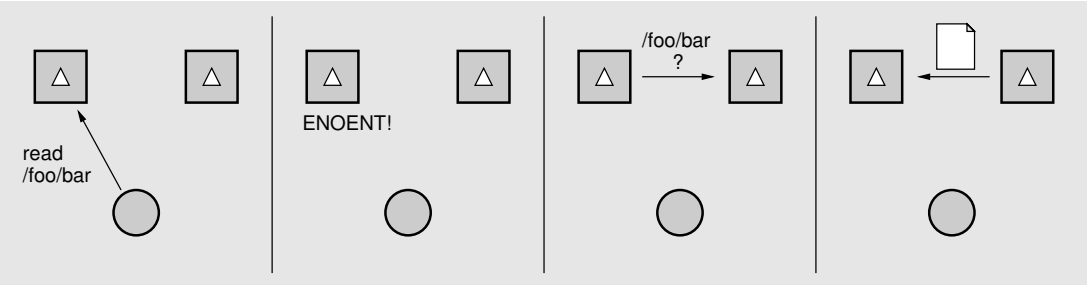

Figura 4.3: Etapas do mecanismo de atualização de meta-arquivos

4.3.2 Adaptação do LRC ao dNFSP

De forma semelhante ao que é feito no sistema TreadMarks, onde a consistência é verificada somente quando um cliente sinaliza a obtenção de um segmento compartilhado (visto que outros clientes podem tê-lo modificado), a proposta para o dNFSP é verificar a consistência dos meta-dados *somente quando do acesso de um cliente a um determinado arquivo*. Em outras palavras, o sistema posterga a difusão de atualizações nos meta-dados até o momento em que realmente sejam necessárias por motivo de acesso de algum cliente.

A noção de um estado consistente no dNFSP é relativa, ou seja, se refere somente ao contexto do arquivo sendo acessado. A ação é iniciada pelo meta-servidor cujos meta-arquivos estão desatualizados. É ele quem solicita aos demais meta-servidores o envio das informações atualizadas. Esse mecanismo evita o envio de mensagens de atualização desnecessárias que poderiam ocupar as conexões de rede.

Adicionalmente, essa verificação de consistência pode não ser sempre necessária: o mecanismo que dispara uma atualização só é ativado quando ocorre algum tipo de erro no acesso ao sistema de arquivos. Portanto, nas operações mais freqüentes, na quais a leitura ou escrita de dados ocorre sem erros, a verificação de consistência não é necessária. A Figura 4.3 ilustra os passos decorrentes de uma atualização.

Exemplificando a situação, pode-se imaginar que um determinado cliente iniciou uma operação de leitura de um bloco de dados em um arquivo. A operação é enviada ao meta-servidor correspondente para que o acesso seja realizado. No momento de obter as informações do meta-arquivo que corresponde ao arquivo desejado, porém, o meta-servidor obtém o código de erro `ENOENT`, que significa “arquivo inexistente”. Esse erro pode ser um indício de que o arquivo foi criado por algum outro cliente, associado a um meta-servidor diferente. Nesse caso, aciona-se o mecanismo de atualização de meta-arquivos.

Em outra situação, suponha-se que um determinado arquivo foi criado pelo cliente A, associado ao meta-servidor X, com tamanho inicial de 10 kbytes. Esse, portanto, é o tamanho do arquivo como conhecido pelo cliente A. Em um segundo momento, um cliente B, associado a outro meta-servidor, Y, necessita trabalhar sobre o arquivo e finaliza sua operação por aumentar o tamanho daquele para 50 kbytes. Posteriormente, o cliente A volta a acessar o arquivo, lendo os primeiros 4 kbytes de dados. Neste caso, apesar de o cliente B ter modificado o meta-arquivo correspondente (no meta-servidor Y) atualizando o tamanho do arquivo, a informação disponível no meta-servidor X é suficiente para que o acesso possa ser feito, pois os dados desejados encontram-se ainda na parte comum às duas “versões” do arquivo. É importante salientar que, mesmo que o meta-arquivo não esteja atualizado, os dados efetivamente lidos pelo cliente são armazenados nos IODs, e

portanto foram devidamente atualizados quando do acesso pelo cliente B.

Três situações podem ser identificadas como causadoras de atualizações entre os meta-servidores:

- *quando o meta-arquivo não existe*, o que corresponde à situação apresentada no primeiro exemplo. Nesse caso, para determinar de qual meta-servidor buscar o meta-arquivo, é necessário o estabelecimento de uma política de busca. Diversas alternativas são possíveis, as quais são apresentadas posteriormente.

- *quando o meta-arquivo está desatualizado*; no segundo exemplo dado, um cliente A pode ler com sucesso os blocos de dados iniciais de um arquivo que teve seu tamanho aumentado, conquanto que a parte desejada já estivesse presente na versão conhecida por ele (no exemplo, o tamanho original do arquivo era de 10 kbytes, e o cliente desejava ler os primeiros 4 kbytes). O contrário, porém, não é possível. Se o cliente desejasse ler um bloco de dados além dos 10 kbytes originais, a operação falharia. Nesse caso, o mecanismo de atualização de meta-dados é igualmente ativado.

- *quando a operação requer uma atualização explícita*. Alguns tipos de operações, como remoção e truncamento (redução no tamanho) de arquivos requerem uma atualização explícita, nesse caso enviada a todos os demais meta-servidores. Isso é necessário porque um cliente pode, nesse caso, chegar a ler informações de um arquivo que não mais existe, e portanto estaria lendo dados inválidos. Cabe salientar que esse tipo de operação não é comum em aplicações paralelas, e portanto o impacto causado pelo atraso de uma atualização de todos os meta-servidores é minimizado, dentro de um contexto mais abrangente.

4.4 Políticas para Atualização de Meta-Arquivos

Como exposto anteriormente, cada vez que é necessária uma atualização de meta-arquivo, cabe ao meta-servidor em questão contactar outros meta-servidores e solicitar a informação desejada. Nesse momento, um problema a ser resolvido é qual meta-servidor contactar. Idealmente, dever-se-ia contactar diretamente aquele que possui o meta-arquivo desejado; como essa informação não está disponível, a solução é estabelecer uma política de busca que estabeleça uma ordem na qual os vários meta-servidores serão contactados. Naturalmente, essa política deve ter por objetivo encontrar o meta-servidor correto o mais cedo possível.

Algumas possíveis políticas de busca de meta-arquivos para o dNFSP são apresentadas a seguir.

4.4.1 Busca por Vizinhança

Esta política consiste em estabelecer uma relação de vizinhança entre os meta-servidores, de modo que se possa identificar o quão próximo um está de outro. Essa relação pode ser feita numerando os meta-servidores, por exemplo de 0 a $n-1$, supondo que existam n meta-servidores. A patir daí, diz-se que dois meta-servidores estão mais próximos quanto menor for a diferença entre os números que os identificam. A contagem é circular, de modo que o meta-servidor 0 está à mesma distância dos meta-servidores 1 e $n-1$.

Estabelecida esta regra, define-se a política de modo que a procura inicia dos meta-servidores *mais próximos para os mais distantes*. Por exemplo, suponha-se um conjunto

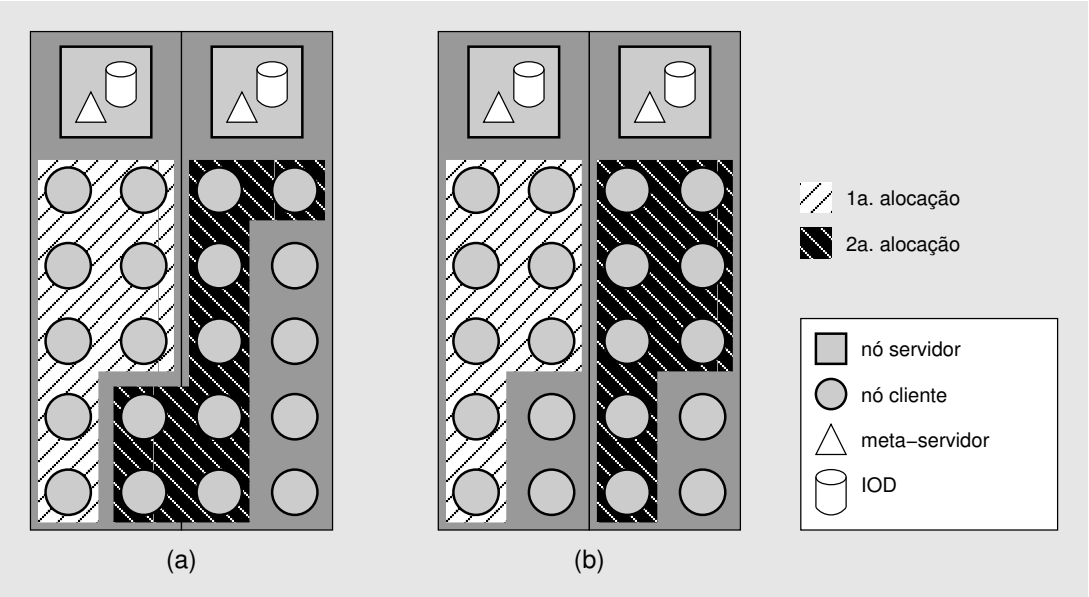

Figura 4.4: Duas situações ilustrando a alocação de duas partições segundo a política de busca por vizinhança: (a) alocação ruim; (b) alocação mais adequada

de 8 meta-servidores, numerados de 0 a 7, e que o meta-servidor 1 precise fazer uma atualização. Ele inicia a procura solicitando a informação ao servidor 0, depois (ou simultaneamente) ao servidor 2. Se não obtiver sucesso, a busca continua com o envio de solicitações aos servidores 7 e 3, em seguida aos servidores 6 e 4, e assim por diante.

Esta política é adequada para casos onde a conexão de rede dos servidores é de maior capacidade, permitindo o agrupamento de vários clientes por meta-servidor, e especialmente se o mecanismo de alocação de nós aos usuários for dotado de certa inteligência. Em resumo, esse mecanismo deve procurar agrupar nós de uma mesma aplicação sob o menor número possível de meta-servidores, de modo a minimizar a ocorrência de atualizações de meta-arquivos. A Figura 4.4 exemplifica duas situações onde duas alocações de 8 nós são feitas, sendo que a situação *a* representa uma alocação ruim e a situação *b* uma alocação mais adequada neste contexto.

4.4.2 Busca Hierárquica

No caso de instalações com um grande número de meta-servidores, pode ser interessante a atualização de meta-arquivos de forma hierárquica. Nesta política, os meta-servidores são organizados em forma de árvore. Quando da necessidade de busca de um determinado meta-arquivo, um meta-servidor faz a solicitação ao seu nodo “pai”; este, se já não estiver de posse do arquivo em questão, repassa a solicitação ao outro ramo que lhe está associado, e, caso o arquivo ainda não seja encontrado, passa por sua vez a solicitação ao seu nodo pai. O nodo raiz é responsável por enviar a mensagem final de erro caso o meta-arquivo não seja encontrado em nenhum dos seus ramos.

Esta pode ser uma alternativa adequada também para aplicações que apresentem grupos de nós clientes trabalhando sobre um mesmo conjunto de arquivos.

4.4.3 Busca Aleatória

Uma outra alternativa para a busca de meta-arquivos entre os meta-servidores é realizar uma ordenação aleatória entre os mesmos. Se a ferramenta de alocação não utiliza

qualquer critério especial quanto ao agrupamento de nós, após algum tempo de uso o conjunto de nós disponíveis pode estar completamente espalhado pelo cluster, não sendo possível estabelecer uma previsão de associação entre clientes e meta-servidores. Nesse caso, mesmo uma política simples como escolher um meta-servidor aleatoriamente pode ser eficaz.

4.5 Considerações sobre o Modelo Proposto

Como já mencionado, o modelo de distribuição proposto para o dNFSP visa proporcionar um aumento de desempenho na operação do NFSP, particularmente sobre as operações de escrita, sem abdicar das linhas gerais que guiam o projeto. Com o estabelecimento da distribuição do meta-servidor e manutenção de consistência por meio de um mecanismo relaxado, baseado em LRC, chegou-se a uma proposta que contempla esses objetivos.

Deve-se ressaltar, entretanto, que o modelo é direcionado para ambientes de computação de alto desempenho, tendo-se abdicado, em seu projeto, de características próprias de um ambiente de compartilhamento de arquivos, função para a qual o NFS é normalmente empregado. Desta forma, recursos como ligações rígidas (*hard links*), tradicionalmente existentes em sistemas de arquivos de ambientes Unix, e arquivos de dispositivos (normalmente encontrados na hierarquia `/dev`), não são tratados pelo modelo. Entende-se que tais recursos, embora imprescindíveis na operação de máquinas Unix e ambientes de rede em geral, não são vitais no sistema de arquivos utilizado pelos usuários de um cluster, e os benefícios obtidos em termos de desempenho e escalabilidade têm maior relevância para as aplicações paralelas.

Outra característica alterada em relação ao NFS normal é uma coerência mais relaxada em termos de meta-dados, como noções de dono/grupo de um arquivo, datas de acesso/modificação, entre outros. Tal característica é devida ao mecanismo de atualização de meta-arquivos, o qual permite acesso aos blocos de dados mesmo que os meta-dados não estejam completamente atualizados, em prol de menor *overhead* no envio de mensagens pela rede. É importante salientar, entretanto, que o próprio NFS não provê um mecanismo forte de coerência (baseado em temporização), portanto os usuários já esperam não poder contar com esse recurso de forma totalmente confiável.

De modo a validar o modelo proposto, foram conduzidos uma série de experimentos envolvendo *benchmarks* e aplicações reais como forma de medir a correção e a eficiência do sistema. Esses experimentos, bem como os resultados obtidos e uma análise dos mesmos, são apresentados no capítulo a seguir.

5 VALIDAÇÃO DO DNFSP: IMPLEMENTAÇÃO E MEDIDAS EXPERIMENTAIS

Para validar os diversos aspectos de funcionalidade e desempenho do modelo de distribuição proposto no dNFSP, um protótipo do mesmo foi implementado e submetido a uma série de experimentos visando uma avaliação do comportamento do modelo na prática. Variando desde medidas primárias de desempenho como leitura e escrita seqüencial até a execução de uma aplicação real, esses experimentos demonstram a viabilidade do dNFSP como um sistema de arquivos para clusters, conforme apresentado a seguir.

5.1 Implementação sobre o uNFSP

O ponto de partida para a implementação de um protótipo do dNFSP foi a decisão sobre a forma de colocar o modelo proposto em prática. A forma mais direta seria partir de uma das implementações já existentes do NFSP; outra possibilidade seria realizar uma nova implementação com caráter maior de simulação do protocolo NFS. Decidiu-se, por fim, basear o protótipo do dNFSP sobre a implementação *user-level* do NFSP, ou uNFSP. Mesmo que a versão *kernel-level* propiciasse melhor desempenho, conforme mostrado anteriormente, a escolha deveu-se a uma série de razões:

- a implementação sobre o uNFSP beneficia-se da facilidade de manutenção de um programa no nível de usuário, como por exemplo a depuração e a flexibilidade de ativação/desativação dos processos. O NFS permite mesmo que o servidor seja disparado por um usuário comum, sem direitos administrativos, bastando fornecer as diretivas adequadas no disparo dos processos. Essa característica é útil em sistemas com restrições administrativas;

- algumas das funcionalidades do NFS v2 ainda não foram implementadas na versão de kernel, como por exemplo o suporte à re-exportação de diretórios remotos. A disponibilidade de tais recursos poderia ser importante no desenvolvimento e avaliação do dNFSP;

- o fato de trabalhar no nível de usuário permite um ciclo de teste/depuração mais eficiente pois, para testar uma nova versão, basta recompilar o sistema e relançar os processos necessários; a versão baseada em kernel implicaria, necessariamente, a cada teste, na reinicialização de todas as máquinas envolvidas, a fim de que o novo kernel fosse utilizado

- por fim, é de interesse do grupo avaliar os potenciais de aumento de desempenho das implementações já existentes; o uNFSP, por diversas razões, apresenta desempenho

a)

```
0
192.168.0.1
192.168.0.2
192.168.0.3
192.168.0.4
```

b)

```
2
192.168.0.1
192.168.0.2
192.168.0.3
192.168.0.4
```

Figura 5.1: Exemplo de arquivos `dms.conf`; em um total de 4 meta-servidores, o arquivo *a)* corresponde ao primeiro meta-servidor, de IP 192.168.0.1, e o arquivo *b)* ao terceiro, cujo IP é 192.168.0.3

inferior ao kNFSP, e portanto uma implementação no sentido de estendê-lo poderia trazer novos benefícios.

5.1.1 Identificação dos Meta-Servidores

O primeiro passo na adaptação do NFSP para o modelo proposto é a identificação dos vários meta-servidores. Essa é uma característica nova, não presente na implementação original, já que o meta-servidor era único. Para a atribuição de um identificador para cada instância do meta-servidor, optou-se, de forma semelhante à existência do arquivo `iods.conf` no NFSP, pela criação de um novo arquivo de configuração para o sistema, `dms.conf`, a ser instalado, por padrão, no diretório `/etc`.

O arquivo `dms.conf` deve ser criado antes do disparo do meta-servidor, e instalado no diretório correto em cada uma das máquinas onde é executada uma instância. A sintaxe da configuração é bastante simples, sendo lida linha a linha:

- a primeira linha deve conter somente um número (ex. “1”), que identifica o meta-servidor local;
- a partir da segunda linha, são listados os endereços IP de todas as máquinas que compõem o meta-servidor, com um IP por linha, na ordem correspondente à numeração dada.

Cabe notar que, como a primeira linha identifica o meta-servidor local, o conteúdo do arquivos deve ser diferente para cada instância. A Figura 5.1 ilustra exemplos para duas máquinas distintas.

O `dms.conf` é lido quando da inicialização do meta-servidor, mais precisamente do processo *nfspd*. Cada instância guarda sua própria identificação e monta uma tabela correspondendo os demais IDs com os IPs listados no arquivo. Esses endereços são usados, posteriormente, quando da necessidade de comunicação entre os meta-servidores.

5.1.2 Comunicação entre os Meta-Servidores

A principal característica do modelo de distribuição do dNFSP é a troca de informações entre os meta-servidores, em especial o envio e recebimento de meta-arquivos. Portanto, o primeiro passo na construção do protótipo foi implementar uma maneira de realizar essa comunicação.

A primeira forma implementada consistia na criação de conexões TCP entre todos os meta-servidores, de modo que cada um tivesse comunicação direta com todos os ou-

tros. Essas conexões seriam usadas para a troca de qualquer informação necessária. O mecanismo foi implementado e alguns testes iniciais foram realizados com sucesso.

Essa implementação, entretanto, apresentava algumas desvantagens. Por exemplo, quando da existência de um número grande de meta-servidores, o tempo necessário para realizar todas as conexões ($n!/2(n-2)!$, onde n é o número de meta-servidores) se torna considerável, chegando à ordem de vários segundos. Além disso, um tal mecanismo de comunicação exige que cada meta-servidor faça uma verificação periódica (*polling*) em suas conexões para poder receber mensagens dos outros servidores, o que acarreta em maior custo computacional. Assim, procurou-se outra forma de comunicação.

A solução encontrada tira proveito dos mecanismos de comunicação já existentes no cluster. Em qualquer cluster, é necessária a existência de um mecanismo básico de comunicação entre os nós de processamento, a fim de que processos possam ser disparados em nós remotos. É dessa forma, por exemplo, que uma aplicação é inicialmente disparada a partir de um nó frontal (*front-end*). Esse mecanismo é normalmente implementado pelo serviço de shell remoto, ou *rsh*. Juntamente com o rsh encontra-se o mecanismo usado no dNFSP para comunicação entre os meta-servidores: o *rcp* (*remote copy*, ou cópia remota).

O *rcp* permite, em suma, que se copie um arquivo de/para um nó remoto, empregando a mesma sintaxe de um comando *cp* comum do Unix, bastando que se inclua, em um dos dois parâmetros obrigatórios, o nome da máquina remota. No dNFSP, o *rcp* é, portanto, utilizado para copiar um meta-arquivo de um meta-servidor para outro.

A iniciativa da cópia parte do meta-servidor que ainda não possui o meta-arquivo, encontrando-se, portanto, em uma situação onde um cliente tentou acessar o arquivo correspondente e o mesmo não existia. De acordo com a política empregada de busca de meta-arquivos, o meta-servidor local decide em qual meta-servidor remoto procurar a informação desejada, e efetua a execução de um *rcp* desse servidor para o sistema de arquivos local. Se a operação é realizada com sucesso, significa que o meta-arquivo foi encontrado, e o acesso originado pelo cliente tem sua execução prosseguida. Caso contrário, se a execução do *rcp* retorna um código de erro, significa que o meta-arquivo não existe no referido meta-servidor. Novamente de acordo com a política de busca, um novo meta-servidor é escolhido, e a operação se repete. Se todas as execuções falharem, é porque o arquivo realmente não existe.

A escolha do mecanismo de *rcp* para busca de meta-arquivos remotos tem como principais justificativas:

- é um mecanismo disponível em praticamente qualquer instalação de cluster (a não ser quando o serviço é substituído pelo *ssh*, o qual, entretanto, disponibiliza o comando equivalente, *scp*), e que apresenta a funcionalidade necessária para realizar essa tarefa;

- sua integração no código do dNFSP é bastante simples de ser realizada, bastando incluir uma chamada `system()` com o comando e os parâmetros necessários; essa chamada permite inclusive a verificação do código de erro retornado pela execução do comando;

- o uso do *rcp* apresenta um efeito colateral favorável: a cópia remota de um meta-arquivo traz, além dos meta-dados armazenados internamente, a manutenção de vários meta-dados importantes como proprietário, grupo e permissões do meta-arquivo. Esses meta-dados são herdados implicitamente pela própria semântica de execução do *rcp* que, assim como o *cp* comum, preserva a maioria dos meta-dados quando da cópia de um arquivo.

Uma preocupação com a utilização do *rcp* é o seu custo em termos de tempo de execução, pois trata-se do disparo de um novo processo na máquina remota, o qual deve ser carregado do disco, inicializado e colocado na fila de escalonamento. Dependendo de fatores como a quantidade de memória da máquina em que roda o meta-servidor e a própria política de escalonamento do SO empregado, o tempo de carga e execução do *rcp* pode ser considerável. Por outro lado, após sua primeira execução, existe uma boa chance de que o seu código seja mantido em memória, eliminando uma grande parcela do tempo de execução referente à carga do disco. Essa preocupação, portanto, foi considerada desde o início da implementação e posteriormente avaliada na prática, conforme mostrado na seção dos resultados.

O ponto da implementação do NFSP alterado para incluir as chamadas de *rcp* necessárias é o módulo `nfs_dispatch()`, o qual implementa, entre outras, a função de `LOOKUP`. Essa função é responsável por verificar a existência de um arquivo em um dado diretório e retornar informações sobre o mesmo. No caso da funcionalidade desejada para o dNFSP, o LOOKUP é o ponto onde se pode detectar que um arquivo não existe e, quando necessário, efetuar sua cópia remota. Neste ponto, assim, é incluído o código que implementa a política de busca e que efetua os *rcps*. Duas políticas foram implementadas:

- *linear*, basicamente usada para a validação do dNFSP; essa política sempre inicia a procura por meta-arquivos no primeiro meta-servidor (#0), passando a seguir para o segundo (#1), depois para o terceiro (#2), e assim por diante até encontrar o meta-arquivo ou chegar ao último meta-servidor (#*n*). Com essa política pôde-se, como será apresentado na seqüência, verificar condições de pior caso (ou seja, maior distância possível entre os meta-servidores);

- *por vizinhança*, que é a política descrita no capítulo anterior, na qual se inicia a procura pelos meta-servidores mais próximos e prossegue-se para os meta-servidores mais distantes até a informação ser encontrada. Essa foi a política usada na execução dos *benchmarks* e das aplicações.

5.1.3 Adaptação sobre Identificação dos Blocos

Uma alteração adicional foi necessária no NFSP de modo a garantir o acesso correto aos blocos de dados armazenados nos IODs.

Cada IOD possui um conjunto de sub-diretórios onde os blocos de dados são armazenados, sob forma de arquivos locais. O nome desses arquivos é dado por uma combinação do *i-node* do meta-arquivo correspondente, no meta-servidor, mais um número gerado aleatoriamente. Essa combinação é feita para minimizar as chances de que dois arquivos diferentes gerem um bloco de dados com o mesmo nome.

Com a distribuição do meta-servidor em várias instâncias, o mesmo mecanismo não pode mais ser adotado, isso porque agora cada cópia do mesmo meta-arquivo terá, muito provavelmente, um número diferente de *i-node* em cada meta-servidor, e portanto não corresponderá sempre ao bloco correto de dados.

A solução encontrada para a implementação do protótipo foi simplesmente substituir o número do *i-node* por um valor constante, usado por todos os meta-servidores. Assim, as chances de duplicidade de blocos de dados no conjunto de arquivos manipulado ficam por conta da geração dos números aleatórios unicamente.

Certamente esta não é uma situação aceitável para um sistema final, visto que a chance de "colisão" nos nomes dos blocos de dados seria muito grande. Para este protótipo, no entanto, a solução adotada foi suficiente para a realização dos experimentos. Uma solução

definitiva poderia, por exemplo, voltar à abordagem inicial de utilizar o *i-node*, porém incluindo-o no conteúdo do meta-arquivo, de modo que possa ser lido pelos demais meta-servidores. Essa solução, em conseqüência, demanda alterações substanciais no código do NFSP, e por isso não foi realizada.

5.2 Medidas Experimentais

Nesta seção são apresentadas as medidas de desempenho e avaliação de funcionalidade do protótipo implementado. São expostos aqui experimentos realizados com o objetivo de verificar características do dNFSP, desde as medidas mais básicas de desempenho até o comportamento do sistema em situações reais.

5.2.1 Medidas Primárias de Desempenho: discos e rede

Antes de realizar medidas de desempenho com o sistema de arquivos, é importante obter informações sobre o comportamento de dois componentes que afetam diretamente o funcionamento de um sistema de arquivos distribuído: o desempenho dos discos rígidos e da rede.

O principal ponto a ser determinado nesta medida é a capacidade de transferência de dados, ou vazão, de cada um desses componentes. Essa medida revela qual o desempenho máximo a ser esperado do sistema de arquivos, pois o uso dos discos e da rede é parte essencial do funcionamento do sistema; não se pode, por exemplo, obter determinada vazão de escrita de dados localmente em um disco rígido e, posteriormente, esperar vazão maior em uma operação remota.

Outro aspecto importante a determinar é qual dos dois componentes será o maior responsável pela limitação do desempenho alcançável; essa informação pode ser fundamental para uma correta configuração do número de clientes e servidores.

Os testes foram realizados em dois ambientes distintos. O primeiro é o i-cluster, a máquina de 225 processadores já mencionada anteriormente, disponibilizada em Grenoble, que foi retirada de funcionamento no início de 2004. Em conseqüência, os demais testes foram conduzidos no cluster LabTeC, também já citado, o qual conta com 40 processadores. Outra opção para os testes realizados foi a máquina que substituiu o i-cluster, o *i-cluster2*, composta de 104 nós Itanium2 bi-processados, perfazendo um total de 208 processadores. Esta máquina, entretanto, não permite a execução do NFSP e suas extensões, pois ocasiona erros durante a utilização do sistema de arquivos. O problema está sendo investigado, e parece estar relacionado à arquitetura de 64 bits apresentada pelos processadores da máquina. Seu uso não foi possível, portanto, para a execução destes experimentos.

As medidas de desempenho da rede são realizadas através de uma pequena aplicação normalmente chamada de *ping-pong* (DILLON; SANTOS; GUYARD, 1995; HIPPER; TAVANGARIAN, 1997; BAKER, 2000). O programa é disparado utilizando-se apenas dois nós do cluster e, ao longo de sua execução, um dos nós envia ao outro uma mensagem de tamanho pré-determinado e aguarda que ela seja enviada de volta. O tempo necessário para realizar essa operação é chamado de *round-trip time*, e é comumente aceito como o dobro do tempo gasto no envio uni-direcional de uma mensagem do referido tamanho. Essa operação é repetida diversas vezes, a fim de se obter um valor médio estável, e também com diversos tamanhos de mensagem, variando desde 0 bytes, quando se pode medir a latência mínima de envio de uma mensagem, até um tamanho grande o suficiente para saturar a conexão de rede (ex. 8 MB), caso em que se pode medir a vazão máxima. É

Tabela 5.1: Medidas de vazão máxima de transferência de dados pela rede e no acesso aos discos nos clusters i-cluster e LabTeC; os valores são expressos em MB/s (megabytes por segundo)

		Discos	
Cluster	**Rede**	**Leitura**	**Escrita**
i-cluster	11,2	42,72	35,60
LabTeC	11,1	43,22	40,63

esta última medida que interessa ao experimento realizado, pois ela indica a taxa máxima de dados que poderão ser trocados entre clientes, meta-servidores e IODs.

O desempenho dos discos foi medido utilizando-se o comando Unix *dd*, que permite a transferência de blocos de dados de tamanho arbitrário entre arquivos do sistema. As vazões máximas de escrita e leitura de dados em disco são medidas, respectivamente, através da temporização da execução de operações de criação de um arquivo de 1 GB e posterior leitura do mesmo arquivo. Mais precisamente, a vazão de escrita é obtida pela medida do tempo de execução do comando `dd if=/dev/zero of=teste bs=1M count=1024 && sync`, e a vazão de leitura pelo comando `dd if=teste of=/dev/null bs=1M count=1024`. Os dispositivos especiais `/dev/zero` e `/dev/null` são usados como fonte e destinação de dados com capacidade infinita, respectivamente. O comando *sync* na operação de escrita força a efetiva gravação dos dados na mídia magnética do disco, e é necessário para evitar mascaramento do tempo de execução pela possibilidade de manutenção de dados em buffers do sistema. Igualmente neste caso, para se obter um valor médio estável, cada medida é repetida no mínimo 5 vezes.

A Tabela 5.1 apresenta os valores obtidos nas medidas de desempenho efetuadas nos dois clusters. Os números são expressos em termos de vazão, em unidades de megabytes por segundo (MB/s). O desempenho da rede nos dois clusters é praticamente igual, em torno de 11 MB/s, pois ambos empregam tecnologia Ethernet 100 Mb/s. Em relação ao desempenho dos discos, pode-se notar uma pequena diferença em favor do cluster LabTeC, devido ao uso de discos SCSI, enquanto que o i-cluster emprega discos IDE.

Em razão dessas medidas, constatou-se como limite máximo para o desempenho por nó de transferência de dados o valor de 11 MB/s, sendo esta a limitação imposta pelas conexões de rede.

5.2.2 Leitura e Escrita Seqüenciais

O primeiro teste realizado com o protótipo do dNFSP consistiu de uma avaliação de seu desempenho bruto, ou seja, capacidade máxima de vazão agregada quando os clientes realizam operações de leitura e de escrita (ÁVILA et al., 2004). Trata-se de um experimento semelhante ao descrito no capítulo anterior, porém variando o número de clientes e contemplando também operações de leitura.

O teste é chamado de “seqüencial”, apesar de sua natureza distribuída, porque os clientes realizam a leitura ou a escrita de um arquivo de forma contínua, do primeiro ao último byte, em ordem. Esse tipo de operação pode ser encontrado em aplicações paralelas que necessitem, por exemplo, ler um conjunto de dados armazenados em disco, ou então realizar a tarefa contrária, após realizar uma série de cálculos.

A execução foi igualmente realizada no i-cluster, com um conjunto de 12 nós IOD, 7 meta-servidores e 21 clientes, perfazendo um total de 40 nós. Dois experimentos foram realizados. No primeiro caso, de forma semelhante ao teste realizado anteriormente, cada

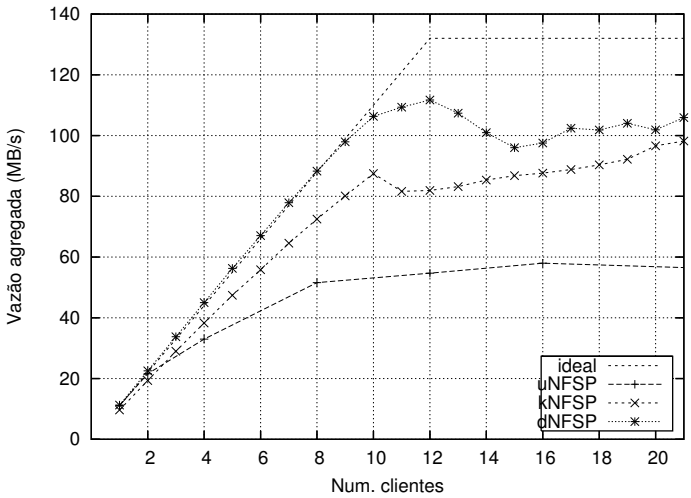

Figura 5.2: Vazão agregada obtida para leitura seqüencial no dNFSP

cliente cria um arquivo próprio de tamanho 1 GB, com o comando `dd if=/dev/zero of=arq$i bs=1M count=1024`, onde `$i` é substituído pelo nome da máquina. No segundo, cada cliente lê o seu próprio arquivo, executando o comando `dd if=arq$i of=/dev/null bs=1M count=1024`.

Os valores medidos correspondem à vazão agregada de escrita/leitura dos arquivos armazenados no servidor, que corresponde à fórmula

$$V = \frac{n_c K}{t}$$

já apresentada no Capítulo 3, onde V é a vazão agregada, em MB/s, n_c é o número de clientes, K é o tamanho do arquivo escrito ou lido, em MB, e t é o tempo de execução, em segundos.

A Figura 5.2 apresenta os resultados obtidos com o teste de leitura seqüencial. A curva obtida é comparada com a curva de desempenho ideal, que segue o mesmo comportamento mostrado anteriormente nos testes do NFSP: a vazão agregada aumenta à medida que se aumenta o número de clientes, até o ponto onde este iguala o número de IODs, quando então o desempenho máximo possível é atingido e se estabiliza. Formalmente,

$$V_l = \begin{cases} n_c R & \text{se } n_c \leq n_i \\ n_i R & \text{caso contrário} \end{cases}$$

O desempenho obtido é também comparado àquele do NFSP, em suas duas versões, a fim de demonstrar a parcela de ganho obtida pela distribuição do meta-servidor.

A curva obtida apresenta ganho evidente de desempenho, inclusive superando os valores obtidos pelas implementações prévias do NFSP. O gráfico corresponde ao comportamento da curva ideal, porém com valores mais atenuados, e apresenta um certo grau de instabilidade a partir do ponto onde $n_c = n_i$.

Um ganho em desempenho em relação ao NFSP nas operações de leitura pode parecer estranho, à primeira vista, pois o dNFSP é concebido para otimizar as operações de escrita, mantendo o comportamento original no caso da leitura. A execução deste experimento revelou, entretanto, que o fato de existirem vários meta-servidores atuando em paralelo ajuda o sistema a melhor gerenciar as diversas requisições concorrentes originadas dos clientes, pois atenua o custo computacional anteriormente imposto sobre um só servidor.

Pode-se observar que o desempenho do sistema evolui de forma praticamente linear até o número de 10 clientes, com valores bastante próximos da curva ideal. A instabilidade apresentada a partir de 12 clientes pode ser considerada dentro do esperado; uma possibilidade para esse comportamento é também o custo computacional elevado nas máquinas que executam os IODs.

No caso das operações de escrita, o comportamento da curva de desempenho segue outro modelo. Isso porque, neste caso, não somente os IODs mas também os meta-servidores estão envolvidos na tarefa de transferir a maior parte dos dados; portanto, o número de meta-servidores disponibilizados também influencia no cálculo. Se esse número for muito pequeno, pode passar a limitar o desempenho global do sistema da mesma forma que o número de IODs.

O experimento foi realizado, assim como no caso da leitura, de modo que o maior número possível de meta-servidores seja utilizado. Assim, nos casos de execuções com até 7 clientes, cada um está associado a um meta-servidor distinto; a partir de 8 clientes, os meta-servidores passam a ter que receber dados de mais de um cliente ao mesmo tempo, o que provoca um degrau na curva de vazão. Tomando como exemplo o caso específico de 8 clientes, pode-se notar que o primeiro meta-servidor terá dois clientes a ele associados, enquanto todos os demais meta-servidores terão apenas um; o fato de estar associado a dois clientes fará com que o primeiro meta-servidor receba o dobro da quantidade de dados do que os demais, praticamente dobrando também o tempo de execução do experimento, apesar de ter sido aumentado somente um cliente. A mesma situação ocorre quando o número de clientes aumenta de 14 para 15, causando os degraus na curva de desempenho.

Matematicamente, sendo n_m o número de meta-servidores, tem-se que o tempo total de escrita de dados t_e é dado por

$$t_e = \frac{1}{11}(((n_c - 1) \text{ div } n_m) + 1)$$

ou seja, o tempo necessário para a transferência de um arquivo por parte de um cliente (1 GB / 11 MB/s) multiplicado pelo maior número de clientes que um único meta-servidor gerencia. O cálculo da vazão ideal de escrita, portanto, é dado por

$$V_e = \frac{n_c 1G}{t_e}$$

ou

$$V_e = \frac{11 n_c}{((n_c - 1) \text{ div } n_m) + 1}$$

O gráfico com o resultado desse experimento é mostrado na Figura 5.3, novamente comparado com a curva ideal. O desempenho teórico do modelo NFSP original (na verdade, o do próprio NFS) é também mostrado para fins de comparação. Nota-se o comportamento peculiar em degraus, conforme recém descrito, que corresponde aos resultados obtidos na prática, porém com forte atenuação. Pode-se perceber que a vazão total atingida com o meta-servidor distribuído é consideravelmente maior do que no caso do meta-servidor centralizado, aproximando-se da curva ideal com o aumento do número de clientes, o que preenche de forma mais constante as conexões de rede. Esse resultado confirma, em última análise, a eficácia do modelo proposto em conferir um nível maior de desempenho às operações de escrita do NFSP.

A atenuação em relação à curva ideal é resultante, em parte, pela simplificação feita na modelagem matemática, que considera que os IODs são sempre capazes de suportar a

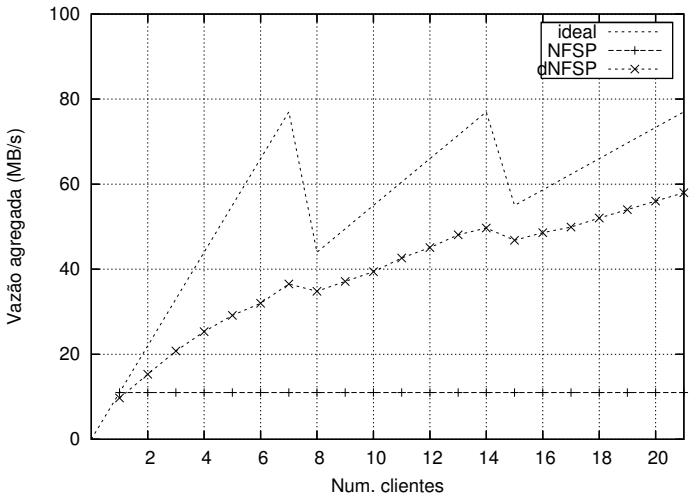

Figura 5.3: Vazão agregada obtida para escrita seqüencial no dNFSP

carga gerada pelos meta-servidores (o fato, por exemplo, de dois meta-servidores estarem acessando um bloco de dados no mesmo IOD provocaria uma perda de desempenho), e também pela carga na CPU das máquinas que hospedam os meta-servidores, uma vez que o tráfego de dados e conseqüente quantidade de trabalho desempenhado pelos servidores é bem maior que no caso da leitura.

5.2.3 *Overhead* de Busca de Meta-Arquivos

O segundo aspecto do modelo dNFSP a ser avaliado é o *overhead* causado pelo procedimento de busca remota de meta-arquivos, ou seja, o tempo gasto com a execução do *rcp* que copia um meta-arquivo remoto para o meta-servidor local. Esse tempo não pode ser elevado, sob pena de influenciar significativamente no desempenho da aplicação, principalmente no caso de haver intercâmbio de manipulação de arquivos por parte dos nós da aplicação.

Como já colocado, a decisão sobre como realizar a busca por meta-arquivos depende de uma política implementada no meta-servidor. Neste protótipo, conforme exposto anteriormente, foram implementadas duas políticas, *linear* e *por vizinhança*, sendo a primeira particularmente útil neste experimento.

O teste consiste de um programa que inicialmente cria 100 arquivos arbitrariamente em um dos nós clientes, e em seguida faz com que os demais clientes leiam os referidos arquivos. De modo a medir somente o tempo de busca dos meta-arquivos, todos os 100 arquivos são criados com tamanho zero.

A execução foi, novamente, conduzida no i-cluster, utilizando um conjunto com somente um IOD, 16 meta-servidores e 16 clientes. Para maior clareza, estes serão referenciados numericamente de 0 a 15.

Os arquivos iniciais são criados no cliente #15, que por sua vez está associado ao meta-servidor #15. Em seguida, iniciando pelo cliente #14 e indo até o cliente #0, foi medido o tempo que cada um leva para ler os 100 arquivos (onde a operação "ler" significa `cat arq > /dev/null`). Apenas um cliente executa a cada medida.

Uma vez que, de acordo com a política linear implementada, a procura se inicia sempre pelo primeiro meta-servidor do conjunto, o acesso inicial do cliente #14 faz com que o meta-servidor #14 tente buscar o primeiro meta-arquivo no meta-servidor #0, obtendo um código de erro; em seguida, ele tenta o meta-servidor #1, que também retorna erro,

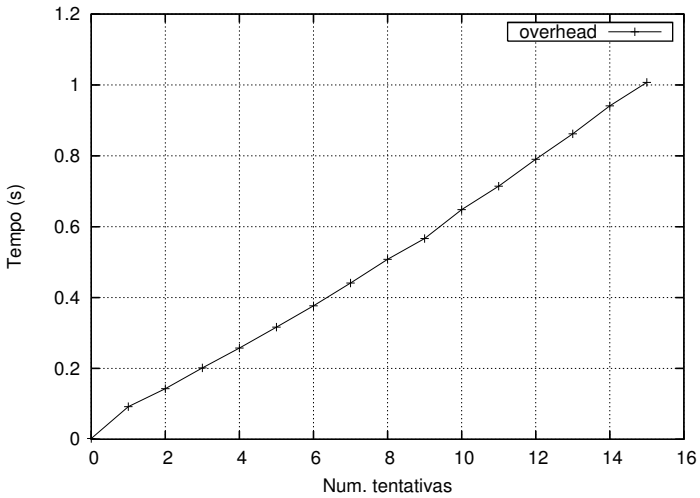

Figura 5.4: *Overhead* de busca de meta-arquivos

e então tenta o #2, o #3 e assim por diante até encontrar a informação desejada no meta-servidor #15. Essa mesma operação é repetida para todos os arquivos do conjunto inicial, e o tempo total gasto nesse procedimento é medido.

Na etapa seguinte do programa, passando para o cliente #13, os mesmos passos são repetidos, exceto que a procura se encerra com uma tentativa a menos, pois agora o meta-servidor #14 já possui os meta-arquivos desejados. O programa continua dessa forma até o cliente #0.

Os resultados obtidos são mostrados na Figura 5.4. Cada valor apresentado corresponde ao tempo medido para o número de tentativas em questão, dividido por 100 para representar o *overhead* por arquivo.

Observa-se que a curva aumenta, com o número de tentativas, de forma praticamente linear, a uma razão de aproximadamente 66 ms para cada "*cache-miss*" adicional. Esses valores correspondem a cerca de apenas 1 segundo de atraso no caso de serem usados 16 meta-servidores, valor considerado plenamente razoável, em comparação com o tempo necessário para ler ou escrever a totalidade de um arquivo da ordem de gigabytes.

5.3 *Benchmarks*

Em uma segunda etapa de experimentos, o protótipo do dNFSP foi avaliado com *benchmarks* bem conhecidos na literatura, e que são freqüentemente usados na avaliação de outros sistemas de arquivos paralelos. Este conjunto de experimentos visa uma avaliação do desempenho do dNFSP em uma situação mais próxima das aplicações reais (KASSICK et al., 2005).

5.3.1 BTIO

O *NAS Parallel Benchmarks*, ou NPB (BAILEY et al., 1995), é um conjunto de aplicações sintéticas em Dinâmica Computacional de Fluidos (CFD, *Computational Fluid Dynamics*) desenvolvido para permitir a avaliação do desempenho de supercomputadores. O NPB atual, em sua versão 2.4[1], apresenta 8 aplicações distintas, implementadas em MPI, que permitem a avaliação do desempenho sob diferentes aspectos (alto ou baixo

[1]Atualmente também é disponibilizado o NPB v3, voltado à computação em grade

grau de paralelismo, processamento regular e irregular, entre outros).

Uma das aplicações do NPB, o BT (*Block-Tridiagonal*), foi recentemente adaptado de forma a explorar os recursos de E/S de um computador paralelo, originando o *benchmark* conhecido como BTIO (WONG; WIJNGAART, 2003). O algoritmo BT consiste de um método implícito para resolver equações 3D Navier-Stokes; no BTIO o mesmo método é usado, porém os resultados devem ser escritos em disco a cada 5 passos de iteração, em um total de 200. Uma restrição para a execução da aplicação é que o número de processadores deve ser um quadrado perfeito (ou seja, 1, 4, 9, 16, e assim por diante). Três classes de tamanho são previstas, com base nas dimensões da matriz cúbica: classe A (64^3), classe B (102^3) e classe C (162^3).

O BTIO é disponibilizado em algumas variantes:

full Utiliza MPI-IO com o recurso de *collective buffering*, o que significa que os dados distribuídos na memória de todos os processadores são reunidos previamente em um sub-conjunto dos nós participantes, de modo a serem reordenados antes da escrita em disco, a fim de aumentar a granularidade dessa operação.

simple Também utiliza MPI-IO, porém sem o *collective buffering*; dessa forma, como nenhuma reordenação é feita, são necessárias muitas operações de *seek* na escrita dos dados.

fortran Semelhante ao *simple*, porém utiliza operações de manipulação de arquivos do Fortran 77 ao invés de chamadas ao MPI-IO.

epio Esta última variante não corresponde à especificação do *benchmark*, pois cada processo escreve os dados relativos à sua parte do domínio em um arquivo separado, de forma contínua. Para produzir o mesmo resultado das outras variantes, pode-se fazer, ao final do processamento, a combinação de todos os dados em um único arquivo, incluindo o tempo necessário para essa operação na temporização global da aplicação. Freqüentemente, entretanto, essa última etapa não é realizada, de modo a prover uma medida realística das capacidades do subsistema de E/S da máquina.

O experimento foi realizado utilizando-se a versão *epio*, pois o NFSP é implementado com base no NFS v2 e o MPI-IO requer NFS v3 para poder usufruir do mecanismo de travas dessa versão. Uma modificação realizada no *benchmark*, como feito usualmente em testes de sistemas de arquivos paralelos, foi a realização de escrita em disco a cada iteração, de modo a produzir uma carga maior. Comparou-se o desempenho do dNFSP com o PVFS e com o NFS tradicional (versão user-level).

Para este experimento foi utilizado o cluster LabTeC, em configurações com 1, 4, 9 e 16 nós de processamento. Quatro nós do cluster foram permanentemente utilizados para executar o servidor de arquivos. No caso do dNFSP, os quatro nós hospedam, cada um, um meta-servidor e um IOD. No caso do PVFS, cada nó hospeda um IOD e um dos nós hospeda o *manager*. Por último, quando do uso do NFS, somente um dos 4 nós foi utilizado como servidor.

A Figura 5.5 mostra os resultados obtidos, onde cada valor foi computado como a média aritmética de 5 execuções. Nota-se um desempenho bastante bom do dNFSP em todas as situações. Os dois sistemas de arquivos paralelos obtiveram, como esperado, melhor desempenho do que o NFS, chegando a um ganho de 32% do dNFSP em relação ao NFS no caso de 16 clientes. A quantidade de nós disponíveis no cluster não permitiu

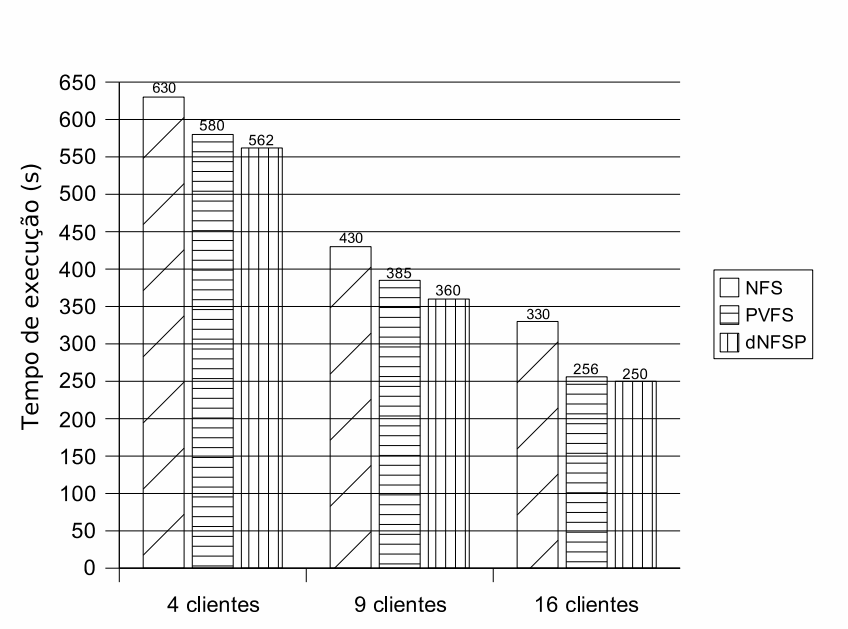

Figura 5.5: Comparação de desempenho do dNFSP em relação ao PVFS e ao NFS na execução do *benchmark* BTIO

a experimentação com um conjunto maior de clientes, porém o gráfico mostra tendência de aumento da diferença de desempenho à medida que se utilizam mais nós, o que corresponde ao resultado esperado.

5.3.2 Andrew Benchmark Distribuído

Outro *benchmark* utilizado para avaliar o desempenho do dNFSP foi o conhecido *Andrew Benchmark* (HOWARD et al., 1988). Esse pacote de software consiste de uma série de scripts que executam operações comuns de manipulação de arquivos e diretórios, tendo sido inicialmente criado para avaliar o desempenho do AFS, mas que é largamente utilizado para avaliação de desempenho de sistemas de arquivos em geral.

O Andrew é composto por 5 fases distintas: criação de diretórios, cópia de arquivos, verificação de atributos (`stat()`), leitura de dados e compilação de um programa. Cada uma delas procura explorar um tipo de operação no sistema de arquivos, sendo que a última fase realiza um combinação de todas as demais.

Para avaliar o desempenho de um sistema de arquivos paralelo como o dNFSP, o Andrew foi adaptado de modo a ter as operações realizadas por diversos clientes ao mesmo tempo. Assim, as cinco fases passam a se comportar da seguinte maneira:

Fase 1: Cada cliente cria um total de 200 sub-diretórios, tendo como raiz um diretório inicial cujo nome é baseado na identificação do próprio cliente. Ou seja, o conjunto de sub-diretórios criado por cada cliente é completamente independente dos demais. Esta fase, assim, se caracteriza pela escrita de meta-dados.

Fase 2: Uma série de arquivos presentes no sistema de arquivos local do cliente — efetivamente o código-fonte do programa compilado na última fase — é copiada para um dos diretórios criados na fase anterior. Esse procedimento caracteriza esta fase principalmente com a escrita de dados.

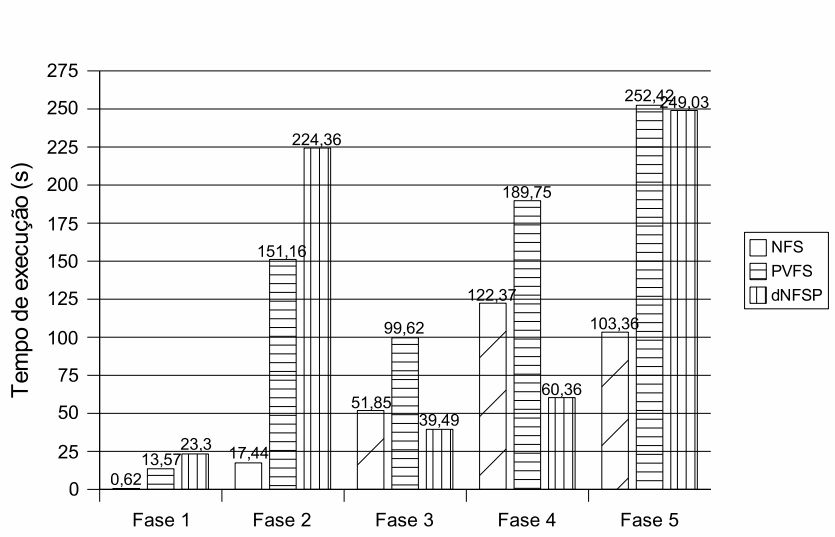

Figura 5.6: Comparação de desempenho do dNFSP em relação ao PVFS e ao NFS na execução do *benchmark* Andrew distribuído

Fase 3: A partir de uma listagem dos arquivos e diretórios locais (código-fonte do programa), realiza-se um `stat()`, através do comando *ls*, em cada um dos correspondentes remotos do itens listados, caracterizando a fase com a leitura de meta-dados.

Fase 4: Novamente utilizando a listagem de arquivos e diretórios locais, cada correspondente remoto tem seu conteúdo completamente lido através da utilização dos comandos *grep* e *wc*, característica essa herdada do Andrew original. Por esse comportamento, esta fase realiza principalmente operações de leitura de dados.

Fase 5: O programa é finalmente compilado, a partir da cópia feita na Fase 2, submetendo o sistema de arquivos aos quatro tipos de operações mencionadas de forma repetidamente alternada (a compilação do programa exige repetidas operações de leitura de arquivos-fonte, escrita de arquivos-objeto e criação e remoção de arquivos temporários).

Dois outros detalhes foram adicionados a esta versão modificada do Andrew. Primeiramente, os clientes executam o *benchmark* de forma síncrona, ou seja, ao final de cada fase, cada cliente aguarda que os demais atinjam o mesmo ponto. Isso é feito dessa forma para que os tempos de execução de cada fase possam ser medidos isoladamente. Em segundo lugar, a partição remota é montada e desmontada a cada fase; esse procedimento tem o objetivo de eliminar eventuais mascaramentos de desempenho pelo fato de dados ou meta-dados estarem em *cache*. O programa compilado é o POV-Ray (PERSISTENCE OF VISION RAY TRACER, 2001).

Os experimentos foram, igualmente, executados no cluster LabTeC, novamente comparando o desempenho obtido no dNFSP com o PVFS e o NFS tradicional. O número de clientes utilizados foi variado de 1 até 8. Os resultados obtidos são mostrados na Figura 5.6, correspondendo à execução com 8 clientes, uma vez que o comportamento observado foi proporcional a esta variação.

As medidas obtidas com a execução deste *benchmark* não se mostram, em geral, favoráveis aos sistemas de arquivos paralelos utilizados. O NFS proporciona menor tempo

de execução em diversas situações, em especial na Fases 1 e 2. Mesmo com o cenário geral desfavorável, ainda podem ser observadas situações onde o dNFSP apresentou ganho significativo de desempenho, como nas Fases 3 e 4, onde os clientes acessam massivamente o servidor de arquivos. Entende-se que o ganho obtido, nesses casos, deve-se à melhor distribuição do serviço, o qual fica centralizado no caso do NFS. Em relação ao PVFS, conclui-se que o problema de desempenho é originado pelo tamanho geralmente pequeno dos arquivos manipulados, situação em que o sistema sofre considerável degradação, como reconhecido pelos próprios autores.

Como justificativa para o comportamento global mais favorável ao NFS, pode-se observar que o Andrew foi concebido para a experimentação de um sistema de arquivos de propósito geral, procurando refletir a carga gerada por um grupo de usuários trabalhando em uma rede local, e não a de uma aplicação paralela. Assim, operações freqüentes envolvendo meta-dados e a manipulação de arquivos de pequeno porte, constantes no cenário cotidiano de uma rede, não são contempladas pelos principais objetivos dos sistemas de arquivos paralelos, que visam mais a manipulação dos dados em si. Contudo, dada a credibilidade do Andrew como parâmetro para a avaliação de sistemas de arquivos, decidiu-se reportar os dados obtidos mesmo que representem desvantagem para o dNFSP, como forma de tornar o estudo mais completo.

5.4 Aplicações

Para completar a avaliação do desempenho do dNFSP, o sistema foi experimentado com a execução de uma aplicação científica real, conforme apresentado a seguir.

5.4.1 GADGET

O GADGET (SPRINGEL; YOSHIDA; WHITE, 2001), abreviação de *Galaxies with Dark Matter and Gas Interact*, é um programa distribuído de forma livre para simulação cosmológica de N-corpos/SPH (*Smoothed Particle Hydrodynamics*). A aplicação realiza o cálculo de forças gravitacionais através de um algoritmo hierárquico em árvore, e pode ser usada para estudos isolados ou em simulações que incluam expansão cosmológica do espaço, com ou sem condições periódicas de borda (GADGET, 2005).

Por realizar a simulação da interação gravitacional entre partículas, o ciclo de funcionamento do programa é relativamente diferente do tradicional. Uma execução do GADGET não busca um dado específico, como o resultado de um cálculo, mas sim a observação da evolução do sistema inicial (posição das partículas, velocidade de deslocamento, forças exercidas, entre outros) dentro de um período de tempo previamente definido. O "resultado" da execução do GADGET é, portanto, um conjunto de dados semelhante ao utilizado como entrada.

Ao longo da execução do programa, em função da grande quantidade de cálculos que são realizados, é conveniente, em intervalos regulares de tempo, realizar um registro em disco da situação momentânea do sistema. Esse processo é chamado de *snapshot* e tem por objetivo evitar a perda de informações caso a aplicação seja interrompida no meio de sua execução (por exemplo, pela interrupção de energia elétrica). Ao final do tempo de execução pré-estabelecido, um *snapshot* final é também gerado.

Uma versão paralelizada do GADGET é implementada sobre MPI, e apresenta a possibilidade de que cada nó de processamento envolvido no cálculo gere um *snapshot* independente dos demais. Aproveitando esse recurso, uma vez que se configura a situação em que vários nós de processamento estão acessando o sistema de arquivos simultaneamente,

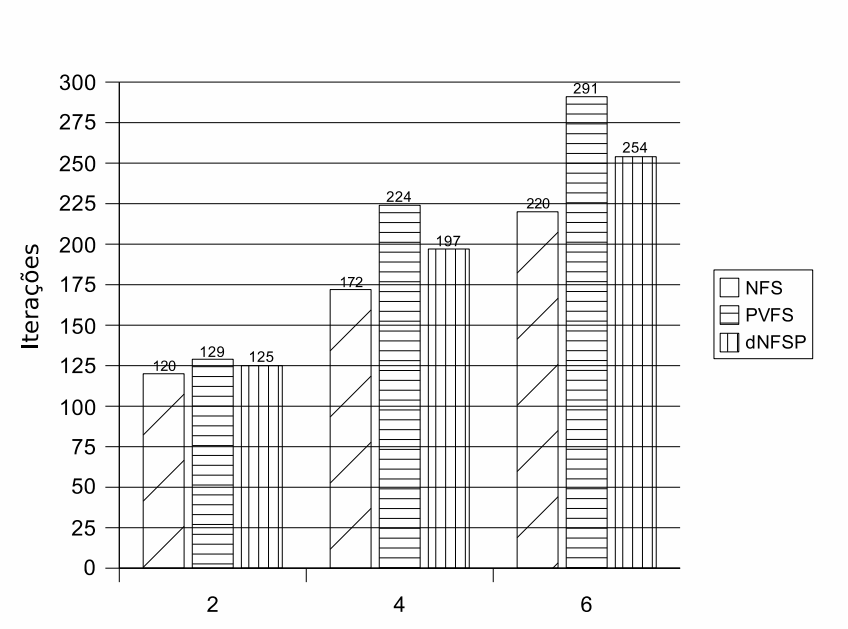

Figura 5.7: Resultados obtidos com a execução do GADGET

o GADGET foi usado para avaliar o desempenho do dNFSP. A vantagem de utilizar um sistema de arquivos paralelo para essa função é que, como o algoritmo é iterativo e a execução tem um tempo fixo de duração, uma eficiência maior de escrita de dados significa poder realizar um maior número de iterações.

Os resultados são apresentados na Figura 5.7. Os valores obtidos representam o número de iterações realizados, portanto um valor maior representa um resultado melhor. Os dados correspondem a execuções de 30 minutos. Também neste caso foi usado o cluster LabTeC, tendo sido igualmente reservados 4 nós para o servidor de arquivos e até 6 nós para a aplicação.

Nota-se novamente neste caso o melhor desempenho proporcionado pelos sistemas de arquivos paralelos, com o PVFS apresentando relativa superioridade. O dNFSP, mesmo assim, ainda obteve desempenho cerca de 15% superior àquele do NFS, apesar do ainda pequeno número de clientes.

5.5 Avaliação dos Resultados Obtidos

Após a realização dos experimentos recém descritos, pode-se chegar à conclusão de que o modelo proposto no dNFSP atende aos objetivos inicialmente definidos para o trabalho, em especial a adição de um nível diferenciado de desempenho ao NFSP e a manutenção de suas características de compatibilidade com os processos administrativos normalmente empregados em um cluster Beowulf.

As características principais do modelo, efetivamente a distribuição do meta-servidor e a manutenção de coerência com mecanismo relaxado, foram avaliadas diretamente pela execução dos experimentos de leitura e escrita seqüenciais e pela medida de *overhead* de busca de meta-arquivos. Os resultados obtidos demonstram um efetivo aumento de desempenho em relação ao modelo original, sem acarretar em atraso demasiado no caso da manutenção de coerência. O teste de escrita revelou a real possibilidade de maior vazão com o modelo de meta-servidor distribuído, ficando em média acima de 50% do

desempenho ideal, e que possivelmente ainda permite melhorias, como por exemplo no caso da implementação em nível de kernel. O teste de leitura comprovou a eficácia já apresentada pelo NFSP, e ainda permitindo um patamar mais elevado de desempenho, em torno de 18% melhor que o original, com uma curva de desempenho acompanhando a curva ideal em metade dos casos testados. Por fim, o teste de *overhead* de busca de meta-arquivos comprovou um valor tolerável, de cerca de 66 ms a cada procedimento de busca, o que representaria, no pior caso, cerca de 1% do tempo gasto para ler um arquivo de 1 GB em uma configuração com 16 meta-servidores.

As medidas com *benchmarks* objetivaram submeter o sistema a situações mais próximas daquelas encontradas nas aplicações dos usuários. Os testes com o BTIO revelaram desempenho em torno de 32% melhor do que o obtido usando NFS, e equiparável ao obtido com o PVFS. Essa característica reforça a idéia de que os objetivos do trabalho foram atingidos, pois o PVFS é um dos projetos com maior atividade, atualmente, na área de sistemas de arquivos para clusters, sendo resultado de grande quantidade de pesquisa. A execução do Andrew Benchmark distribuído revelou desempenho apenas regular de ambos os sistemas de arquivos paralelos, refletindo uma situação mais propícia para um sistema centralizado como o NFS.

Por fim, a execução do GADGET comprovou a viabilidade de uso do dNFSP com aplicações científicas reais, ficando novamente os resultados equiparáveis aos obtidos com o PVFS, e com ganho significativo sobre a execução com NFS.

O capítulo seguinte apresenta as principais conclusões obtidas após a realização do trabalho, destacando as contribuições da pesquisa desenvolvida e apontando possibilidades de extensão do projeto.

6 CONCLUSÕES E CONSIDERAÇÕES FINAIS

A Computação Baseada em Clusters representa, hoje em dia, a forma mais prática de se realizar pesquisa em processamento paralelo, pois os equipamentos podem ser adquiridos no mercado comum e o software é disponibilizado livremente na Internet. É também uma das mais economicamente atraentes alternativas para empresas que necessitam de alto poder de processamento. Essas e outras razões deram um grande impulso na pesquisa em clusters nos últimos anos, e estimularam o desenvolvimento e adaptação de tecnologias que, realimentando o ciclo, levaram os clusters ao nível dos computadores mais poderosos do mundo, apresentando centenas e mesmo milhares de processadores em uma mesma máquina.

Foi essa tendência de crescimento no porte das máquinas que motivou boa parte da pesquisa em sistemas de arquivos paralelos. Uma vez sendo constituído de centenas de nós de processamento, torna-se imperativo que o cluster abandone o modelo de servidor de arquivos centralizado, como é o caso do NFS, e passe a adotar uma solução que proporcione melhor desempenho e escalabilidade.

O projeto NFSP é uma das iniciativas que visam um sistema de arquivos com melhor desempenho, de modo que possa ser usado na computação baseada em clusters. Como caráter diferencial, o projeto busca o alto desempenho, porém é baseado no protocolo NFS padrão com o intuito de reduzir o impacto de introdução de novas tecnologias e métodos em uma instalação de cluster tradicional. Na prática, uma conseqüência dessa filosofia é que o NFSP permite o uso de clientes NFS convencionais, isentos de qualquer alteração, o que reduz significativamente o grau de intrusão na instalação e configuração dos nós do cluster.

O trabalho apresentado nesta tese, dNFSP, consiste em um dos ramos de pesquisa originados no projeto NFSP, e tem como meta permitir um melhor desempenho em situações onde aplicações paralelas realizam tarefas tanto de leitura como de escrita de dados, o que não é contemplado pelo modelo original.

A base de funcionamento do sistema se apóia sobre dois pontos principais: um modelo de gerenciamento distribuído de meta-dados, que permite melhor escalabilidade e reduz o custo computacional sobre o meta-servidor original do NFSP, e também um mecanismo relaxado de manutenção de coerência baseado em LRC (Lazy Release Consistency), o qual permite a distribuição do serviço sem acarretar em operações onerosas de sincronização de dados.

6.1 Metas Alcançadas e Contribuições do Trabalho

Após a apresentação do modelo de distribuição do servidor de arquivos proposto no dNFSP e analisados os resultados práticos obtidos, pode-se concluir que o trabalho alcan-

çou seus objetivos principais:

- Melhor escalabilidade e desempenho global da aplicação quando da realização de operações de escrita de dados, como ocorre no *benchmark* BTIO apresentado no capítulo anterior, com cerca de 32% de ganho de desempenho em relação ao NFS convencional;

- Importância de não acarretar, com a solução adotada, impacto no bom desempenho já apresentado pelo NFSP nas operações de leitura; como mostrado, a solução de distribuição do meta-servidor na verdade contribuiu para melhorar ainda mais o desempenho desse tipo de operação, e o *overhead* causado pela busca de meta-arquivos em servidores remotos foi considerado tolerável em relação ao custo das operações mais freqüentes;

- Manutenção do baixo nível de intrusividade na instalação e no gerenciamento de um cluster, em especial o fato de manter a compatibilidade com os clientes NFS disponíveis em qualquer sistema Unix.

Ao longo do desenvolvimento do trabalho, assim como fez o próprio NFSP, várias vezes tomou-se o PVFS como ponto de comparação, pelo fato de ser este um dos principais projetos da atualidade na área de sistemas de arquivos paralelos. Recentemente, a confiança no trabalho desenvolvido foi reforçada ainda mais com o lançamento oficial, em novembro de 2004, do PVFS v2, o qual inclui, entre outros recursos, a gerência distribuída de meta-dados, técnica que também é proposta na tese aqui apresentada.

Dentre as principais contribuições do trabalho desenvolvido, podem ser destacadas:

- Definição de um sistema de arquivos para clusters que permite bom desempenho e escalabilidade, e que ao mesmo tempo minimiza o esforço de adaptação a uma nova tecnologia por manter a compatibilidade com o software no lado cliente;

- Investigação sobre as possibilidades de uso do NFS, que apresenta importantes características de estabilidade, maturidade e semântica de uso bem conhecidas, como o sistema de arquivos para um cluster;

- Investigação sobre o potencial de utilização de um mecanismo de coerência fraca no sistema de arquivos de um cluster para processamento de alto desempenho;

- Contribuição para a evolução e amadurecimento do projeto NFSP como um todo;

- Reforço da parceria entre os grupos de pesquisa do ID, em Grenoble, e do GPPD, em Porto Alegre, através do desenvolvimento de atividades acadêmicas e científicas conjuntas.

6.2 Publicações

A pesquisa desenvolvida durante a tese proporcionou a publicação de artigos e resumos de IC nos seguintes eventos nacionais e internacionais:

- *CCGrid 2005*, artigo intitulado *Evaluating the Performance of the dNFSP File System* (KASSICK et al., 2005), apresenta os resultados dos *benchmarks* BTIO e Andrew

- *ERAD 2005*, resumo de IC intitulado *Avaliação de Desempenho de Sistemas de Arquivos Paralelos com o Andrew Benchmark Modificado* (MACHADO et al., 2005), apresenta resultados preliminares com o Andrew Benchmark

- *SBAC-PAD 2004*, artigo *Performance Evaluation of a Prototype Distributed NFS Server* (ÁVILA et al., 2004) com os resultados obtidos nos testes de leitura e escrita seqüenciais e medida de *overhead* do mecanismo de busca

- *WSPPD 2004*, artigo *Performance Evaluation of a Prototype Distributed NFS Server* (ÁVILA; NAVAUX; DENNEULIN, 2004), versão resumida do artigo anterior

- *WSGPPD 2003*, artigo intitulado *A Comparison on Current Distributed File Systems for Beowulf Clusters* (ÁVILA; NAVAUX; DENNEULIN, 2003), apresenta um estudo comparativo entre sistemas de arquivos paralelos para clusters

6.3 Trabalhos Futuros

Com as atividades desenvolvidas no presente trabalho, alguns novos temas de pesquisa podem desde já ser apontados como possibilidades de continuação do projeto NFSP:

Protocolo dedicado de comunicação entre os meta-servidores Os resultados mostrados nos testes de *overhead* de busca de meta-arquivos comprovaram que a estratégia de realizar a cópia remota de meta-arquivos por meio do comando *rcp* apresenta custo tolerável, de cerca de 66 ms por tentativa. É possível, entretanto, que o desempenho desta operação possa ser ainda melhorado pela implementação de um protocolo dedicado para troca de meta-dados entre os servidores. Como mencionado no capítulo anterior, tal mecanismo chegou a ser implementado em parte, por meio de conexões TCP, o que mostrou custo demasiadamente elevado de inicialização. Uma solução dedicada, baseada em UDP, parece ser uma boa alternativa.

Tolerância a falhas nos IODs Um ponto certamente de grande interesse para o projeto NFSP é a inclusão de algum mecanismo de tolerância a falhas no armazenamento dos dados, possivelmente implementado diretamente sobre os IODs. A garantia de ter as informações armazenadas com segurança é ponto-chave de muitas aplicações e usuários de clusters. Atualmente, os IODs somente armazenam cópias individuais de cada bloco de dados dos arquivos do sistema. Um mecanismo do tipo RAID em software, como feito no xFS, é certamente uma escolha viável e eficaz, devendo entretanto ser feita uma avaliação prévia sobre o impacto desse mecanismo no desempenho do sistema de arquivos.

Integração com o servidor NFS em nível de kernel Assim como foi feito com a primeira implementação do NFSP, onde o alto custo computacional no meta-servidor levou ao porte do código em nível de usuário para o nível de kernel, a adaptação do dNFSP para o servidor NFS presente no kernel do Linux pode representar menor carga na CPU da máquina hospedeira e conseqüentemente melhor desempenho por parte das aplicações. Tal problema pode ser observado nos experimentos com elevado número de clientes, e uma implementação no nível do kernel poderia propiciar menos *overhead*.

Porte para o NFS v3 O NFSP teve sua implementação baseada no NFS v2, e com isso algumas restrições podem ser observadas, como a limitação do tamanho dos arquivos em 2 GB e a ausência do mecanismo de travas. Essas restrições impedem, por exemplo, o uso de software como o MPI-IO, que utiliza aquele mecanismo no controle de acesso aos arquivos que manipula. O porte do NFSP e conseqüentemente do dNFSP para o NFS v3 eliminaria essas limitações, possibilitando a execução de novos experimentos e aplicações com o sistema.

REFERÊNCIAS

AMZA, C.; COX, A. L.; DWARKADAS, S.; KELEHER, P.; LU, H.; RAJAMONY, R.; YU, W.; ZWAENEPOEL, W. TreadMarks: shared memory computing on networks of workstations. **IEEE Computer**, Los Alamitos, v.29, n.2, p.18–28, Feb. 1996.

ANDERSON, T. E.; DAHLIN, M. D.; NEEFE, J. M.; PATTERSON, D. A.; ROSELLI, D. S.; WANG, R. Y. Serverless Network File Systems. In: SYMPOSIUM ON OPERATING SYSTEMS PRINCIPLES, 15., 1995, Copper Mountain Resort, Colorado. **Proceedings...** [S.l.: s.n.], 1995. p.109–126.

ÁVILA, R. B.; NAVAUX, P. O. A.; DENNEULIN, Y. A Comparison on Current Distributed File Systems for Beowulf Clusters. **Cadernos de Informática**, Porto Alegre, v.3, n.1, p.121–126, jun. 2003. Trabalho apresentado no 1. Workshop do Grupo de Processamento Paralelo e Distribuído.

ÁVILA, R. B.; NAVAUX, P. O. A.; DENNEULIN, Y. Performance Evaluation of a Prototype Distributed NFS Server. In: WORKSHOP DE PROCESSAMENTO PARALELO E DISTRIBUÍDO, 2., 2004, Porto Alegre. **Anais...** Porto Alegre: Instituto de Informática da UFRGS, 2004. p.161–166.

ÁVILA, R. B.; NAVAUX, P. O. A.; LOMBARD, P.; LEBRE, A.; DENNEULIN, Y. Performance Evaluation of a Prototype Distributed NFS Server. In: SYMPOSIUM ON COMPUTER ARCHITECTURE AND HIGH-PERFORMANCE COMPUTING, 16., 2004, Foz do Iguaçu, Brazil. **Proceedings...** Washington: IEEE, 2004. p.100–105.

BAILEY, D. H.; HARRIS, T.; SAPHIR, W.; WIJNGAART, R. van der; WOO, A.; YARROW, M. **The NAS Parallel Benchmarks 2.0**. Moffett Field, CA: NASA Ames Research Center, 1995. (NAS-95-020).

BAKER, M. (Ed.). **Cluster Computing White Paper**. [S.l.]: IEEE Task Force in Cluster Computing, 2000. Disponível em: <http://www.dcs.port.ac.uk/∼mab/tfcc/WhitePaper/final-paper.pdf>. Acesso em: mar. 2003. Final Release, Version 2.0.

The BERKELEY NOW Project. Disponível em: <http://now.cs.berkeley.edu>. Acesso em: jan. 2005.

BODEN, N. et al. Myrinet: a gigabit-per-second local-area network. **IEEE Micro**, Los Alamitos, v.15, n.1, p.29–36, Feb. 1995.

BUYYA, R. (Ed.). **High Performance Cluster Computing**: architectures and systems. Upper Saddle River: Prentice Hall PTR, 1999. 849p.

CALLAGHAN, B.; PAWLOWSKI, B.; STAUBACH, P. **NFS Version 3 Protocol Specification**: RFC 1831. [S.l.]: Internet Engineering Task Force, Network Working Group, 1995.

CARNS, P. H.; LIGON III, W. B.; ROSS, R. B.; THAKUR, R. PVFS: a parallel file system for Linux clusters. In: ANNUAL LINUX SHOWCASE AND CONFERENCE, 4., 2000, Atlanta, GA. **Proceedings...** [S.l.: s.n.], 2000. p.317–327. Best Paper Award.

CLUSTER FILE SYSTEMS, INC. **Lustre**: a scalable, high-performance file system. Disponível em: <http://www.lustre.org/docs/whitepaper.pdf>. Acesso em: jul. 2004.

DILLON, E.; SANTOS, C. G. dos; GUYARD, J. Homogeneous and Heterogeneous Networks of Workstations: message passing overhead. In: MPI DEVELOPERS CONFERENCE, 1995, Notre-Dame, IN. **Proceedings...** [S.l.: s.n.], 1995.

FREE Software Foundation. Disponível em: <http://www.fsf.org>. Acesso em: jan. 2005.

GADGET. Disponível em: <http://www.mpa-garching.mpg.de/galform/gadget/index.shtml>. Acesso em: jan. 2005.

GEIST, A. et al. **PVM**: parallel virtual machine. Cambridge: MIT Press, 1994.

HELLWAGNER, H.; REINEFELD, A. (Ed.). **SCI**: Scalable Coherent Interface: architecture and software for high-performance compute clusters. Berlin: Springer-Verlag, 1999. 490p. (Lecture Notes in Computer Science, v.1734).

HIPPER, G.; TAVANGARIAN, D. Advanced Workstation Cluster Architectures for Parallel Computing. **Journal of Systems Architecture**, Amsterdam, v.44, n.3/4, p.207–226, Dec. 1997.

HOWARD, J. H.; KAZAR, M. L.; MENEES, S. G.; NICHOLS, D. A.; SATYANARAYANAN, M.; SIDEBOTHAM, R. N.; WEST, M. J. Scale and Performance in a Distributed File System. **ACM Transactions on Computer Systems**, New York, v.6, n.1, p.51–81, 1988.

INSTITUTE OF ELECTRICAL AND ELECTRONIC ENGINEERS. **IEEE 1596-1992**: IEEE standard for scalable coherent interface (SCI). New York, 1992.

INSTITUTE OF ELECTRICAL AND ELECTRONIC ENGINEERS. **IEEE 802.3u-1995**: local and metropolitan area networks-supplement—media access control (MAC) parameters, physical layer, medium attachment units and repeater for 100Mb/s operation, type 100BASE-T (clauses 21–30). New York, 1995.

INSTITUTE OF ELECTRICAL AND ELECTRONIC ENGINEERS. **IEEE 802.3z-1998**: information technology–telecommunications and information exchange between systems–local and metropolitan area networks–specific requirements–part 3: carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications. New York, 1998.

KASSICK, R.; MACHADO, C.; HERMANN, E.; ÁVILA, R.; NAVAUX, P.; DENNEULIN, Y. Evaluating the Performance of the dNFSP File System. In: IEEE INTERNATIONAL SYMPOSIUM ON CLUSTER COMPUTING AND THE GRID, CCGRID, 5., 2005, Cardiff, UK. **Proceedings...** Los Alamitos: IEEE Computer Society Press, 2005.

KELEHER, P.; COX, A. L.; ZWAENEPOEL, W. Lazy Release Consistency for Software Distributed Shared Memory. In: ANNUAL INTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECTURE, 19., 1992, Gold Coast, Queensland, Australia. **Proceedings...** New York: ACM Press, 1992. p.13–21.

KIM, G. H.; MINNICH, R. G.; MCVOY, L. **Bigfoot-NFS**: a parallel file-striping NFS server. Disponível em: <http://public.lanl.gov/rminnich/vecrpc/bigfoot.ps>. Acesso em: jan. 2005.

LEBRE, A. **Composition de Service de Données et de Méta-Données dans un Système de Fichiers Distribué**. Grenoble: Laboratoire Informatique et Distribution, 2002.

The LINUX Homepage. Disponível em: <http://www.linux.org>. Acesso em: jan. 2005.

LIU, W.; OU, X.; ZHENG, W.; WU, M.; SHEN, M. Prefetching and Caching Metadata in a Distributed NFS Server. In: IEEE INTERNATIONAL CONFERENCE ON ALGORITHMS AND ARCHITECTURES FOR PARALLEL PROCESSING, 4., 2000, Hong Kong. **Proceedings...** [S.l.: s.n.], 2000. p.490–495.

LIU, W.; ZHENG, W.; SHEN, M.; OU, X.; WU, M. Design and Implementation of a Distributed NFS Server on Cluster of Workstations. In: IASTED INTERNATIONAL CONFERENCE ON PARALLEL AND DISTRIBUTED COMPUTING AND SYSTEMS, 12., 2000. **Proceedings...** [S.l.: s.n.], 2000. p.7–12.

LOMBARD, P. **NFSP** : une solution de stockage distribué pour architectures grande échelle. 2003. Thèse — Institut National Polytechnique de Grenoble, Grenoble.

MACHADO, C. d. S.; KASSICK, R. V.; ÁVILA, R. B.; NAVAUX, P. O. A. Avaliação de Desempenho de Sistemas de Arquivos Paralelos com o Andrew Benchmark Modificado. In: ESCOLA REGIONAL DE ALTO DESEMPENHO, 5., 2005, Canoas. **Anais...** Canoas: SBC, 2005. p.153–156.

MPI FORUM. **The MPI Message Passing Interface Standard**. Knoxville: University of Tennessee, 1994.

MUNTZ, D. **Building a Single Distributed File System from Many NFS Servers**. Palo Alto: HP Laboratories, 2001. (HPL-2001-176).

OPEN Source Initiative. Disponível em: <http://www.opensource.org>. Acesso em: jan. 2005.

PATTERSON, D. A.; GIBSON, G. A.; KATZ, R. H. A Case for Redundant Arrays of Inexpensive Disks (RAID). In: ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, 1988, Chicago, IL. **Proceedings...** New York: ACM Press, 1988. p.109–116.

PERSISTENCE of Vision Ray Tracer. Disponível em: <http://www.povray.org>. Acesso em: abr. 2001.

PRESLAN, K. W.; BARRY, A. P.; BRASSOW, J. E.; ERICKSON, G. M.; NYGAARD, E.; SABOL, C. J.; SOLTIS, S. R.; TEIGLAND, D. C.; O'KEEFE, M. T. A 64-bit, Shared Disk File System for Linux. In: IEEE SYMPOSIUM ON MASS STORAGE SYSTEMS, 16., 1999, San Diego, California. **Proceedings...** Los Alamitos: IEEE Computer Society, 1999. p.22–41.

QSNET High Performance Interconnect. Disponível em: <http://www.quadrics.com/website/pdf/qsnet.pdf>. Acesso em: jun. 2003.

SANDBERG, R.; GOLDBERG, D.; KLEIMAN, S.; WALSH, D.; LYON, B. Design and Implementation of the Sun Network Filesystem. In: SUMMER USENIX CONFERENCE, 1985, Portland, OR, USA. **Proceedings...** [S.l.: s.n.], 1985. p.119–130.

SATYANARAYANAN, M. Scalable, Secure, and Highly Available Distributed File Access. **IEEE Transactions on Computers**, Los Alamitos, v.23, n.5, p.9–21, May 1990.

SATYANARAYANAN, M.; KISTLER, J. J.; KUMAR, P.; OKASAKI, M. E.; SIEGEL, E. H.; STEERE, D. C. Coda: a highly available file system for a distributed workstation environment. **IEEE Transactions on Computers**, Los Alamitos, v.39, n.4, p.447–459, Apr. 1990.

SAVARESE, D. F.; STERLING, T. Beowulf. In: BUYYA, R. (Ed.). **High Performance Cluster Computing**: architectures and systems. Upper Saddle River: Prentice Hall PTR, 1999. p.625–645.

SCHIKUTA, E.; STOCKINGER, H. Parallel I/O for Clusters: methodologies and systems. In: BUYYA, R. (Ed.). **High Performance Cluster Computing**: architectures and systems. Upper Saddle River: Prentice Hall PTR, 1999. p.439–462.

SCHMUCK, F.; HASKIN, R. GPFS: a shared-disk file system for large computing clusters. In: CONFERENCE ON FILE AND STORAGE TECHNOLOGIES, 2002, Monterey, CA. **Proceedings...** [S.l.: s.n.], 2002. p.231–244.

SHEN, X.; CHOUDHARY, A. DPFS: a distributed parallel file system. In: INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING, 2001, Valencia, Spain. **Proceedings...** Los Alamitos: IEEE Computer Society, 2001. p.533–544.

SOLTIS, S.; ERICKSON, G.; PRESLAN, K.; O'KEEFE, M.; RUWART, T. The Design and Performance of a Shared Disk File System for IRIX. In: GODDARD CONFERENCE ON MASS STORAGE SYSTEMS AND TECHNOLOGIES, 6., 1998, College Park, Maryland. **Proceedings...** [S.l.: s.n.], 1998. p.41–56.

SPRINGEL, V.; YOSHIDA, N.; WHITE, S. D. M. GADGET: a code for collisionless and gasdynamical cosmological simulations. **New Astronomy**, Amsterdam, v.6, p.79–117, 2001.

STERLING, T.; BECKER, D. J.; SAVARESE, D.; DORBAND, J. E.; RANAWAKE, U. A.; PACKER, C. V. BEOWULF: a parallel workstation for scientific computation. In: INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING, 24., 1995, Oconomowoc, WI. **Proceedings...** [S.l.: s.n.], 1995. p.11–14.

STERLING, T. L. **Beowulf Cluster Computing with Linux**. Cambridge: MIT Press, 2002.

STERLING, T. L.; SALMON, J.; BECKER, D. J.; SAVARESE, D. F. **How to Build a Beowulf**: a guide to the implementation and application of PC clusters. Cambridge: MIT, 1999. 239p.

SWEENEY, A.; DOUCETTE, D.; HU, W.; ANDERSON, C.; NISHIMOTO, M.; PECK, G. Scalability in the XFS File System. In: USENIX 1996 TECHNICAL CONFERENCE, 1996, San Diego, CA, USA. **Proceedings...** [S.l.: s.n.], 1996. p.1–14.

THEKKATH, C. A.; MANN, T.; LEE, E. K. Frangipani: a scalable distributed file system. In: ACM SYMPOSIUM ON OPERATING SYSTEMS PRINCIPLES, 16., 1997, Saint Malo, France. **Proceedings...** New York: ACM Press, 1997. p.224–237.

TOP500. Disponível em: <http://www.top500.org>. Acesso em: jan. 2005.

WONG, P.; WIJNGAART, R. F. Van der. **NAS Parallel Benchmarks I/O Version 2.4**. Moffett Field, CA: NASA Advanced Supercomputing (NAS) Division, 2003. (NAS-03-002).

ANEXO A PUBLICAÇÕES

As publicações mais relevantes obtidas ao longo do desenvolvimento da tese são incluídas aqui para referência. São os artigos:

- KASSICK, R.; MACHADO, C.; HERMANN, E.; ÁVILA, R.; NAVAUX, P.; DENNEULIN, Y. Evaluating the Performance of the dNFSP File System. In: IEEE INTERNATIONAL SYMPOSIUM ON CLUSTER COMPUTING AND THE GRID, CCGRID, 5., 2005, Cardiff, UK. **Proceedings...** Los Alamitos: IEEE Computer Society Press, 2005.

- ÁVILA, R. B.; NAVAUX, P. O. A.; LOMBARD, P.; LEBRE, A.; DENNEULIN, Y. Performance Evaluation of a Prototype Distributed NFS Server. In: SYMPOSIUM ON COMPUTER ARCHITECTURE AND HIGH-PERFORMANCE COMPUTING, 16., 2004, Foz do Iguaçu, Brazil. **Proceedings...** Washington: IEEE, 2004. p.100–105.

- ÁVILA, R. B.; NAVAUX, P. O. A.; DENNEULIN, Y. A Comparison on Current Distributed File Systems for Beowulf Clusters. **Cadernos de Informática**, Porto Alegre, v.3, n.1, p.121–126, jun. 2003. Trabalho apresentado no 1. Workshop do Grupo de Processamento Paralelo e Distribuído.

Um artigo com novos resultados, recentemente obtidos no i-Cluster2, foi preparado e será submetido a uma conferência. O trabalho, intitulado “An Analysis on the Scalability of the dNFSp File System”, é igualmente incluído ao final deste apêndice.

Evaluating the Performance of the dNFSP File System

Rodrigo Kassick[1,*], Caciano Machado[1,†], Everton Hermann[1,‡],
Rafael Ávila[1,2,‡], Philippe Navaux[1], Yves Denneulin[2]

[1] Instituto de Informática/UFRGS
Caixa Postal 15064
91501-970 Porto Alegre – Brazil
Email: {avila,navaux}@inf.ufrgs.br

[2] Laboratoire ID/IMAG
51, avenue Jean Kuntzmann
38330 Montbonnot-Saint Martin – France
Email: *First.Last*@imag.fr

Abstract

Parallel I/O in cluster computing is one of the most important issues to be tackled as clusters grow larger and larger. Many solutions have been proposed for the problem and, while effective in terms of performance, they usually represent a considerable amount of hacking into a "traditional" Beowulf cluster installation. In this paper, we investigate a parallel solution based on NFS, which reduces the level of intrusion in the file server installation, keeps the client side untouched, and still provides an improved level of performance and scalability for parallel applications. We compare our proposal to other existing file systems using known benchmarks, and demonstrate that it is a valid alternative for general-purpose cluster computing.

1. Introduction

High performance file systems are nowadays a special source of attention for researchers in cluster computing, since technology and prices allow one to build machines that are each time larger and larger in size and capacity. In the November 2004 TOP500 list[1], 294 parallel machines are classified as clusters, and many of them feature more than 1000 processors. This trend has led researchers and vendors around the world to adapt existing (mostly commercial) high-performance I/O solutions as well as to design and implement new parallel file systems, which might be better suited to the "commodity-of-the-shelf" approach of Beowulf cluster computing [16].

In any case, solutions for the performance and scalability of cluster file systems generally present a significant amount of *intrusion* in a standard cluster installation, meaning that new tools, daemons and/or kernel drivers must be compiled, installed and configured. Traditional Beowulf software like the GNU/Linux system, TCP/IP, NFS and the like are well-known to system administrators and management tools, representing a good share of confidence and stability in a typical cluster installation. In this way, we believe that a solution for parallel I/O that could be built up from such established systems might be of considerable interest to cluster computing facilities.

Following this approach, in this paper we present a performance evaluation of *dNFSP* [1], an extension of the standard NFS file server intended for an improved level of performance and scalability on clusters while still maintaining compatibility with the standard NFS clients available on every Unix system. Our main goal is to evaluate the feasability of dNFSP as an alternative for cluster file systems.

We begin with a brief introduction to dNFSP and its principles, as well as its relation to other solutions for parallel I/O. Section 4 then describes the benchmarks and criteria used to evaluate dNFSP, followed by Section 5 that presents and discusses the obtained results. Finally we present in Section 6 our conclusions and future activities.

2. dNFSP – Distributing NFS

NFSP [8] is a project started at the *Laboratoire Informatique et Distribution*, in Grenoble, France, with the goal of providing an improved level of performance to a standard NFS [3] server. By following this ap-

---

∗ PIBIC/CNPq research assistant
† Dell/UFRGS research assistant
‡ Work supported by HP Brazil
1 http://www.top500.org

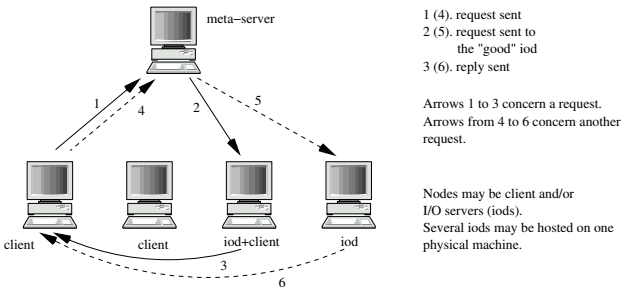

Figure 1. NFSP architecture

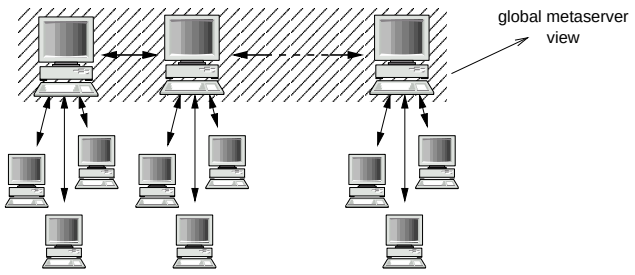

Figure 2. Meta-servers are replicated in dNFSP

proach, the project expects to reach a level of performance and scalability suitable for many parallel applications, and at the same time keep the intrusion level in relation to a traditional cluster installation to a minimum.

2.1. Design of NFSP

The principle behind NFSP is inspired in PVFS [4]. The functionality of the NFS server is split into two parts: a set of I/O nodes, or *iods* for short, which are responsible for storing and retrieving the data blocks that result from file striping, and a *meta-server*, which plays the main role in NFSP: it appears as the "normal" NFS server for the clients but, upon receiving a request, instead of reading/writing the data on its local file system, it forwards the request to the appropriate iods, which then respond to the clients as needed. Once several requests are received by the meta-server, they are forwarded to the iods, which can work in parallel, thus improving performance. Figure 1 illustrates one possible scenario. Iods can be run on client machines without any restriction. This allows one to easily benefit from the (usually forgotten) disk space on the compute nodes.

Several variations of NFSP have been implemented so far [9, 10], being based on both the user- and (Linux) kernel-level implementations of the standard NFS v2. One common characteristic among them is that the meta-server is one single process run on one of the nodes. This design allows for increased performance in the case of read operations, because most of the data involved in the complete operation are sent directly from the iods to the clients; however, in the case of writes, the data must be sent from the clients to the meta-server, and thus the original bottleneck remains the same.

2.2. dNFSP

In order to alleviate this problem, *dNFSP* [1] has been proposed as a variation of NFSP in which the meta-server is replicated onto several compute nodes. Figure 2 illustrates the new design.

In this approach, each replica of the meta-server works exactly as in the original model, receiving requests from the clients and forwarding them to the iods. However, one single replica serves only a subset of the clients, which see it as the only NFS server in the system (i.e. the client side is still unchanged). For example, if there are 4 meta-server replicas and 20 clients, each replica can be bound to 5 clients in order to balance the system. Now if all the clients need to access the server at the same time, several meta-server entry points (4, in the example) will be used instead of just one, thus allowing for an increased overall bandwidth also for write operations. Each replica is still capable of accessing the whole set of iods.

As a side-effect of meta-server replication, however, a new problem arises: to keep meta-data consistency among the several replicas. For example, if a client creates a new file, this file will exist on the meta-server that client is bound to, but not on the others, since meta-data are stored on each meta-server's local file system. This means that the new file will be visible for every client bound to the same meta-server, but not for the others.

In order to maintain meta-data coherence among the meta-server replicas without incurring in too much overhead, a mechanism based on LRC (Lazy Release Consistency [7]) is used. In this mechanism, meta-data are made consistent (which means to copy some information from one remote meta-server to another) only when effectively needed by the clients. This situation is basically detected when a client tries to access a file that *apparently* does not exist (*lookup* function in NFS). At this moment the meta-server in ques-

tion starts searching for the replica that contains the file, and copies it when found. We rely on the fact that accesses to really inexistent files should not occur (at least not often) in a parallel application. More details on the model are presented on another paper [1].

3. Related Work

The problem of efficient I/O in parallel computing arises every time the number of compute nodes grows beyond a few nodes (naturally, given that the application needs it). In this way many solutions for high performance storage in clusters and parallel machines in general have been proposed. Two main approaches seem to exist. The first one achieves improved performance by making use of dedicated hardware like high speed data links (e.g. fiber optics), redundant storage, non-volatile RAM, and several combinations thereof. This approach is mostly used by the file systems of commercial parallel machines, such as IBM GPFS [14] and Sistina GFS [15, 12]. Following another direction, file systems such as PVFS [4] and Lustre [5] try to obtain better performance by distributing the file system functionalities among the compute nodes. Since this approach generally does not require the use of any special kind of hardware, it is better suited to the philosophy behind cluster computing.

dNFSP belongs in the second group. The main difference in relation to other distributed file systems is the replicated meta-server design and its LRC-based coherence mechanism, which allows for reduced overhead operation in applications with low meta-data profile. Another aspect that distinguishes NFSP in general is the NFS compatibility. By building the system upon the traditional NFS foundation, we aim at obtaining a system with well-known configuration and management procedures, thus reducing the impact of introducing a new technology, while being able to keep with a good level of both performance and scalability which are suitable for many parallel applications.

In order to better evaluate the performance levels of dNFSP, we have carried out a series of experiments with traditional file system benchmarks found in the literature. Such experiments and test-bed are described next.

4. Description of the Experiments

The goal of our analysis is to evaluate the level of performance presented by dNFSP in comparison to a real cluster file system, as well as to measure the impact of our extensions to the traditional NFS server in comparison to an unmodified version of that system. It is important to clarify that we do not consider NFS a proper high-performance cluster file system, but rather recognise that it is widely used for that purpose on environments where parallel I/O applications are not dominant. In this way, we try to evaluate dNFSP in both situations, by using benchmarks and applications for both general-purpose and high-performance file systems.

4.1. Distributed Andrew Benchmark (DAB)

In order to evaluate the performance of the proposed filesystem, as well as the overhead caused by splitting the files through IODs and replicating metafiles in metaservers, we have created a variation of the well-known Andrew Benchmark, here called Distributed Andrew Benchmark.

The original Andrew Benchmark was conceived to test the performance of the Andrew File System (AFS) [6], also a distributed file system. It tries to simulate the load that would be achieved in normal use of the file system with several users connected. The original benchmark is meant to be run on a single machine. In our modified version, we intend to evaluate the performance of several nodes accessing the distributed file server. This is done by executing several instances of the benchmark on different machines (here called *clients*).

In order to measure the times of meta-server synchronization and data I/O independently, the benchmark was modified to make all clients execute phases in a coordinated manner, i.e., all clients must complete a phase before proceeding to the next one. Also, as a guarantee that no results are masked by data caching or buffering, the remote file system is mounted and unmounted respectively at the beggining and at the end of each phase.

The modified benchmark is composed of five phases:

mkdir Creates the directories which will be used in the next phases.

cp Each client makes a copy of the original tree in its own directory in the shared file system.

stat The benchmark performs a *stat* in each file in the client's directory.

read Reads all the contents of all files in the client's directory.

make Compiles an average size program (in the case of this test, POV-Ray[2]).

2 http://www.povray.org

Phases 1 & 3 focus on how efficient and scalable the file system is when accessing the *metafiles*. Phases 2 & 4 try to evaluate the performance of data access. Phase 5 tries to measure the file system performance in a situation where both metafiles and data are needed.

4.2. The NAS/BTIO Benchmark

The NAS Parallel Benchmarks (NPB) are a set of applications based on Computational Fluid Dynamics (CFD) designed to help evaluate the performance of parallel supercomputers. There are several flavours of the NPB, allowing to evaluate different aspects. The BTIO benchmark tool is the responsible to evaluate the storage performance. It is an extension of the BT benchmark [2] which is based on a CFD code that uses an implicit algorithm to solve the 3D compressible Navier-Stokes equations. The BTIO version of the benchmark uses the same computational method, but with the addition that results must be written to disk at every fifth time step. There are different versions of BTIO, which are described below:

- BTIO-full-mpiio: uses MPI-IO file operations with *collective buffering*, which means that data blocks are potentially re-ordered previously to being written to disk, resulting in coarser write granularity
- BTIO-simple-mpiio: Also uses MPI-IO operations, but no data re-ordering is performed, resulting in a high number of seeks when storing information on the file system
- BTIO-fortran-direct: This version is similar to simple-mpiio, but uses the Fortran direct access method instead of MPI-IO
- BT-epio: In this version each node writes in a separate file. This test gives the optimal write performance that can be obtained, because the file isn't shared by all the processes, so there is no lock restrictions. In order to compare with other versions, the time to merge the files must be computed, as required by the Application I/O benchmark specification.

There is one restriction to run the test: the number of processes must be a perfect square $(1, 4, 9, 16, \ldots)$. To determine the amount of memory required for the run, a class of problem size must be chosen which represents the cubic matrix dimensions : Class A (64^3), Class B (102^3), Class C (162^3). The original code runs for 200 iterations and writes at every five iterations.

The tests were performed using only the epio version of the benchmark, since dNFSP was designed based on NFSv2 protocol, and MPI-2 IO requires NFSv3 to control the file access using locks. Also we have developed another version of BTIO to perform writes on every iteration instead of every five iterations, resulting in a more intensive write test.

5. Experimental Results

The experiments have been carried out on the LabTeC[3] cluster. This machine is composed of 20 nodes interconnected by Fast Ethernet, where each node features two Pentium III processors at 1 GHz, 1 GB RAM and one 18 GB SCSI hard disk. The operating system on all nodes is Debian GNU/Linux with kernel 2.4.26. All systems and applications have been compiled (where appropriate) with GCC v2.95.

5.1. The Analyzed File Systems

In order to compare the performance of dNFSP we have selected two representative file systems according to our evaluation criteria mentioned before:

UNFS The user level version of the widely used Network File System. We considered the values obtained with NFS [11] as a base of comparision to the parallel file system measures in the situation where cluster applications are not demanding high performance I/O. The results have been obtained with UNFS v2.2beta47.

PVFS The well-known parallel file system for the Beowulf world. The main aspect that differs PVFS from dNFSP is the fact that PVFS1 does not implement multiple meta-servers[4]. PVFS is developed jointly by the Parallel Architecture Research Laboratory (PARL) at Clemson University and The Mathematics and Computer Science Division at Argonne National Laboratory. The version used is PVFS 1.6.2.

In the PVFS and dNFSP experiments, a set of 4 nodes has been dedicated to the file server, each one holding one iod. The PVFS manager runs together with the first iod. For dNFSP, the 4 corresponding meta-servers also share the same nodes with the iods, and clients are evenly distributed among them. For each measured value, the mean of a series of five executions is presented.

---

3 Deployed within the context of a partnership between Dell Computers and the Instituto de Informática since 2002 (http://www.inf.ufrgs.br/LabTeC)

4 This feature is present in PVFS version 2, whose first stable version was only recently released, and as such we did not have experimental data at the time the paper was prepared

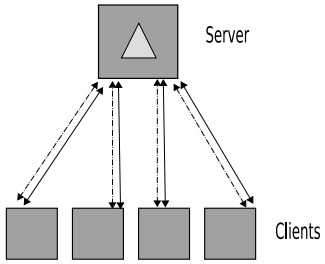

Figure 3. UNFS Communication Model

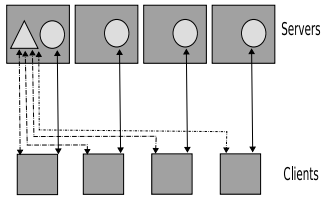

Figure 4. PVFS Communication Model

5.2. DAB Results

Figures 3 to 5 show the communication model for the three studied distributed file systems. The dashed lines show communication relative to file descriptor (metafiles in PVFS and dNFSP). The continuous lines show communication relative to data transfers. The triangles represent metaservers and the circles iods.

Figures 6 and 7 show the execution times for DAB using 8 and 16 clients respectively. The measured values for each system are grouped by phase for better comparison.

In phases 1 and 2, we can notice that the parallel file systems present high overhead when they execute operations like directories and file creation. In dNFSP, This overhead is mainly originated by the metafile replication mechanism, while PVFS has shown a poor performance in operation regarding metafiles.

The POVRay source code, used in phase 5 of DAB, has 1796 files and 74 directories. In the case of dNFSP, considering the test configuration with 16 clients, in

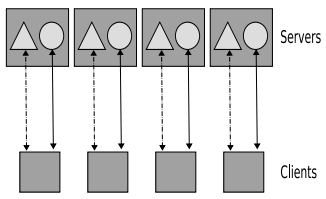

Figure 5. dNFSP Communication Model

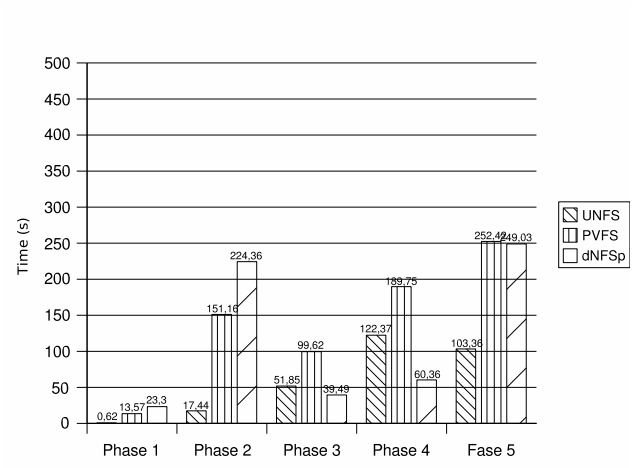

Figure 6. Modified Andrew Benchmark Test (8 clients)

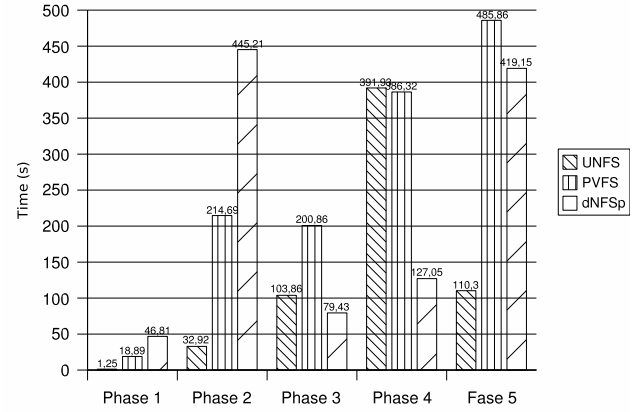

Figure 7. Modified Andrew Benchmark Test (16 clients)

the extreme situation when we are copying the source (phase 2), we realize a total of 89760 lookup operations in the metaservers to find the metafiles (that don't exist anywhere yet). This is because each metaserver searches for the metafile of each copied file on all other metaservers. The mechanism used to replicate the metafiles in the present version of dNFSP is relatively heavy and is one of the main aspects where we are working on. The values obtained in phase 5 reflect this behaviour.

According to Satyanarayanan [13], the reading operations are much more common than the writing operations, so the results of phases 3 and 4 represent considerable advantage to dNFSP. This advantage of multiple metaservers can be noticed clearly in phase 4. In this phase, the single NFS server is a serious bottleneck, which becomes more evident for a higher number of clients. PVFS, on the other hand, due to a problem with the handling of small files, presents poor performance.

In phase 5 we must consider that the compilation process consists actually in the alternation of the reading, compilation and writing. This means that the bottleneck of communication in the NFS server is not a major problem in this phase, because this alternation may result in a ad hoc synchronization over the NFS calls of the clients. Considering the current version of the parallel file systems and their inherent overhead to manage metafiles the parallel.

5.3. BTIO Results

This section presents results obtained using the BTIO benchmark to compare our file system with other related ones. BTIO, as described in section 4.2, is a variation of Computational Fluid Dynamics application where the intermediary results are written to disk during computation. Therefore, it requires a reactive and fast file system to achieve good performance.

As BTIO requires a perfect square number of process, the tests were executed using 1, 4, 9 and 16 clients. Figure 8 shows the average of the execution time obtained running the modified version of BTIO.

We can see that dNFSP was more effective in almost all the situations. It was between 0.4% slower and 33% faster than UNFS. Compared to PVFS the results were closer: our file system was between 1% and 6% faster than PVFS.

One reason for having a better performance compared to UNFS is the fact that dNFSP can be started using as many meta-servers and iods as needed, allowing to perform independent parallel writes. Another reason for better performance using dNFSP is the fact

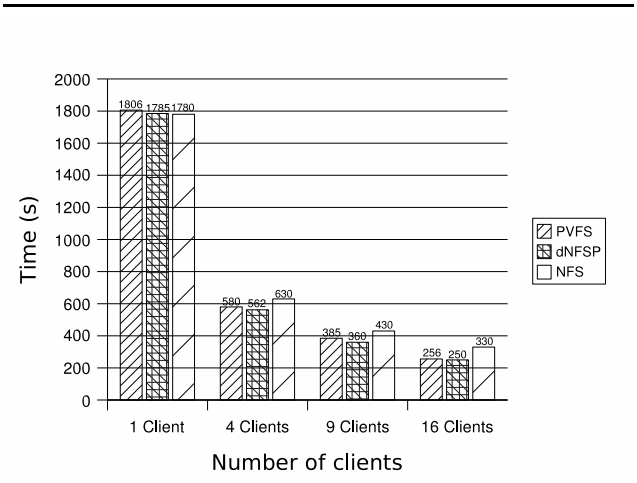

Figure 8. Comparison between UNFS, PVFS and dNFSP using BTIO-epio benchmark

that BTIO performs small write requests and PVFS has poor performance when used with small write block size, as stated by the authors on the PVFS website.

6. Conclusions and Future Work

Our experiments with dNFSP lead to the conclusion that the system is suitable for parallel applications on clusters, in the sense that an effective gain in read and write operations, comparable to those of a true parallel file system, can be observed. In our BTIO comparison, the levels of performance achieved with dNFSP are similar to those of PVFS, actually with a gain of up to 6% in execution time. Both systems perform clearly better than NFS with this benchmark, reaching 30% of advantage in some cases.

In the case of DAB, the performance of the three systems vary depending on each phase, with NFS sometimes showing better performance. This is due to the fact that DAB, being based on the Andrew Benchmark, mimics the load of a general-purpose file system, and not that of a parallel computing environment, especially by the frequent creation of new files and the handling of files of only a few kbytes. In dNFSP, the generation of lookup messages upon file creation is the main responsible for the decrease in performance, while for PVFS the problem lies on the handling of small data chunks.

Currently, in dNFSP, we are investigating a solution for the file lookup problem. We do not expect the file creation case to be very frequent, but the copying of remote meta-data is realistic, and may cause signif-

icant overhead if there are many files involved. A possible solution will be to copy several files (e.g. all the files on the same directory) on each update instead of just one, in order to try to anticipate future requests.

As a future work, we consider the possibility of introducing a level of fault tolerance on the iods, so that file striping may be performed redundantly (e.g. as in RAID). Another possibility is to port the implementation into the kernel-level NFS, since this version provides less overhead due to fewer memory copies, which also improves performance.

References

[1] R. B. Ávila, P. O. A. Navaux, P. Lombard, A. Lebre, and Y. Denneulin. Performance evaluation of a prototype distributed NFS server. In J.-L. Gaudiot, M. L. Pilla, P. O. A. Navaux, and S. W. Song, editors, *Proc. of the 16th Symposium on Computer Architecture and High Performance Computing*, pages 100–105, Foz do Iguaçu, Oct. 2004. Washington, IEEE.

[2] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, D. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga. The nas parallel benchmarks. *The International Journal of Supercomputer Applications*, 5(3):63–73, Fall 1991.

[3] B. Callaghan, B. Pawlowski, and P. Staubach. *NFS Version 3 Protocol Specification: RFC 1831.* Internet Engineering Task Force, Network Working Group, June 1995.

[4] P. H. Carns, W. B. Ligon III, R. B. Ross, and R. Thakur. PVFS: a parallel file system for Linux clusters. In *Proc. of the 4th Annual Linux Showcase and Conference*, pages 317–327, Atlanta, GA, 2000. Best Paper Award.

[5] Cluster File Systems, Inc. Lustre: A scalable, high-performance file system, 2002. Available at http://www.lustre.org/docs/whitepaper.pdf (July 2004).

[6] J. H. Howard, M. L. Kazar, S. G. Menees, D. A. Nichols, M. Satyanarayanan, R. N. Sidebotham, and M. J. West. Scale and performance in a distributed file system. *ACM Trans. Comput. Syst.*, 6(1):51–81, 1988.

[7] P. Keleher, A. L. Cox, and W. Zwaenepoel. Lazy release consistency for software distributed shared memory. In D. Abramson and J.-L. Gaudiot, editors, *Proc. of the 19th Annual International Symposium on Computer Architecture*, pages 13–21, Gold Coast, Queensland, Australia, 1992. New York, ACM Press.

[8] P. Lombard. *NFSP : Une Solution de Stockage Distribué pour Architectures Grande Échelle.* Thèse, Institut National Polytechnique de Grenoble, Grenoble, 2003.

[9] P. Lombard and Y. Denneulin. nfsp: a distributed NFS server for clusters of workstations. In *Proc. of the 16th International Parallel & Distributed Processing Symposium, IPDPS*, page 35, Ft. Lauderdale, Florida, USA, Apr. 2002. Los Alamitos, IEEE Computer Society. Abstract only, full paper available in CD-ROM.

[10] P. Lombard, Y. Denneulin, O. Valentin, and A. Lebre. Improving the performances of a distributed NFS implementation. In R. Wyrzykowski, J. Dongarra, M. Paprzycki, and J. Wasniewski, editors, *Proc. of the 5th International Conference on Parallel Processing and Applied Mathematics*, volume 3019 of *Lecture Notes in Computer Science*, pages 405–412, Czestochowa, Poland, 2003. Berlin, Springer.

[11] S. Microsystems. NFS: Network file system protocol specification, 1989.

[12] K. W. Preslan, A. P. Barry, J. E. Brassow, G. M. Erickson, E. Nygaard, C. J. Sabol, S. R. Soltis, D. C. Teigland, and M. T. O'Keefe. A 64-bit, shared disk file system for Linux. In *Proc. of the 16th IEEE Symposium on Mass Storage Systems*, pages 22–41, San Diego, California, Mar. 1999. Los Alamitos, IEEE Computer Society.

[13] M. Satyanarayanan. A study of file sizes and functional lifetimes. In *Proceedings of the eighth ACM symposium on Operating systems principles*, pages 96–108. ACM Press, 1981.

[14] F. Schmuck and R. Haskin. GPFS: A shared-disk file system for large computing clusters. In *Proc. of the Conference on File and Storage Technologies*, pages 231–244, Monterey, CA, 2002.

[15] S. Soltis, G. Erickson, K. Preslan, M. O'Keefe, and T. Ruwart. The design and performance of a shared disk file system for IRIX. In *Proc. of the 6th Goddard Conference on Mass Storage Systems and Technologies*, pages 41–56, College Park, Maryland, Mar. 1998.

[16] T. L. Sterling, J. Salmon, D. J. Becker, and D. F. Savarese. *How to Build a Beowulf: a Guide to the Implementation and Application of PC Clusters.* MIT, Cambridge, 1999.

Performance Evaluation of a Prototype Distributed NFS Server

Rafael B. Ávila*, Philippe O. A. Navaux
Instituto de Informática/UFRGS
Caixa Postal 15064
91501-970 Porto Alegre – Brazil
Email: {avila,navaux}@inf.ufrgs.br

Pierre Lombard, Adrien Lebre, Yves Denneulin
Laboratoire ID/IMAG
51, avenue Jean Kuntzmann
38330 Montbonnot-Saint Martin – France
Email: *First.Last*@imag.fr

Abstract

A high-performance file system is normally a key point for large cluster installations, where hundreds or even thousands of nodes frequently need to manage large volumes of data. While most solutions usually make use of dedicated hardware and/or specific distribution and replication protocols, the NFSP (NFS Parallel) project aims at improving performance within a standard NFS client/server system. In this paper we investigate the possibilities of a replication model for the NFS server which is based on Lasy Release Consistency (LRC). A prototype has been built upon the user-level NFSv2 server and a performance evaluation is carried out.

Keywords: *Parallel file systems, NFS, Lazy Release Consistency, parallel I/O.*

1. Introduction

With the development of new technologies and consequently lower prices of cluster components, clusters may be seen growing each time larger and larger in size and capacity, for solving each time more and more complex problems. This continuous growth in cluster sizes implied a new issue: how to manage permanent storage.

Given the potential bottleneck presented by traditional, centralized systems like NFS [5], and the fact that hardware-based solutions do not fit well in the Beowulf philosophy, the research in the field of cluster file systems has followed the direction of distributing the file service across the cluster.

In an attempt to achieve a balance between performance and management simplicity, the NFSP project [11] proposes a distributed version of the traditional NFS server which better handles the load generated by multiple concurrent accesses from the compute nodes, but still remains compatible with the NFS client present on every Linux system. In this paper, we investigate the possibilities of a NFSP branch in which the NFS server is replicated across several nodes, and as a consequence consistency needs to be maintained among the various clients. Our main goal with this work is to propose an alternative for medium-to-large cluster systems which do not benefit from expensive data storage systems and whose administrators wish to keep management tasks within the established common-knowledge.

Next section introduces the NFSP model in details in order to provide a background for the work being developed; in Section 3, we present the proposed replication model, and in Section 4 the results of a performance evaluation. Section 5 brings an overview of the related research activities and makes some observations in relation to our work. Finally, Section 6 presents some final considerations and future activities.

2. NFSP

NFSP — NFS Parallel — is an extension of the traditional NFS implementation, developed at the ID/IMAG Laboratory[1] of Grenoble, France. It distributes the functionality of the NFS daemon over several processes on the cluster [11]. The idea behind NFSP is to provide an improvement in performance and scalability and at the same time keep the system simple and fully compatible with standard NFS clients.

Inspired in PVFS [6], NFSP makes use of I/O daemons running on several machines in the cluster in order to distribute regular files across a set of disks, in a mechanism referred to as *striping*. A file is "striped" by having its data separated into several blocks, which are then stored over several distinct machines. When the file is accessed, there can be potentially several data blocks being fetched at the same time, consequently increasing performance. In the

* Work partially supported by CAPES and HP Brazil

1 http://www-id.imag.fr

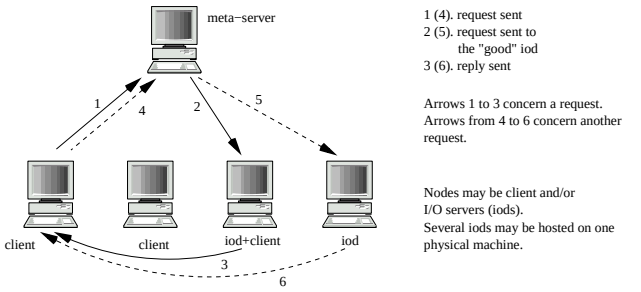

Figure 1: Request forwarding in NFSP

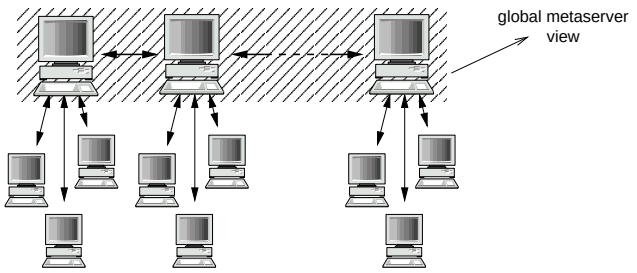

Figure 2: Distributed metaserver design

case of NFSP, the unit for striping is the NFS block (usually 8192 bytes).

The role of NFS server is played by *nfspd*, defined in NFSP as the *metaserver*. This daemon appears to clients as the regular NFS server; when a request for a given data block is received, the metaserver forwards the request to the corresponding I/O daemon, which in turn performs the operation and sends the desired information directly to the client[2]. Figure 1 illustrates this mechanism. The information regarding where and how each file is stored is kept on the metaserver's local file system, being part of the metadata (i.e. date/time, ownership, permissions, size, and the like).

This design of NFSP allows concurrent read operations to be effectively performed in parallel, given that distinct requests may correspond to distinct I/O daemons. However, write operations still need to be centralized at the metaserver, since the data come from the clients. This is one specific issue we are tackling with the model proposed in this paper, presented next.

3. The Server Replication Model

The goal of this work is to reach an improved level of performance in NFSP by replicating the metaserver. There are mainly two aims in this approach: offering several entry points for the clients (and hence let them use more bandwidth when doing write operations) and better balance the load onto several metaservers instead of only one.

3.1. Replicating nfspd

In the original NFS design, there is one single server (nfsd) to which all the client machines connect. An immediate approach to try to improve scalability in this scenario is to replicate the NFS server over several machines.

2 Techniques of IP spoofing may be necessary to achieve this goal of transparency from the clients' point of view, since some implementations require the answer to be originated from the server, not the I/O daemon.

The main idea is that a kind of grouping be established among the compute nodes, so that each metaserver instance serves only a given number of clients. For example, if the metaserver is replicated over 10 machines and there are 50 compute nodes, then each metaserver instance would be (considering a simple equal division of load) bound to 5 clients.

For each group of compute nodes accessing the same metaserver, the situation is similar to the original NFS model, i.e., all accesses to the file system are directed to one single server. Besides reducing the scalability problem, this approach allows a good deal of performance improvement if the connections to the metaservers are faster than that of the compute nodes, as often featured by Fast Ethernet switches that provide one extra Gigabit Ethernet port.

The metaservers run independently from but in cooperation with each other, by means of network communication, and together they form the notion of a single, global metaserver (see Figure 2). With that picture in mind, it is simple to observe that the goal here is to distribute the load of metadata operations, so that a single server does not become saturated in the face of multiple concurrent requests. In addition, this design may allow for future enhancements in terms of redundancy/fault tolerance.

3.2. Consistency Model

The direct implication of replicating the metaserver is to keep consistency between the multiple instances. In order to accomplish that task without incurring in too much overhead (e.g. increasing the network load with control messages), we rely upon a relaxed consistency model, supported by some characteristics that can be observed on a typical cluster computing environment.

The usual procedure for running a parallel program on a cluster is to allocate a given number of nodes, or partition, and then launch the application on it. Naturally, the size of such partitions and the allocation time vary depending on the application, but also according to the local policy of the site hosting the cluster, which depends on the users' status, priorities, previous executions, and the like. In addi-

tion, each user is normally using a private account, in which his files are stored, and thus the set of files being manipulated by an application is usually not the same as that of another application. This means that *we do not need to have all the files available everywhere all the time*.

As a coherence protocol for the distributed metaserver design, we make use of an adaptation of the Lazy Release Consistency protocol [8], or LRC, as used in the TreadMarks distributed shared memory system [2]. The basic principle is that the enforcement of coherence is postponed until the moment a client taking part in the protocol effectively needs it.

Similarly to TreadMarks, where consistency is checked only when a new client signals entry upon a shared segment (given that other clients may have modified it), our proposed system only checks for it when a client effectively accesses a file.

Coherence-checking messages are issued when some kind of error occurs, which is possibly an indication that the metadata regarding that specific file has changed. For example, if a file opening operation results in a `ENOENT` (non-existent file) error code, it is possible that the file has been created by a client bound to another metaserver, which means that the corresponding metadata only exists on that metaserver's local file system. In this case, the metadata needs to be copied to the current metaserver's file system, and then the operation (file opening) can proceed.

In some cases an operation can be executed even if the metadata is outdated. For example, if a client needs to read the first 4 kB of a file whose metadata indicate 10 kB of length, the operation can be successfully executed even if the file has already been enlarged (by some other client) to 100 kB. Notice that, even if the metadata is old, the file contents (i.e. the "real" data) are stored on the I/O daemons, which are shared among all the clients, and consequently have been updated by the operation that originally enlarged the file. In other words, even if the metadata is outdated, the client will not read stale data.

Naturally, some operations like file deletion need to be explicitly notified to all the metaservers in order to prevent access to data which is no longer valid. We understand, however, that this kind of operation occurs less frequently and mainly on application startup/shutdown, and thus should not cause significant impact on performance.

One important issue is to decide which metaserver to contact when metadata is missing/outdated. Several approaches can be thought of, like establishing a neighbourhood relationship between the metaservers and perform the search from the nearest to the farthest. Another possibility is to organize the metaservers hierarchically in a tree. This is one of our current study subjects.

4. Performance Evaluation

In order to validate the distribution model, we have implemented a prototype of the replicated metaserver based on the user-level implementation of NFSP. The main intention was to allow an evaluation of the "cache-miss" overhead imposed by the eventual need of fetching metadata (actually a metafile), from another metaserver, which is the basis of our model. Additionally, we wanted to observe the level of performance gain obtained by distributing the metaserver load among several replicated instances.

The original user-level NFSP implementation has been extended in order both to assign an address to each metaserver and to introduce a first level of communication between them. To keep the changes simple, we rely upon `rcp` to transfer metafiles between metaservers. This ensures not only that the contents of the metafile be transferred, but also the implicit metadata (owner, group, permissions, etc.) associated with it. On the other hand, we are aware that this mechanism is likely to impose a higher overhead than that of a dedicated protocol. Finally, only the case of missing metadata is being evaluated.

4.1. Cache-miss Overhead

The first evaluation on the implemented prototype refers to measuring the overhead of a cache-miss on a given metaserver and consequent metafile searching and fetching.

The metafile fetching mechanism must be based upon a decision algorithm which should indicate where to search. In our evaluation, in order to obtain the worst-case results, we forced each metaserver to perform a sequential search, i.e. starting from metaserver #0, then metaserver #1, and so on, up to metaserver $\#(n-1)$ if necessary.

The benchmark we have run consists on creating 100 files on one of the client nodes, and then forcing the other nodes on reading all the files. As we want to measure the worst-case situation, the created files have 0-byte length (i.e., we want to measure metadata overhead only).

The execution has been carried out on the *i-cluster*[3] available at the ID Laboratory. We have configured the system with 1 iod (which is enough for this test), 16 metaservers and 16 clients (one for each metaserver).

The files for the benchmark have been created on the client mounting metaserver #15. Then, starting at client #14 and going down to client #0, we measured the time each node takes to read the 100 files (here *read* means `cat file > /dev/null`). Only one node executes at a time.

3 A 225-node cluster deployed within the context of a joint ID/INRIA/HP project; the nodes are Pentium III 733 MHz with 15 GB IDE disks connected by Fast Ethernet; all the nodes run Mandrake GNU/Linux with kernels 2.2 and 2.4

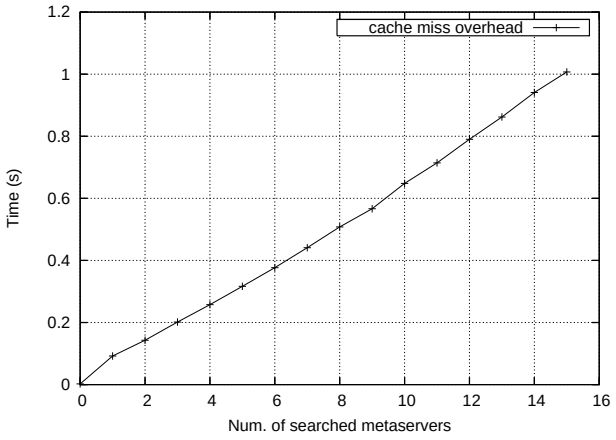

Figure 3: Cache-miss overhead according to the number of metaservers searched

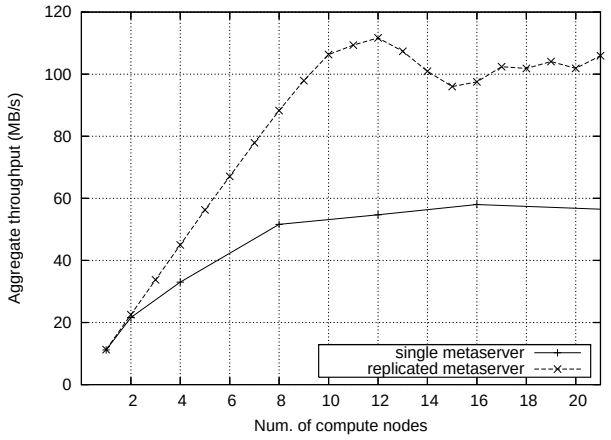

Figure 4: Aggregate read bandwidth obtained with the prototype, in comparison to the original model

Since the search always begins by metaserver #0, the first metaserver (#14) will try 15 times before finding the files on metaserver #15. The second metaserver to run will try one time less, since now the metafiles are available at #14. And so on, up to client/metaserver #0.

The obtained results are shown in Figure 3. Each result has been divided by 100 in order to represent the per-file overhead. We can observe that the curve grows in a practically linear slope, which corresponds to about 66 ms for each additional cache miss. This results in a worst-case delay of roughly 1 second in the case of 16 metaservers, which we consider a reasonable overhead to bear with. For example, considering the case where an application spans three metaservers, the maximum overhead to pay in this case would be of about 200 ms. In addition, the simplicity of using a simple `rcp` to fetch the remote metafile incurs in extra overhead, which we can expect to be reduced if a dedicated protocol is used.

4.2. Read Performance

We have also run performance evaluation experiments with the implemented prototype, to observe the level of gain that can be obtained in relation to the previous NFSP implementations. In the first case, we are evaluating distributed read performance. The experiment consists on reading a large file concurrently on all the clients. For that purpose we have created a 1 Gigabyte file on one of the nodes, and then concurrently launched the command `dd if=file of=/dev/null bs=1M count=1024` on the nodes (we did not force a metafile update on all the nodes prior to running the experiment, since the time is takes to read the data is much higher than that of the measured overhead).

The system, in this case, has been configured with 12 iods, 7 metaservers and 21 clients (which makes 3 clients per metaserver), in a total of 40 nodes.

The results shown in Figure 4 compare the aggregate read bandwidth obtained with the replicated metaserver model with that of the original model. As expected, having multiple metaserver instances distributes the load imposed by the concurrent accesses, and thus allows for a higher level of performance. In a previous work [11], we had observed that the performance was limited by the CPU on the metaserver saturating. In the replicated model we can observe that the performance reaches its maximum when the number of clients equals the number of iods, which was also expected.

As an alternative form of evaluation, we can consider a *per-client efficiency* measure in relation to the maximum network bandwidth achievable by each client. Considering 11 MB/s for each one (given the Fast Ethernet connection), we obtain practically 100% efficiency up to 10 clients, when then the curve flattens and tends to stabilize at around 110 MB/s, equivalent to 50% efficiency with 21 clients. As a simple comparison, a regular, centralized NFS server would reach only 4% with the same 21 clients, since the bandwidth would be always limited by the 11 MB/s network connection.

4.3. Write Performance

A similar experiment was also executed, but now with the clients writing a 1 Gigabyte file (each client writes a distinct file).

A comparison with the original NFSP implementation is not meaningful in this case, since writing is, as we stated before, centralized as that of the traditional NFS model.

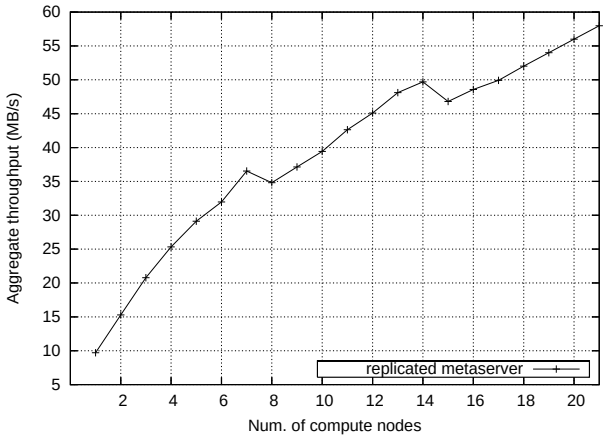

Figure 5: Aggregate write bandwidth obtained with the prototype

Measured results are presented in Figure 5. The execution has been carried out so that the maximum number of metaservers is always used (i.e. in the case of 7 clients, for example, each client connects to one distinct metaserver). The curve presents three distinct linear segments, separated by a "step" in performance every time the number of clients reaches a multiple of the number of metaservers. This is due to a single metaserver having to suddenly serve a higher number of clients than the others. For example, performance increases constantly up to 7 clients, when each metaserver is connected to at most one client. When we add the eighth client, metaserver #0 starts serving two clients, who share that server's bandwidth and consequently performances drops.

Again, we can observe the advantage of having multiple entry points for the clients, and specially in the case of writing. With one single metaserver, write bandwidth is primarily limited by the single network connection, and would remain constant at the lowest value shown (around 10 to 11 MB/s) for any number of clients. If we consider the per-client efficiency, the replicated model presents a mean value of about 33% with the configuration used, in contrast to 9% achievable with a centralized server.

5. Related Work

Several research projects exist which aim at a better performance for cluster file systems, using different approaches. Petal/Frangipani [9, 10] is a parallel file system built upon the concept of a *distributed virtual disk*, where a set of daemons running on a number of machines cooperate to form the view of a single storage device. The *Shared Logical Disk* [13] follows the same idea. Another approach is that of *Storage Area Networks*, in which there is a dedicated hardware support for parallel access to permanent storage. These are mostly commercial systems; examples are the IBM General Parallel File System [7], SGI XFS [15], and the OpenGFS (Global File System) [1].

It is undeniable that these systems are of recognized importance, however we do not consider that a comparison would be appropriate, since they follow different approaches and present distinct requirements. In the sequence, we present systems more directly related to the model we propose.

NFSP has its basis on the Network File System [5], or NFS, which is the *de facto* standard for distributed file sharing in the Unix world, and consequently has been naturally absorbed by the Beowulf cluster model since its first steps [14]. It is actually a protocol for transparent remote access from a client to a server's file system, and thus has not been devised for parallel computing. NFS raises a potential scalability constraint when clusters start to grow larger, due to its centralized server design. For this reason, many developments are currently being carried out in both the directions of enhancing NFS and providing new solutions [4, 12].

The *Berkeley xFS* [3] was a prototype "serverless" file system developed at the University of California at Berkeley from 1993 to 1995. It builds upon several research efforts developed at the time, like RAID, LFS (Log-structured File System), Zebra and Multiprocessor Cache Consistency, and thus presents a totally distributed design, where data and metadata are spread among the available machines (which may be all or part of the available computing resources) and can dynamically migrate. Though an interesting design and promising results, the project was interrupted with the end of project NOW, and hence, to our knowledge, no further development or porting to Linux has been carried out to the present days.

A possible successor of xFS in the Beowulf world is PVFS, *Parallel Virtual File System* [6], a joint project conducted by the Parallel Architecture Research Laboratory, at Clemson University, and the Argonne National Laboratory, both in the USA. The goal of PVFS is to provide a high-performance file system for the Beowulf class of parallel machines, being able to profit from commodity hardware. PVFS is able to deliver very good performance and provides different interfaces for applications: VFS, MPI-IO and a native one.

NFSP compares with such projects in the sense that it aims at performance and scalability for cluster computing. A different approach taken by NFSP, however, is that of keeping the changes to a traditional Beowulf system as minimal as possible. For example, the client side on a NFSP installation remains untouched, which eases the task of managing the cluster. Even though we are aware that a NFS-compatible file system may not be the best solution for certain applications, we also believe that the NFS approach

might still be enough for many situations, and in such cases a simple solution close to the traditional Beowulf environment might be desirable.

6. Final Considerations and Future Work

Our intention with the replicated metaserver model for NFSP is to provide an additional level of performance while still keeping the simple design that guides the project. With the implementation of a prototype based on the user-level version of NFSP, we could observe an effective gain in performance in relation to a single, centralized NFS server, and with other "flavors" of NFSP. With minimum effort, we are able to improve by about 5 times the write performance and to double the read performance (in this case reaching almost 100% network efficiency), in comparison with the previous implementation.

Our evaluation of the adopted replication model is positive. The consistency mechanism based on LRC is efficient for applications running on a typical cluster environment, and the overhead imposed by eventual cache misses was considered low (around 0.066 seconds), even if a simple and heavier `rcp` mechanism was used.

Current and future activities in NFSP follow different approaches. As mentioned before, we are investigating different methods such as closest-neighbour and hierarchy for finding the correct metaserver in the case of metafile cache misses. Another activity concerns the integration of NFSP into the kernel-level NFS server, which, according to results already obtained, offers a significant gain in performance in relation to the user-level version. Concerning the metaserver replication, our next steps will be the design and implementation of a dedicated protocol for communication between the multiple servers, and further performance evaluations with real applications.

Acknowledgements

The authors would like to thank Caciano Machado and Rodrigo Kassick for their help in setting up the execution environment and running the experiments.

References

[1] The openGFS project, Apr. 2003. http://opengfs.sourceforge.net.

[2] C. Amza, A. L. Cox, S. Dwarkadas, P. Keleher, H. Lu, R. Rajamony, W. Yu, and W. Zwaenepoel. Treadmarks: Shared memory computing on networks of workstations. *IEEE Computer*, 29(2):18–28, Feb. 1996.

[3] T. E. Anderson, M. D. Dahlin, J. M. Neefe, D. A. Patterson, D. S. Roselli, and R. Y. Wang. Serverless network file systems. In *Proc. of the 15th Symposium on Operating Systems Principles*, pages 109–126, Copper Mountain Resort, Colorado, Dec. 1995. ACM.

[4] A. Calderón, F. García, J. Carretero, J. M. Pérez, and J. Fernández. An Implementation of MPI-IO on Expand: A Parallel File System Based on NFS Servers. In *9th PVM/MPI European User's Group*, September 2002.

[5] B. Callaghan, B. Pawlowski, and P. Staubach. *NFS Version 3 Protocol Specification: RFC 1831*. Internet Engineering Task Force, Network Working Group, June 1995.

[6] P. H. Carns, W. B. Ligon III, R. B. Ross, and R. Thakur. PVFS: a parallel file system for Linux clusters. In *Proc. of the 4th Annual Linux Showcase and Conference*, pages 317–327, Atlanta, GA, 2000. Best Paper Award.

[7] P. F. Corbett et al. Parallel file systems for the IBM SP computers. *IBM Systems Journal*, 34(2):222–248, Jan. 1995.

[8] P. Keleher, A. L. Cox, and W. Zwaenepoel. Lazy release consistency for software distributed shared memory. In D. Abramson and J.-L. Gaudiot, editors, *Proc. of the 19th Annual International Symposium on Computer Architecture*, pages 13–21, Gold Coast, Queensland, Australia, 1992. New York, ACM Press.

[9] E. K. Lee and C. A. Thekkath. Petal: Distributed virtual disks. In *Proc. of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 84–92, Cambridge, MA, 1996.

[10] E. K. Lee, C. A. Thekkath, C. Whitaker, and J. Hogg. A comparison of two distributed disk systems. Technical Report 155, Systems Research Center, Digital Equipment Corporation, Apr. 1998.

[11] P. Lombard and Y. Denneulin. nfsp: a distributed NFS server for clusters of workstations. In *Proc. of the 16th International Parallel & Distributed Processing Symposium, IPDPS*, page 35, Ft. Lauderdale, Florida, USA, Apr. 2002. Los Alamitos, IEEE Computer Society. Abstract only, full paper available in CD-ROM.

[12] D. Muntz. Building a Single Distributed File System from Many NFS Servers. Technical Report HPL-2001-176, 2001.

[13] R. A. Shillner and E. W. Felten. Simplifying distributed file systems using a shared logical disk. Technical Report TR-524-96, Dept. of Computer Science, Princeton University, Princeton, NJ, 1996.

[14] T. L. Sterling, J. Salmon, D. J. Becker, and D. F. Savarese. *How to Build a Beowulf: a Guide to the Implementation and Application of PC Clusters*. MIT, Cambridge, 1999.

[15] A. Sweeney, D. Doucette, W. Hu, C. Anderson, M. Nishimoto, and G. Peck. Scalability in the XFS file system. In *Proceedings of the USENIX 1996 Technical Conference*, pages 1–14, San Diego, CA, USA, Jan. 1996.

A Comparison on Current Distributed File Systems for Beowulf Clusters

Rafael Bohrer Ávila[1]
Philippe Olivier Alexandre Navaux[2]
Yves Denneulin[3]

Abstract

This paper presents a comparison on current file systems with optimised performance dedicated to cluster computing. It presents the main features of three of such systems — xFS, PVFS and NFSP — and establishes comparisons between them, in relation to standard NFS, in terms of performance, fault tolerance and adequacy to the Beowulf model.

1 Introduction

With the popularisation of Beowulf-class parallel machines (STERLING, 2002) and the constant improvements in technologies for computer components, it is becoming more and more common the deployment of clusters with hundreds or even thousands of nodes. For such large-size systems, the performance of the file system becomes critical. Traditional systems such as NFS (CALLAGHAN; PAWLOWSKI; STAUBACH, 1995) cannot be used directly since the nature of its centralised server becomes a critical bottleneck for the whole cluster. In other words, scalability is an important issue for file systems of large clusters. In this way, many alternatives to greater scalability, in several directions, are currently being pursued. In this paper we present and compare some of the many current file systems which present extended and/or modified features to accomplish the needed scalability for use with large clusters.

2 File Systems for Beowulf Clusters

The **Network File System** (CALLAGHAN; PAWLOWSKI; STAUBACH, 1995), or NFS, is the *de facto* standard for distributed file sharing in the Unix world, and consequently has been naturally absorbed by the Beowulf cluster model since its first steps (STERLING, 1999).

[1] avila@inf.ufrgs.br Bolsista CAPES
[2] navaux@inf.ufrgs.br
[3] Yves.Denneulin@imag.fr

NFS was developed by Sun Microsystems and first came out in 1985 with the release of SunOS 2.0. It is in fact a protocol for transparent remote access from a client to a server's file system. NFS is designed to be a stateless protocol[4], which means that a crash from one of the clients does not affect the server, and server crashes can be easily and transparently recovered when the server comes back.

The **Berkeley xFS**[5] (ANDERSON, 1995) was a prototype "serverless" file system developed at the University of California at Berkeley from 1993 to 1995. It builds upon several research efforts developed at the time, mainly on RAID, LFS (Log-structured File System), Zebra and Multiprocessor Cache Consistency (all described on the same article from 1995). xFS presents a totally distributed design, where data and control information, or *meta-data*, are spread among the available machines (which may be all or part of the available computing resources) and can dynamically migrate.

A prototype of xFS has been built and tested on a 32-node SPARCStation cluster at Berkeley. Despite the very promising results, however, the project stopped shortly after, and hence, to our knowledge, no further development or porting to Linux has been carried out to the present days.

PVFS, *Parallel Virtual File System* (CARNS, 2000), is a joint project conducted by the Parallel Architecture Research Laboratory, at Clemson University, and the Argonne National Laboratory, both in the USA. The goal of PVFS is to provide a high-performance file system for the Beowulf class of parallel machines, being able to profit from commodity hardware.

The system can be viewed as a simplified version of xFS. Access to permanent storage is performed by *I/O daemons*, and a single, centralised *manager* is responsible for manipulating meta-data. Clients access the file system by means of the PVFS API, which has also been ported to a Linux VFS kernel module (i.e. it can be `mount`ed) and to MPI-IO.

NFSP — NFS Parallel — is an extension of the traditional NFS implementation, developed at Laboratory ID/IMAG of Grenoble, that distributes the functionality of the NFS daemon over several processes on the cluster (LOMBARD; DENNEULIN, 2002). The idea behind NFSP is to provide an improvement in performance and scalability and at the same time keep the system simple and fully compatible with standard NFS clients.

Similarly to PVFS, NFSP makes use of I/O daemons to access data on the disks. The role of the NFS server is played by the *nfspd*, defined by the authors as a *meta-server*. This daemon appears to clients as the regular nfsd server; when a request for a given piece of data is received, the meta-server forwards the request to the corresponding I/O daemon, which in turn performs the operation and sends the desired information directly to the client.

Other Approaches — Several other research projects aim at a better performance for cluster file systems, using different approaches. Petal/Frangipani (LEE, 1998) is a parallel file system built upon the concept of a *distributed virtual disk*, where a set of daemons running on a number of machines cooperate to form the view of a single storage device. The *Shared*

[4] The new NFS Version 4 will be stateful

[5] Not to be confused with SGI's XFS, with a capital "X"

Table 1: Overall performance comparison for the analysed systems (bandwidths are presented as an improvement ratio over that of regular NFS)

Item	**xFS**	**PVFS**	**NFSP**
I/O servers	32	24	16
Read bandwidth	11.04	17.76	4.0
Write bandwidth	11.12	18.08	–
Read efficiency	34.75%	74%	25%
Write efficiency	34.5%	75.3%	–

Logical Disk (SHILLNER; FELTEN, 1996) follows the same idea. Another approach is that of *Storage Area Networks*, in which there is a dedicated hardware support for parallel access to permanent storage. Examples are the IBM General Parallel File System (CORBETT et al., 1995), SGI XFS (SWEENEY, 1996), and the OpenGFS (Global File System) (THE..., 2003). Though these systems are of recognised importance, we will concentrate the following analysis on the previous ones, since we are targeted at Beowulf class systems, in which the use of commodity software and components is strongly desired.

3 Analysis of the Presented Systems

Overall Performance This comparison is based on performance measurements, in terms of maximum achievable bandwidth, presented by the authors of each system on the indicated references. It is difficult to compare them directly since the results have been obtained on different test-beds. Therefore, we choose to present, for each system, the results relative to the reported regular NFS performance on the same environment. We also present the efficiency of each system in relation to the number of I/O servers used.

Table 1 summarises the relative performances of each system. We have separated write and read performances since NFSP is not at present optimised for distributed writing, and thus a comparison would be unfair. It is also important to state that the results presented for xFS have been obtained from an early prototype, admittedly reported by the authors as not optimised. The system which presents best overall performance is PVFS, reflecting a very good efficiency in using the available I/O servers and thus reaching improved bandwidth. For both xFS and PVFS, read and write performances are similar, since they use a symmetric approach for both operations, to the difference of NFSP.

Scalability In this item we are evaluating the capacity of each system in increasing performance as the number of clients and I/O servers increase.

NFS scalability, as already exposed, is deficient for large clusters. In fact, this deficiency

depends mostly on the interconnection technology being used. Current off-the-shelf hard disks deliver bandwidths in the range of 50–150 MB/s; this means that access to the NFS server is likely to be limited in about 11 MB/s by an ordinary Fast Ethernet network card even before the disk's full capacity may be reached. In other words, even a small 16-node cluster can be easily affected by the problem.

As a consequence, all of the three systems presented are able to overcome standard NFS performance very quickly, even for a few client nodes; however, they differ in how the performance increase scales. xFS presents good scalability, especially up to about 8 client nodes, when the performance increase is practically linear (around 1 MB/s for 1 client, 7.5 MB/s for 8). NFSP scales linearly up to 6 clients and then stabilises, roughly at 50 MB/s. PVFS, once more, shows very good results: the system has presented practical linear scaling up to 24 client nodes and I/O servers, when 226 MB/s can be achieved.

The limits reached by the three systems reflect the characteristic mentioned before, that performance is limited by network bandwidth rather than that of disk. The last two systems present results obtained using Fast Ethernet as interconnect, where a maximum bandwidth of 9–11 MB/s is usually achieved; as a consequence, 6 NFSP clients are limited at 50 MB/s (8.33 MB/s per client), and 24 PVFS clients are limited at 226 MB/s (9.41 MB/s per client). It also explains why PVFS does not scale beyond this limit: even though up to 30 nodes have been used, the number of I/O servers was kept at 24. In the case of NFSP, the authors suspect that the limiting in performance happens because of the meta-server's CPU saturating; they expect the system to deliver better performance after some optimisation in the code.

These results suggest that the network design in a cluster file system should be carefully studied. Techniques such as channel bonding (using two network cards as one), whose support is readily available in the Linux kernel, or an expected popularisation of Gigabit Ethernet may contribute to that.

Fault Tolerance This feature can be viewed in two levels: temporary service unavailability and crash failure. NFS supports the first, but not the second. Up to version 3, NFS is a stateless protocol, which means that temporary server unavailability is tolerated by the clients, but a crash is fatal. This adapts well to a real scenario, where hardware failures are less frequent but eventual stops/restarts may occur (e.g. when upgrading software "on the fly").

When discussion comes to distributed servers, fault tolerance becomes more difficult do realise or imposes significant overhead, and as such it is frequently left aside. xFS makes use of RAID to provide fault tolerance, as well as for improved performance. Storage servers are divided in groups, and in each group one server is dedicated for parity. Thus, failure in one of the servers (even permanent) can be coped with. NFSP benefits from the same stateless model of NFS, in the sense that temporary failures can be tolerated. The current implementation does not provide a mechanism for tolerating failures in the I/O nodes, but the authors do mention redundancy as a future improvement, which may introduce a level of error recovery capability. The PVFS design does not include support for fault tolerance; this

seems to be planned for version 2 of the system.

Integration with the Beowulf model For integration with the Beowulf model, we consider characteristics such as availability of the software, no requirement for a specific technology, and licensing of the whole system as open source. We also evaluate the complexity of integrating the system in a regular Linux cluster.

Except for xFS, all the other systems fully comply with the first requirements. The former was only implemented for an early version of Solaris, and is to our knowledge not yet available for Linux. NFS is traditionally included on every current Linux distribution. Its use is straightforward, requiring single file configuration on both server and client side. NFSP presents an important feature in this sense, allowing for the client side to remain untouched. Configuration on the server side is also not too complex, being similar to that of NFS.

PVFS represents the most "intrusive" solution, since it requires dedicated configuration on both sides. While the client side does not differ much from NFS, requiring only a kernel VFS (Virtual File System) module to be compiled, the server side does demand further configuration.

4 Final Considerations

Many developments, in several directions, can be observed today concerning high performance file systems for clusters, given that NFS represents a potential bottleneck. In this paper we have concentrated on some of the systems more directly targeted at the Beowulf class of parallel machines, in the sense that no specific hardware or communication technology be required for their proper utilisation. Available commercial file systems for high-performance computing often exhibit that requirement.

Table 2 summarises the analysed features of each system in a simplified form. The Berkeley xFS seems a promising design, since very good results have been obtained, even with an unoptimised prototype; the system, however, has not been further developed or ported to Linux, rendering itself currently not eligible as a solution for Beowulf clusters. PVFS is currently becoming a kind of *de facto* standard in the Beowulf world. The system presents very good performance and scalability, and supports several APIs including a Linux VFS module, with a slight complexity in management. Version 2 is currently being developed, and will include features for grid computing and fault tolerance. NFSP is also in an early stage of development, but presents encouraging results, besides being compatible with the standard NFS client. A design alternative allowing for concurrent write operations would also be desirable.

References

ANDERSON, T. E. et al. Serverless network file systems. In: ACM. *Proc. of the 15th Sym-*

Table 2: Summary of the analysed features

Item	**NFS**	**xFS**	**PVFS**	**NFSP**
Performance	regular	good	very good	good
Scalability	poor	good	very good	regular
Fault tolerance	temporary	crash	none	temporary
Beowulf integration	very easy	none	not trivial	easy

posium on Operating Systems Principles. Copper Mountain Resort, Colorado, 1995. p. 109–126.

CALLAGHAN, B.; PAWLOWSKI, B.; STAUBACH, P. *NFS Version 3 Protocol Specification: RFC 1831*. [S.l.], jun. 1995.

CARNS, P. H. et al. PVFS: a parallel file system for Linux clusters. In: *Proc. of the 4th Annual Linux Showcase and Conference*. Atlanta, GA: [s.n.], 2000. p. 317–327. Best Paper Award.

CORBETT, P. F. et al. Parallel file systems for the IBM SP computers. *IBM Systems Journal*, v. 34, n. 2, p. 222–248, jan. 1995.

LEE, E. K. et al. *A Comparison of Two Distributed Disk Systems*. [S.l.], abr. 1998.

LOMBARD, P.; DENNEULIN, Y. nfsp: a distributed NFS server for clusters of workstations. In: *Proc. of the 16th International Parallel & Distributed Processing Symposium, IPDPS*. Ft. Lauderdale, Florida, USA: Los Alamitos, IEEE Computer Society, 2002. p. 35. Abstract only, full paper available in CD-ROM.

SHILLNER, R. A.; FELTEN, E. W. *Simplifying Distributed File Systems Using a Shared Logical Disk*. Princeton, NJ, 1996.

STERLING, T. L. *Beowulf Cluster Computing with Linux*. Cambridge: MIT Press, 2002.

STERLING, T. L. et al. *How to Build a Beowulf: a Guide to the Implementation and Application of PC Clusters*. Cambridge: MIT, 1999. 239 p.

SWEENEY, A. et al. Scalability in the XFS file system. In: *Proceedings of the USENIX 1996 Technical Conference*. San Diego, CA, USA: [s.n.], 1996. p. 1–14. Disponível em: <citeseer.nj.nec.com/sweeney96scalability.html>.

THE OpenGFS Project. abr. 2003. Disponível em: <http://opengfs.sourceforge.net>. Access em: abril 2003.

An Analysis on the Scalability of the dNFSp File System

Everton Hermann[1]* Rafael Ávila[1,2]† Philippe Navaux[1] Yves Denneulin[2]

[1]Instituto de Informática/UFRGS
Caixa Postal 15064
91501-970 Porto Alegre – Brazil
Email: {ehermann,avila,navaux}@inf.ufrgs.br

[2]Laboratoire ID/IMAG
51, avenue Jean Kuntzmann
38330 Montbonnot-Saint Martin – France
Email: *First.Last*@imag.fr

Abstract

The leveraging of existing storage space in a cluster is a desirable characteristic of a parallel file system. While undoubtedly an advantage from the point of view of resource management, this possibility may face the administrator with a wide variety of alternatives for configuring the file server, whose optimal layout is not always easy to devise. Given the diversity of parameters such as the number of processors on each node and the capacity and topology of the network, decisions regarding the placement of server components like metadata servers and I/O servers have a direct impact on performance and scalability. In this paper, we explore the capabilities of the dNFSp file system on a large cluster installation, observing how scalable the system behaves in different scenarios and comparing it to a dedicated parallel file system. Our obtained results show that the design of dNFSp allows for a scalable and resource-saving configuration for clusters with a large number of nodes.

Keywords: *Parallel file system, scalability, performance, NFS, parallel I/O*

1. Introduction

Solutions for efficient management of I/O in large clusters have long been the focus of several research groups and industrials working on parallel computing [12]. Ranging from RAID arrays and fibre optics to virtual distributed disks, many approaches have been proposed in the last decade that vary considerably in terms of performance, scalability and cost.

In previous works [8, 2], we have presented the *dNFSp* file system, an extension of NFSv2 that aims at improving both performance and scalability of a regular NFS server while keeping its standard administration procedures and, mainly, compatibility with the regular NFS clients available on every Unix system. Similarly to other parallel file systems such as PVFS [5] and Lustre [6], dNFSp is based on a distributed approach where the gain in performance is obtained by executing tasks in parallel over several machines of the cluster.

One important aspect of a parallel file system is its capability of leveraging existing resources. In the case of commodity clusters, the hard disks that are installed on the compute nodes are frequently under-used: a typical GNU/Linux node installation takes only a few gigabytes, and today's PCs are hardly ever configured with less than 40 gigabytes of storage. This leaves us with at least 75% of the total hard disk capacity available for the storage of data, and consequently it is important that a cluster file system have the ability to use it.

dNFSp provides such a feature, so that the storage on the compute nodes can be used to form a single cluster file system. It is then up to the cluster administrator to decide how to configure the system, finding a good balance between resource utilization, performance and scalability, which might not be an obvious task.

For this reason, we have conducted a series of experiments varying the configuration of dNFSp on a large cluster and watching how scalable the system behaves, trying to identify layouts best suited for one or another situation, and whose results and conclusions are presented in this work.

In the remainder of the paper, Section 2 presents the dNFSp file system and its main characteristics, with the purpose of providing some background knowledge; Section 3 describes in more details the experiments we have conducted and introduces the evaluation criteria; in Section 4 we present the results obtained in the experiments and provide the discussion which is the focus of this work; Section 5 brings a comparison of our work to systems with related objectives, and finally Section 6 draws some conclusions on the obtained results and analysis and reveals future directions.

* Work supported by CAPES
† Work supported by HP Brazil

2. dNFSp – A Distributed NFS Server

The NFSp project [11] has been established in 2000 at the *Laboratoire Informatique et Distribution* of Grenoble, France, with the goal of improving performance and scalability in a regular NFS installation. The main idea of the project is to provide a cluster file system that benefits from the standard administration procedures and behavior of a well-known protocol such as NFS. As a result, NFSp — for *parallel* NFS — presents some simple extensions to the NFS server implementation that distributes its funcionalities over a set of nodes in the cluster, thus gaining performance. On the other hand, the client machines do not have to be modified at all, favoring portability.

As a subproject within the NFSp group, dNFSp has been proposed as a further extension to the model, aiming at an improvement on concurrent write operations by client machines.

Figure 1 depicts the distribution model proposed by dNFSp. On the top of the figure, the so-called *meta-servers* are daemons that play the role of the NFS server, and cooperate with each other to form the notion of a single file server. Working along with the meta-servers, a number of I/O daemons, or *IODs*, are responsible for data storage and retrieval. On the bottom, client machines connect to the meta-servers in the same way that a client connects to a regular NFS server.

Each client is connected to one metaserver that is responsible to handle its operations. The operations involving only metadatas are replied directly by the metaserver, which in some cases can contact other metaservers to obtain the needed metadata. I/O operations are forwarded by the metaserver to the IODs, which will perform the operation and reply directly to the client. When performing read operations the file contents are transferred between client and IOD, allowing parallel reads. However, to execute a write operation, the data are transferred between the client and the metaserver, as it is the only access point to the file system the client has. The data received by the metaserver is forwarded to the IODsr. Therefore, writes performance depends not only on the amount of IODs but also on the level of parallelism that can be achieved using multiples metaservers. That is why dNFSp has a better write performance when compared to the original NFSp.

The metaservers exchange information to keep the coherence among file systems views across all metaservers. The information is retrieved only when it's requested by a client, avoiding unnecessary network traffic. However, there are situations when all metaservers must be contacted (e.g. file creation), and this communication becomes more visible as we increase the amount of metaservers.

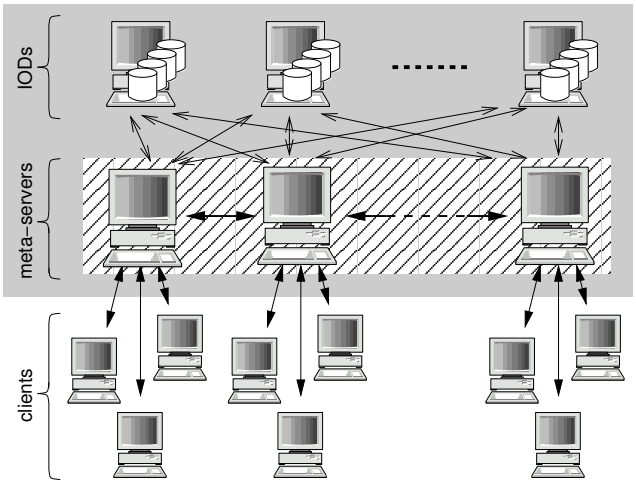

Figure 1. Distributed meta-server architecture of dNFSp

3. Measuring dNFSp Scalability

The measurement goals of this paper are to evaluate the scalability of dNFSp in a large number of nodes using a real application-based benchmark. The benchmark application we used is the *NAS/BTIO*, and the machine used to run the applications is the *INRIA i-cluster2*. Both the application and the cluster are detailed in the following sections.

3.1. The NAS/BTIO Benchmark

The BTIO Benchmark is a part of the NAS Parallel Benchmarks (NPB) [17]. It is the responsible to evaluate the storage performance of parallel and distributed computer systems. The application used by BTIO is an extension of the BT benchmark [3]. The BT benchmark is based on a Computational Fluid Dynamics (CFD) code that uses an implicit algorithm to solve the 3D compressible Navier-Stokes equations.

BTIO uses the same computational method employed by BT. The I/O operations were added by forcing the writing of result to disk. In BTIO the results must be written to disk at every fifth step of BT.

The number of process running one execution of the BTIO must be a perfect square (1, 4, 9, 16, ...). The problem size is chosen by specifying a class. Each class represents the cubic matrix dimensions: class A (64^3), class B (102^3), class C (162^3). We used class A since it was enough to stress the underlying file systems.

Another customization of BTIO is the way the data are stored in the file system. There are four flavors that can be chosen at compilation time:

- BTIO-full-mpiio: uses MPI-IO file operations with *collective buffering*, which means that data blocks are potentially re-ordered previously to being written to disk, resulting in coarser write granularity
- BTIO-simple-mpiio: Also uses MPI-IO operations, but no data re-ordering is performed, resulting in a high number of seeks when storing information on the file system
- BTIO-fortran-direct: This version is similar to simple-mpiio, but uses the Fortran direct access method instead of MPI-IO
- BTIO-epio: In this version each node writes in a separate file. This test gives the optimal write performance that can be obtained, because the file is not shared by all the processes, so there is no lock restriction. In order to compare with other versions, the time to merge the files must be computed, as required by the Application I/O benchmark specification.

To perform MPI-IO operations using an NFS-based file system it would be necessary to have an implementation of the NFSv3 protocol, due to the need of controlling the file access through locks. Since dNFSp was designed based on NFSv2 protocol, we were forced to perform the BTIO-epio version of the benchmark.

In order to have a more write-intensive benchmark, we have made a small change in the BTIO benchmark code, modifying the frequency of file writes. The original code performes writes on every five iterations, resulting in a total amount of writes of 400 megabytes; with the modification, BTIO performes writes on every iteration, resulting in 2 gigabytes of written data.

3.2. The i-Cluster2

The *i-cluster2* [7] is installed in Montbonnot Saint Martin, France, in the INRIA Rhône-Alpes facility. The cluster is composed by 100 nodes. Each node is equipped with a dual Itanium2 900 MHz with 3 gigabytes of memory and a disk storage with 72 gigabytes, 10000 rpm, SCSI. All the nodes are interconnected using a 1 Gigabit Ethernet network, Fast Ethernet network and Myrinet Network. The experiments were performed using the 1 Gigabit Ethernet network.

The software installed on i-cluster2 is based on Red Hat Enterprise Linux AS release 3 distribution, with a Linux kernel version 2.4.21. The MPI implementation used with the BTIO benchmark is mpich version 1.2.6.

4. Results and Discussion

In this section we present the results obtained with our experiments. The reported execution times are those informed by BTIO at the end of the execution, together with a confirmation of correct computation. Each value reported is the arithmetic mean of at least 5 runs of BTIO with the same configuration, so as to obtain a stable value. Standard deviations lie within a maximum value of 3 seconds.

4.1. Performance Evaluation

The first step in our analysis of dNFSp has been an evaluation of the performance of the system on the i-cluster. We have run BTIO on a large subset of the available nodes, and compared the performance of dNFSp with that of a dedicated parallel file system. We have chosen PVFS [5] for this task, as it is a representative parallel file system in the Beowulf cluster context in which our work in inserted.

For both systems, we have varied the number of IODs in the file server from 4 up to 12 IODs, in steps of 2. In the case of dNFSp, we always use a number of meta-servers equal to that of IODs. PVFS uses only one extra node in all cases, for the *manager*. The experiments have been executed from 4 up to 49 clients, respecting the feature of BTIO that the number of clients must be a perfect square.

We show, in Figure 2, the results obtained using the minimum and the maximum number of IODs, respectively 4 and 12. Intermediate configurations have shown proportional variation. Shorter values are best.

As expected, execution times drop as the number of clients increase. For dNFSp, the reduction in execution time is progressive on the whole range of clients, except for the case of 49 clients using 4 IODs, where it slightly starts to rise again. For PVFS, one observes a lower limit at around 100 seconds. We conclude that the main cause for the limitation in both systems is that, as the number of clients increase, the amount of data written by each one decreases, and reaches a point where parallelism does not pay off anymore due to the management cost of the striping mechanism. dNFSp seems to handle the situation better than PVFS, reaching around 47 seconds in the case of 12 IODs.

It is important to remark that such experiments were run using PVFS v1 (more precisely, version 1.6.3), while PVFS2 is already available and should presumably yield more performance than its predecessor. PVFS2 was effectively our initial choice for comparison, not only because of its performance, but also because it features a distributed metadata model which is closer to that of dNFSp. However, early experiments with that version on the i-cluster resulted in strangely poor performance (results are shown in Table 1 for information). While we are still investigating the cause of that problem, we chose to report results for PVFS v1, which nevertheless represents no loss in significance since it is still fully supported by the developers as a production system.

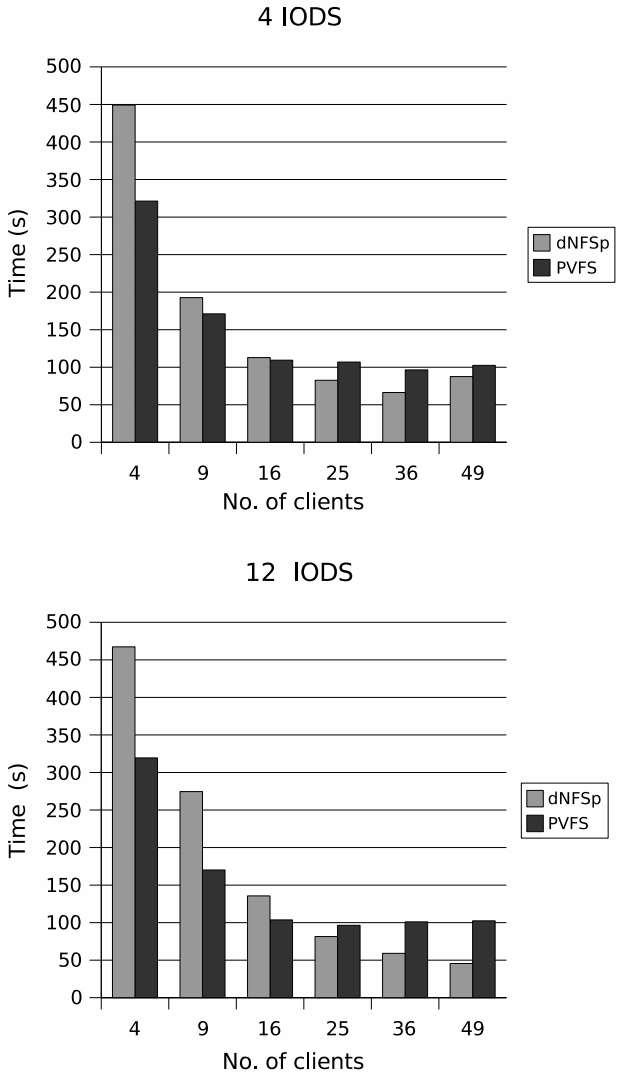

Figure 2. Performance comparison for dNFSp vs. PVFS using 4 and 12 IODs on the file server

No. of clients	dNFSP	PVFS v1	PVFS2
4	464.416	319.273	363.395
9	272.728	170.663	263.655
16	135.164	103.247	253.88
25	76.712	95.35	252.505
36	53.745	96.313	293.44
49	43.776	105.44	353.64

Table 1. Sample execution times (in seconds) obtained for dNFSp and PVFS v1 in comparison to PVFS2

4.2. Scalability Evaluation

As a subsequent analysis of our experiments we have compared how both file systems reacts to the addiction of more nodes to the file system. The number of IODs, meta-servers and clients is the same as described in the previous section.

In Figure 3 we have three samples from our experiment. In the first chart we fixated the number of BTIO clients on 4 and varied the number of IODs and meta-servers. We can see that both file systems sustain a almost constant performance due the fact that the application doesn't have enough clients to stress the capability of storage offered by the file systems. In this case dNFSp even has an small decrease of performance as we add more nodes, this lost of performance comes from the meta-servers communication that is more significant when we have more meta-servers. The better performance of PVFS lies on the size of messages, as we have only four BTIO clients the amount of computation designated to each node is large resulting in larger writes on the file system.

The second chart shows the transition case where both file systems have a similar behavior. Using 25 clients BTIO seams to have a block size that results in similar performance to both file systems. They have an improvement of performance as we add more nodes to the file system, reaching a limit where adding more nodes increases the execution time instead of reducing.

In the third chart we show a more stressing case, where we have 49 clients accessing the file system. In this situation we can see that from 4 to 10 nodes dNFSp has an improvement of performance as we increase the file system size. When we achieve 12 IODs and 12 meta-servers the overhead added by the insertion of more nodes is not compensated by the performance gain. The clients doesn't have enough writes to perform to stress the file system and the communication between meta-servers is more expressive, lying in the same situation shown by the four clients sample. PVFS as shown a performance limitation when we have small writes to the file system as the number of clients is larger than the previous case, we have smalls chunks of data being written.

4.3. Overlapping of IODs and metaservers

As a last experiment, we have investigated the capabilities of dNFSp in saving cluster nodes for the deployment of the file server. As usual in distributed file systems (and distributed systems in general) like dNFSp and PVFS, each task of the system is usually performed on a distinct machine or compute node. For example, in the previous exper-

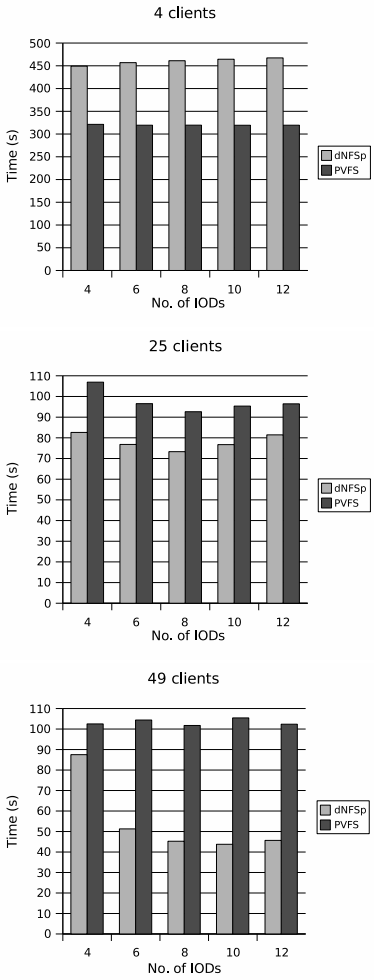

Figure 3. Scalability comparison for dNFSp vs. PVFS using 4 and 49 BTIO clients processes

iments we have always used distinct nodes for running the IODs and the metaservers/manager.

It would be desirable, however, that one could make use of as few nodes for the file server as possible, in order to maximize the number of nodes available for the real computing tasks. While the decision depends mostly on the amount of storage desired for the server, some considerations can be made regarding performance and scalability that might allow for a shorter number of nodes than that initially accounted. This is specially true if the compute nodes are dual-processed.

The results in Figure 4 correspond to the execution of BTIO with dNFSp with IODs and metaservers running on the same nodes, compared to the original execution where the two entities run on separate nodes. Again, we show re-

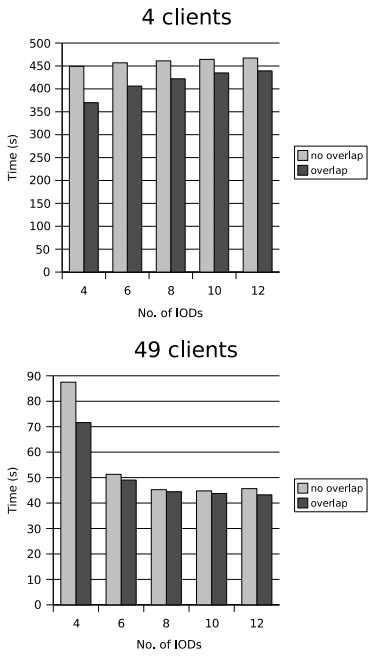

Figure 4. Results for dNFSp with and without overlapping IODs and metaservers on the same nodes

sults for a small and a large number of client nodes.

Two different situations are presented, both favoring the overlapping of IODs and metaservers. In the first case, with few client nodes, more information is written by each single client, and thus there is a visible difference in performance in favor of the overlapping configuration, since communication betweem the IOD and the metaserver on the same node is done faster (by means of memory copy). On the second case, there are much more, smaller client writes, and consequently the differences are not much evident. Increasing the number of IODs also contributes to minimize the differences. As a conclusion, we can see that such an overlapping configuration can be employed without loss of performance, contributing to the amount of nodes dedicated to computation.

5. Related Work

The increasing performance gap between I/O and processor has placed the file system performance as the most severe bottleneck to applications that massively use the storage subsystem [12, 4]. Several approaches have been proposed since the deployment of the first large-scale parallel machines. Many are based on the use of specialized technologies (e.g. RAID, fiber optics) as a means to increase

performance, such as GPFS [13] and GFS [1]. This kind of system usually relies on the concept of a Storage Area Network (SAN), which basically defines a common storage "device" composed of several physical devices. As such, scalability is a direct consequence of this concept. Other projects like Petal/Frangipani [10, 16] and the Shared Logical Disk [15] make use of the same concept, but the SAN is implemented in software over a network. Good performance and scalability thus depend heavily on the communication technology.

Research projects like PVFS [5] and Lustre [14] follow another trend. To achieve high performance on I/O operations, these file systems distribute the functionality of a file system across a set of nodes in a cluster. To perform parallel I/O operations, they stripe the data across the nodes, keeping the striping transparent to the application.

PVFS is a parallel cluster file system composed of two types of nodes: the *I/O server* and the *manager*, which is a metadata server. The nodes in the cluster used by the file system can be configured as I/O servers, and one of them as a manager. Lustre is an object-based file system designed to provide performance, availability and scalability in distributed systems. Like PVFS, Lustre comprises two types of server nodes: Metadata Servers (MDS) are responsible for managing the file system's directory layout, as well as permissions and other file attributes. The data is stored and transferred through the Object Storage Targets (OST). Figure 5 shows the architecture of PVFS and Lustre in comparison to that of dNFSp.

High performance in PVFS is achieved by distributing the contents of a file across the I/O server nodes (striping). Clients are allowed to access the contents of the file in parallel. The way the files are striped is handled by the metadata manager, which is also responsible for managing file properties and a name space to applications, but has no participation in I/O operations. The I/O servers are accessed directly by the clients to deal with the data transfers. The user can access the file system through the PVFS library or using an OS-specific kernel module. The latter allows the user to mount the file system using a POSIX interface, and access files as any other file system.

In Lustre, similarly to PVFS, the client contacts the Metadata Servers to know which OST contains a given part of a file. After obtaining this information, the client establishes direct connections to the OST performing reads and writes. The MDS has no intervention in the I/O process, being contacted by the OST only to change file attributes like file size. Both types of nodes can have replicas working in pairs and taking the place of each other when a failure occurs. Figure 5 shows an *active* MDS, and its replica is represented by the *failover* node. Information concerning the overall system configuration is stored in a Lightweight Directory Access Protocol (LDAP) server. In the event of a MDS failure, the client can access the LDAP server to ask for an available MDS.

As in dNFSp, PVFS version 2, or PVFS2 [9], has the option of running more than one manager. The main difference lies on the way PVFS controls the distribution of metadata. Each manager stores the metadata information about a range of files, while in dNFSp each server has the metadata information about all the files. The PVFS approach can result in a surcharged manager when all the clients access files in the same range.

6. Conclusions and Final Considerations

The execution of the BTIO benchmark with dNFSp and PVFS on the i-cluster2 has confirmed the objectives of our system in providing good performance and scalability while keeping compatibility with NFS. The results show very good, scalable performance in comparison to a dedicated parallel file system. dNFSp is able to reach the same level of performance of PVFS, and many times even reach beyond it. We understand that this advantage comes from the fact that dNFSp can tolerate a smaller size of writes than PVFS before reaching the point where parallelism in no longer benefic. When configured with a large number of clients, dNFSp has outperformed PVFS in up to 50% of its execution time.

Another positive aspect of our benchmarking is that dNFSp performs efficiently when IODs and metaservers have been put together on the same nodes. This allows for a resource-saving configuration which maximizes the availability of compute nodes without sacrificing performance. Although the nodes of the i-cluster2 are dual-processed, which favors this configuration, we believe that a similar approach, at least partial, should be possible on single-processor clusters, since the IODs present a typical I/O-bound profile, while the metaservers do little disk activity. This evaluation was not possible on the i-cluster2, as it would require booting with a non SMP-enabled kernel, but we intend to carry it out as soon as possible.

Our future activities include further benchmarking with dNFSp and specially the implementation of a dedicated communication mechanism between meta-servers. One can notice some loss of performance in a few of the experiments due to the relatively heavy lookup mechanism that the metaservers performs. A dedicated protocol should minimize the impact of lookup operations by implementaing some kind of prefetching and message aggregation for the exchange of metadata.

References

[1] The openGFS project, Apr. 2003. http://opengfs.sourceforge.net.

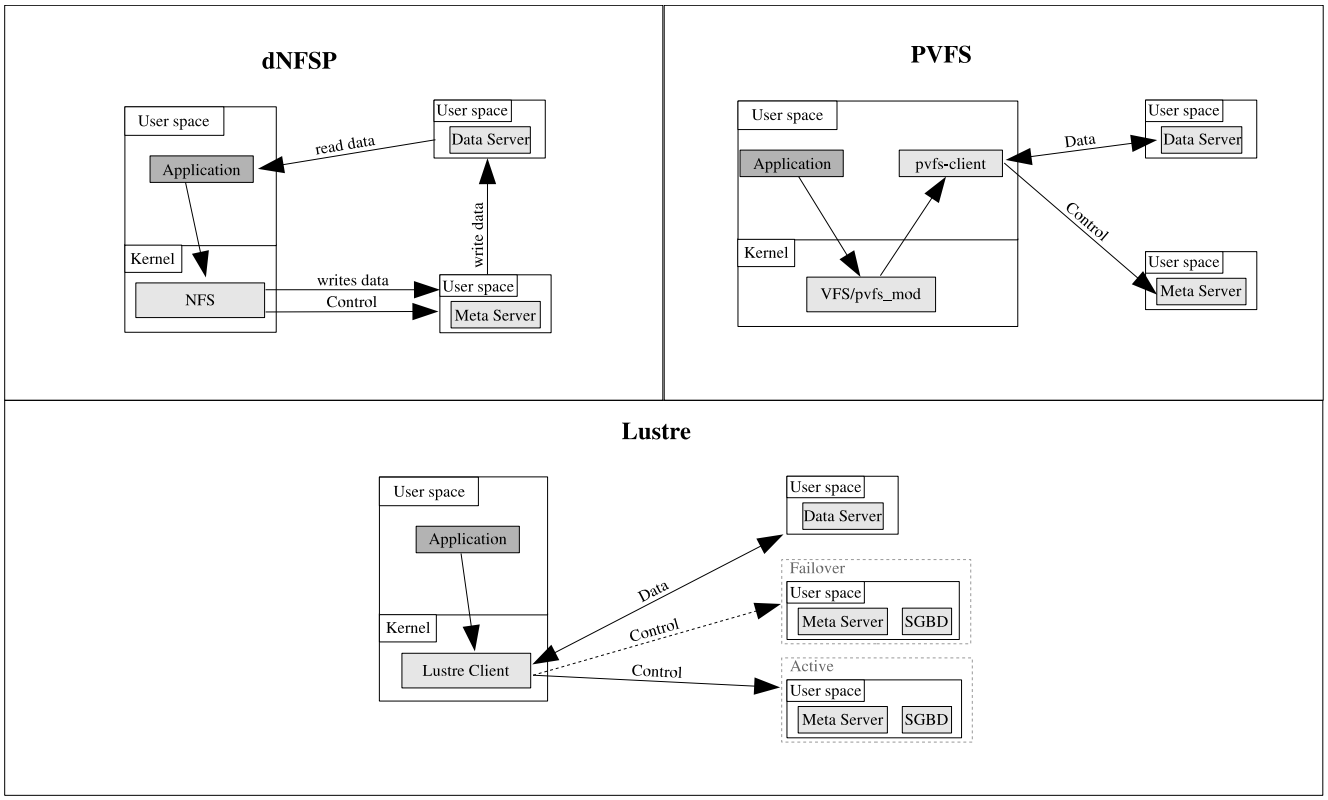

Figure 5. Architecture of the studied cluster file systems

[2] R. B. Ávila, P. O. A. Navaux, P. Lombard, A. Lebre, and Y. Denneulin. Performance evaluation of a prototype distributed NFS server. In J.-L. Gaudiot, M. L. Pilla, P. O. A. Navaux, and S. W. Song, editors, *Proceedings of the 16th Symposium on Computer Architecture and High-Performance Computing*, pages 100–105, Foz do Iguaçu, Brazil, 2004. Washington, IEEE.

[3] D. H. Bailey et al. The NAS parallel benchmarks. *International Journal of Supercomputer Applications*, 5(3):63–73, 1991.

[4] M. Baker, editor. *Cluster Computing White Paper*. IEEE Task Force in Cluster Computing, Dec. 2000. Available at http://www.dcs.port.ac.uk/∼mab/tfcc/WhitePaper/final-paper.pdf. Final Release, Version 2.0.

[5] P. H. Carns, W. B. Ligon III, R. B. Ross, and R. Thakur. PVFS: a parallel file system for Linux clusters. In *Proc. of the 4th Annual Linux Showcase and Conference*, pages 317–327, Atlanta, GA, 2000. Best Paper Award.

[6] Cluster File Systems, Inc. Lustre: A scalable, high-performance file system, 2002. Available at http://www.lustre.org/docs/whitepaper.pdf (July 2004).

[7] i-Cluster 2, 2005. Available at <http://i-cluster2.inrialpes.fr>. Access in May 2005.

[8] R. Kassick, C. Machado, E. Hermann, R. Ávila, P. Navaux, and Y. Denneulin. Evaluating the performance of the dNFSP file system, 2005. Aceito para publicação no CCGrid 2005, a ser realizado em Cardiff, Reino Unido, de 9 a 12 de maio de 2005.

[9] R. Latham, N. Miller, R. Ross, and P. Carns. A next-generation parallel file system for Linux clusters. *LinuxWorld Magazine*, Jan. 2004.

[10] E. K. Lee and C. A. Thekkath. Petal: Distributed virtual disks. In *Proc. of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 84–92, Cambridge, MA, 1996.

[11] Nfsp: homepage, 2000. Available at <http://www-id.imag.fr/Logiciels/NFSP>. Access in May 2005.

[12] E. Schikuta and H. Stockinger. Parallel I/O for clusters: Methodologies and systems. In R. Buyya, editor, *High Performance Cluster Computing: Architectures and Systems*, chapter 18, pages 439–462. Prentice Hall PTR, Upper Saddle River, 1999.

[13] F. Schmuck and R. Haskin. GPFS: A shared-disk file system for large computing clusters. In *Proc. of the Conference on File and Storage Technologies*, pages 231–244, Monterey, CA, 2002.

[14] P. Schwan. Lustre: Building a file system for 1000-node clusters. In *Proceedings of the 2003 Linux Symposium*, July 2003.

[15] R. A. Shillner and E. W. Felten. Simplifying distributed file systems using a shared logical disk. Technical Report TR-524-96, Dept. of Computer Science, Princeton University, Princeton, NJ, 1996.

[16] C. A. Thekkath, T. Mann, and E. K. Lee. Frangipani: A scalable distributed file system. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles*, pages 224–237, Saint Malo, France, 1997. New York, ACM Press.

[17] P. Wong and R. F. V. der Wijngaart. NAS Parallel Benchmarks I/O Version 2.4. RNR 03-002, NASA Ames Research Center, 2003.

ANEXO B RESUMO ESTENDIDO EM LÍNGUA FRANCESA

Incluímos neste anexo um resumo estendido da tese, em língua francesa, redigido em cumprimento à convenção estabelecida com o Institut National Polytechnique de Grenoble para realização da tese em co-tutela.

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

N° attribué par la bibliothèque
/_/_/_/_/_/_/_/_/_/

THÈSE

pour obtenir le grade de

DOCTEUR DE L'INPG

Spécialité : "Informatique"

préparée au laboratoire Informatique et Distribution
dans le cadre de ***l'École Doctorale "Mathématiques, Sciences et Technologies de l'Information, Informatique"***

présentée et soutenue publiquement

par

Rafael BOHRER ÁVILA

Le 17 juin 2005

Titre :

Un Modèle de Distribution du Serveur de Fichiers pour Grappes

---

Directeurs de thèse : Brigitte PLATEAU et Philippe NAVAUX

JURY

M. JEAN-FRANÇOIS MÉHAUT	Président
M. BERNARD TOURANCHEAU	Rapporteur
M. SIANG SONG	Rapporteur
M^me^ BRIGITTE PLATEAU	Directeur de thèse
M. PHILIPPE NAVAUX	Directeur de thèse
M. YVES DENNEULIN	Co-encadrant

Table des matières

1 Introduction **5**

2 Systèmes de Fichiers à Haute Performance **7**

2.1 Terminologie 7

2.2 Systèmes à disques partagés 8

2.2.1 XFS 8

2.2.2 Frangipani/Petal 9

2.2.3 GFS 9

2.2.4 GPFS 10

2.3 Systèmes distribués 10

2.3.1 NFS 10

2.3.2 xFS 11

2.3.3 PVFS 12

2.3.4 DPFS 12

2.3.5 Lustre 13

2.4 Évaluation des systèmes présentés 13

2.4.1 Systèmes à disques partagés 13

2.4.2 Systèmes distribués 14

2.4.2.1 Gestion de données 14

2.4.2.2 Gestion de méta-données 14

2.4.2.3 Tolérance aux pannes 15
2.4.2.4 Adaptation au modèle Beowulf 15
2.4.3 Bilan 15

3 NFSP : NFS Parallèle 17
3.1 Structure de NFSP 17
3.2 Implémentation de prototypes 18

4 Le Modèle de Distribution Proposé : dNFSP 21
4.1 Vision générale de la solution proposée 22
4.2 Optimisation des opérations d'écriture dans NFSP : dNFSP 23
4.3 Maintien de la cohérence entre les méta-serveurs 24
4.4 Politiques d'actualisation des méta-fichiers 25
4.5 Considérations sur le modèle proposé 25

5 Validation de dNFSP : Implémentation et Mesures 27
5.1 Implémentation sur uNFSP 27
5.1.1 Adaptations au méta-serveur 28
5.2 Mesures expérimentales 28
5.2.1 Disques et réseau 28
5.2.2 Lectures et écritures séquentielles 29
5.2.3 Sur-coût de récupération de méta-fichiers 30
5.3 *Benchmarks* 32
5.3.1 BTIO 32
5.4 Application : GADGET 33
5.5 Bilan 34

6 Conclusions 35

1

Introduction

La réalisation du calcul parallèle par moyen de grappes d'ordinateurs (*Cluster Computing*) est un champs de recherche qui s'est développé très vite au cours des dernières années [1, 33], et pour lequel on dédie aujourd'hui une quantité importante de temps et de ressources.

L'intérêt pour les grappes vient principalement du fait que ses composants matériels et logiciels font partie du marché « off-the-shelf », à la portée du grand public, ce qui apporte immédiatement deux avantages significatifs : suivi rapide des évolutions technologiques, comme les réseaux Gigabit Ethernet [15] e Myrinet [9], et faible coût, ceci principalement par l'utilisation de logiciel libre comme GNU/Linux [3, 2, 4], PVM [14] et MPI [20].

Les origines du calcul haute performance basé sur grappes remonte au début des années 90, lorsque des chercheurs de la NASA ont assemblé les premiers prototypes de la grappe Beowulf [33, 27, 34, 35]. Quelques années après, l'idée étant déjà diffusée partout dans le monde, les grappes ont même pu rentrer dans le TOP500, la liste des machines les plus puissantes du monde.

Ayant hérité des réseaux Unix ses caractéristiques techniques, l'un des principaux problèmes liés à la gestion de grappes Beowulf est la performance et la passage à l'échelle du système de fichiers. Le *Network File System* [24], ou NFS, système de fichiers standard pour le partage de fichiers dans les réseaux locaux, a aussi été emprunté dans les grappes. Le serveur NFS étant toutefois centralisé, ce modèle n'est pas bien adapté aux grappes de grande taille, en particulier dans les cas des applications gourmandes en entrée/sortie [28].

Pour attaquer ce problème, deux approches ont été établies au cours des dernières années : soit l'utilisation de technologies plus performantes comme la fibre optique et les disques RAID [22], soit la répartition des fonctionnalités du serveur de fichiers sur un

ensemble de machines. Bien que le premier se montre plus performant, le dernier est plus adapté au modèle Beowulf.

C'est dans ce contexte que le projet NFSP a été établi, en 2001, au sein du Laboratoire ID de Grenoble, avec le but principal d'offrir une solution de stockage répartie dans une grappe, solution que doit être performante et adaptée aux demandes d'entrée/sortie des applications parallèles, mais qui peut également garder la compatibilité avec les clients NFS standards, ainsi que les procédures de configuration et de gestion d'un système NFS traditionnel. Le projet a depuis produit de nombreux travaux scientifiques d'implantation de prototypes, outils et publications d'articles.

Parmi les possibilités de recherche sur NFSP, on trouvera le sujet principal de la thèse présentée ici : le modèle de distribution de NFSP n'étant conçu que pour l'optimisation des opérations de lecture de données, les écritures concurrentes souffrent encore du modèle centralisé de NFS. Ceci est donc le principal problème sur lequel nous avons travaillé, dont les résultats sont présentés dans les pages suivantes.

2

Systèmes de Fichiers à Haute Performance

Le problème de stockage efficace de données a toujours existé en calcul parallèle, vu que les problèmes présentant une grande quantité de données font souvent l'objet de recherche dans les activités scientifiques. Par conséquent, plusieurs solutions d'optimisation des systèmes de fichiers ont été proposées ; dans ce chapitre, nous en présenterons certaines, que nous considérons représentatives et qui sont adaptées au scénario envisagé pour le travail.

Pour bien organiser la présentation, nous avons divisé les systèmes étudiés en deux groupes :

- les systèmes à *disques partagés*, où l'espace de stockage de tous les disques disponibles est traité de manière égalitaire par toutes les machines
- les systèmes *distribués*, où les fonctions du système de fichiers sont réparties sur un sous-ensemble des machines

2.1 Terminologie

Nous présentons d'abord quelques mots et expressions-clé couramment utilisés dans la littérature de systèmes de fichiers.

Données Les informations enregistrées par les utilisateurs dans les fichiers.

Méta-données Les « données sur les données », ou bien les informations de contrôle

(taille, propriétaire, date de création/modification, etc.) qui forment, avec les données, la notion complète d'un fichier.

Méta-fichier Certains systèmes distribués utilisent des fichiers locaux pour garder les méta-données de fichiers distants ; ces fichiers locaux sont les méta-fichiers.

Serveur d'entrée/sortie (IOD) Utilisés souvent par les systèmes distribués, les IODs sont des démons dédiés à la lecture et l'écriture de blocs de données.

Serveur de méta-données Entité complémentaire aux IODs, sert à fournir des méta-données à distance.

Striping Le *striping* consiste à diviser les données d'un fichier en plusieurs morceaux, ou « tranches », afin de les faire manipuler en parallèle (e.g. par les IODs).

NAS (*Network Attached Storage*) Caractéristique d'un élément de stockage qui lui permet d'être branché directement sur le réseau (par exemple par une connexion Ethernet).

SAN (*Storage Area Network*) Utilisation d'un ensemble de disques en tant qu'un seul espace de stockage de façon transparente.

2.2 Systèmes à disques partagés

2.2.1 XFS

XFS [36] est un système de fichiers développé pour IRIX, de Silicon Graphics, depuis 1994. Son but principal est le support aux fichiers de grande taille (de l'ordre du *petabyte*) aussi que la haute performance et le passage à l'échelle.

Le mécanisme de base est l'utilisation des arbres B+ dans les structures internes de données, ce qui permet une manipulation efficace de blocs de données et de la gestion de l'espace libre. XFS utilise également un système de transactions identique à celui des bases de données, afin de supporter la tolérance aux pannes et une récupération rapide en cas de défaillance.

Bien qu'il soit un système de fichiers à vocation généraliste, XFS présente des caractéristiques bénéficiant au calcul parallèle, comme la possibilité d'accès concurrent à la plupart des fonctions et le *direct I/O*, ou accès direct à l'E/S, qui permet à certaines applications d'accéder directement aux disques.

XFS a été porté dans le noyau Linux et est donc librement distribué avec celui-ci.

2.2.2 Frangipani/Petal

Le système de fichiers Frangipani/Petal [37] consiste de deux composants : la couche *Petal*, qui fournit un service de stockage distribué ayant la capacité de passage à l'échelle et de tolérance aux pannes, et *Frangipani*, qui implémente le système de fichiers proprement dit, en créant les abstractions fichier et répertoire au dessus de la couche de base.

La couche Petal implémente un service de *disque virtuel*. Le disque virtuel est composé de plusieurs disques physiques situés dans des machines distinctes. Des démons exécutant sur ces machines communiquent via le réseau pour coordonner la distribution des opérations de lecture et d'écriture.

Dans la couche supérieure, le sous-système Frangipani fournit aux machines clientes la vision de fichiers et de répertoires, en utilisant l'espace disque virtuel créé par Petal comme si c'était un unique dispositif de stockage. Il implémente un mécanisme de verrous distribués pour garantir la cohérence des données. Frangipani utilise aussi un système d'historique de transactions pour la récupération de pannes.

2.2.3 GFS

Le *Global File System* [31, 23] est un système de fichiers à haute performance développé par l'Université de Minnesota. Il vise surtout l'utilisation des caractéristiques de faible latence et de haut débit des technologies réseau comme Fibre Channel et Gigabit Ethernet pour éliminer le rôle central du serveur et construire ce que les auteurs nomment « système symétrique multi-clients ».

Les dispositifs de stockage de GFS ont la caractéristique NAS, de sorte que chacun d'entre eux (e.g. disque dur, volume RAID) est directement accessible par le réseau IP, ce qui forme le *Network Storage Pool*.

Plusieurs mécanismes de contrôle y ont été ajoutés depuis la version originale pour garantir le passage à l'échelle : par exemple, les *device locks* permettent des actions atomiques à grain fin sur les disques ; les *stuffed dinodes* qui consistent en l'inclusion de données dans les propres i-nodes de méta-données ; et les *extendible hashing*, un mécanisme efficace pour l'enregistrement et la localisation d'informations sur les fichiers.

Sistina Software est devenue propriétaire des droits commerciaux en 2001, et un projet parallèle OpenGFS continue le développement de la version libre sous Linux.

2.2.4 GPFS

GPFS [29] est un projet développé par IBM depuis la fin des années 90, et est le système actuellement utilisé dans la machine ASCI White.

Les disques partagés peuvent être connectés directement au réseau, s'ils ont la capacité NAS, ou bien gérés par un serveur local qui fait l'interface avec les clients. Dans les deux cas, le protocole d'accès aux disques est défini par les fonctions usuelles d'accès aux blocs de données.

Le système parallélise complètement l'accès aussi bien à un ensemble de fichiers qu'à des blocs distincts au sein d'un même fichier. La taille de blocs est volontairement grande, afin de minimiser les pertes dûes au temps de recherche (*seek overhead*). Dans le cas de petits fichiers, il est aussi possible d'utiliser de *sous-blocs* qui peuvent avoir jusqu'à 1/32ème de la taille d'un bloc normal.

Un mécanisme de contrôle d'accès est implémenté par un système distribué de verrous. Chaque noeud de calcul exécute un gestionnaire local de verrous, qui communique avec le gestionnaire global pour obtenir de *tokens* de verrouillage.

2.3 Systèmes distribués

2.3.1 NFS

NFS, ou *Network File System* [11], est le standard *de facto* pour le partage de fichiers dans les environnements Unix, et par conséquent dans la communauté des grappes. Le système a été conçu par Sun Microsystems en 1985.

En réalité, NFS est un protocole pour l'accès distant et transparent d'un client au système de fichiers d'un serveur. Son but initial n'est donc pas la haute performance, mais l'utilisation dans des réseaux locaux de stations de travail comme le font AFS [25] et Coda [26].

Le système a été conçu pour être *sans état*. Chaque requête d'un client doit être complètement identifiée par la localisation du fichier, la position et la taille du bloc de données manipulé (en ce sens, le serveur NFS peut être considéré comme à la fois gestionnaire de méta-données et serveur de données, dans le même démon). Ceci favorise la tolérance aux pannes : les pannes de clients n'affectent pas le serveur, et les pannes du serveur sont supportés de façon simple et transparente par les clients, le fonctionnement redevenant normal quand cesse la panne. Pour cette raison, NFS a été amplement diffusé dans les environnements Unix. D'autres caractéristiques importantes sont le support à l'hétérogénéité entre serveurs et clients et une sémantique propre d'accès (indépendante de tout système d'exploitation), ce qui favorise la portabilité.

Malgré les avantages cités, NFS pose un problème potentiel de passage à l'échelle dans les grappes de grande taille, à cause de sa centralisation à trois niveaux : *i*) la connexion réseau au serveur, *ii*) le disque du serveur, et *iii*) le processeur du serveur. Pour cette raison, des nombreux projets de recherche essaient d'améliorer NFS ou de trouver des solutions alternatives.

NFSˆ2 [21] est une architecture d'intégration de plusieurs serveurs NFS sous le même espace de nommage. Un ensemble de serveurs NFS conventionnels est utilisé comme espace de stockage ; un répartiteur de charge est introduit entre l'espace de stockage et les clients, qui analyse les requêtes reçus et les dirige vers le serveur correspondant, ce qui découple la localisation visible d'un fichier de sa localisation physique.

D-NFS [19], ou *Distributed NFS*, est un projet de l'Université Tsinghua de Pekin, Chine. Le principe de fonctionnement du système est la distribution du serveurs NFS traditionnel en plusieurs copies, chacune exécutant dans une machine distincte, où chaque copie sert seulement un sous-ensemble des clients totaux. Si un client tente d'accéder à un fichier qui est indisponible sur le serveur auquel il est connecté, il est nécessaire de copier le contenu complet du fichier d'un serveur distant, ce qui entraîne une opération lente de transfert de données via le réseau.

Bigfoot-NFS [18] suit une approche différente : la modification du logiciel client au lieu de celui du serveur. L'envoi de requêtes par les clients est fait par une bibliothèque de RPCs en vecteur, grâce à laquelle le même service peut être envoyé à plusieurs machines en même temps. Le serveur de fichiers de Bigfoot-NFS est constitué d'un groupe de serveurs NFS traditionnels, complètement indépendants les uns des autres. De même que NFSˆ2 et D-NFS, Bigfoot-NFS stocke des fichiers entiers.

2.3.2 xFS

xFS [5] est un projet de système de fichiers « sans serveur » développé dans l'Université de Californie à Berkeley de 1993 à 1995. Il présente une structure complètement distribuée, où les données et les méta-données sont diffusées parmi l'ensemble des machines disponibles (soit la totalité, soit une partie d'entre elles) et peuvent de plus être migrées.

Le système est composé de 4 types d'entités : *serveurs de stockage*, *gestionnaires*, *ramasse-miettes* et clients. Les serveurs de stockage sont des serveurs d'entrée/sortie, c'est à dire responsables de la lecture et de l'écriture de blocs de données sur les disques ; les gestionnaires sont des serveurs de méta-données ; les ramasse-miettes sont des entités particulières de xFS qui exercent la fonction de défragmenter et recycler l'espace disque de façon à récupérer de l'espace pour des futures opérations d'écriture.

Un mécanisme de cache coopératif est défini pour diminuer le retard dû à la recherche de données et de méta-données. Le principe de ce mécanisme est qu'un client peut obtenir

des informations qui sont déjà disponibles sur un autre client. Cela forme un cache à trois niveaux dans xFS, avec un degré variable de retard : un bloc de données peut être dans le cache local d'un noeud (retard de cache uniquement), dans le cache d'un autre noeud (retards de cache et réseau), ou bien sur les disques (retards de cache, réseau et disques).

Malgré des bons résultats, xFS n'a pas été poursuivi après la conclusion du projet NOW, et par conséquent aucun portage sous Linux n'est connu à ce jour.

2.3.3 PVFS

PVFS [12] est le système de fichiers conjoint entre le laboratoire PARL/Université Clemson et le laboratoire ANL à Argonne, États-Unis. Son objectif est de fournir un système de fichiers à haute performance pour les grappes Beowulf.

Le système peut être considéré comme une version simplifiée de xFS. L'accès aux données est réalisé par des serveurs d'entrée/sortie, et un gestionnaire unique (et donc central) est responsable du stockage et de la diffusion des informations de méta-données. Les clients disposent de trois méthodes pour accéder au système de fichiers : au moyen d'une interface dédiée, par des appels POSIX standards, où encore via MPI-IO.

Chaque fichier de PVFS est divisé entre plusieurs serveurs d'entrée/sortie par *striping*. L'accès à un fichier est divisé en deux étapes : premièrement, le client contacte le gestionnaire pour obtenir une spécification de la localisation du fichier ; deuxièmement, le client contacte directement les serveurs d'E/S pour exécuter l'opération désirée. Le gestionnaire n'a pas à être contacté de nouveau pour des opérations sur le même fichier.

2.3.4 DPFS

Le DPFS [30] (*Distributed Parallel File System*) a été développé dans l'Université Northwestern, aux États-Unis, avec le but principal de récupérer les ressources de stockage éventuellement disponibles dans un réseau (e.g. les disques durs des stations de travail), ce qui permet l'exécution d'applications manipulant des grands volumes de données, et en même temps d'obtenir de bonnes performances grâce à la combinaison du parallélisme et de la distribution.

DPFS présente deux caractéristiques nouvelles par rapport aux autres systèmes de fichiers distribués rencontrés dans la littérature : la méthode n'est pas statique et peut même être personnalisée par l'utilisateur final, et la gestion de méta-données est implémentée par une base de données relationnelle, comme Postgres ou MySQL. L'accès au système de fichiers par un client se fait de même que dans PVFS, en deux étapes.

Le système définit trois *niveaux de fichiers*, chacun ayant une méthode correspondante distincte pour faire le *striping*. Pour la méthode *linéaire*, la division du fichier se fait en

tranches de taille fixe. Pour la méthode *multi-dimensionnelle*, les tranches sont regroupées pour correspondre aux patrons d'accès typiques des applications scientifiques. Enfin, la méthode *striping par bloc*, correspond spécifiquement aux patrons d'accès de HPF.

Le stockage des informations dans une base de données relationnelle permet l'utilisation d'une interface de haut niveau et fiable, principalement grâce aux mécanismes de transactions atomiques, qui garantissent automatiquement la cohérence du système de fichiers. Pour accéder au système, les clients DPFS doivent utiliser une interface spécifique, ou bien un *shell* Unix modifié.

2.3.5 Lustre

Lustre [13] est l'un des plus récents systèmes de fichiers distribués, développé par Cluster File Systems, Inc., orienté spécifiquement vers les grappes d'ordinateurs. Ce système mélange des caractéristiques bien connues comme la séparation entre données et méta-données avec des techniques modernes de gestion comme LDAP et XML.

Lustre considère les fichiers comme des objets, dont la localisation est récupérée au moyen des MDS (*Metadata Servers*, serveurs de méta-données). Les MDS supportent des opérations au niveau de l'espace de nommage du système de fichiers. Les opérations d'accès aux blocs de données sont redirigées sur les *objets de stockage*, ou OST (*Object Storage Targets*), qui gèrent le stockage d'informations physiquement situées dans des *disques basés sur objets*, ou OBD (*Object-Based Disks*).

Les MDS gardent un historique sur les transactions de modifications des méta-données et sur l'état de la grappe, afin de supporter et récupérer des pannes éventuelles sur le réseau et/ou du matériel. Un MDS a un réplicat qui prend en charge ses fonctions en cas de panne.

Lustre supporte l'hétérogénéité du réseau par l'utilisation d'une abstraction appelée NAL (*Network Abstraction Layer*). C'est une couche logiciel qui normalise les fonctions d'accès au réseau, jouant le rôle d'intermédiaire entre le reste du système et la technologie utilisée.

2.4 Évaluation des systèmes présentés

2.4.1 Systèmes à disques partagés

Il est relativement difficile d'analyser les systèmes de fichiers dans ce groupe, car la plupart d'entre eux exigent des technologies spéciales, normalement non disponibles pour la réalisation de tests. En plus, la documentation existante est généralement plus commerciale que technique. Les indicateurs de performance montrent toujours des débits et un

passage à l'échelle très bons, ce qui est compréhensible au vu des techniques employées, mais cela reste une information donnée par le propre constructeur. De toutes façons, on constate que ces systèmes sont effectivement utilisés dans des machines parallèles commerciales de nos jours, ce qui justifie leur étude.

2.4.2 Systèmes distribués

Les systèmes de fichiers distribués étant plus proche des objectifs du travail développé, une analyse plus détaillée en est faite dans cette section.

2.4.2.1 Gestion de données

Les systèmes exposés présentent des formes différentes de traiter les données des fichiers. Dans NFS et ses dérivés, les données d'un fichier son traitées intégralement, c'est à dire sans l'utilisation d'aucun mécanisme de *striping*. Cela favorise la simplicité du système.

Une autre possibilité est effectivement l'utilisation de *striping*, comme le font xFS, PVFS, DPFS et Lustre. La division du contenu des fichiers en tranches permet que plusieurs processeurs puissent accéder en parallèle aux blocs de données, ce qui améliore l'exécution des opérations. De plus, ce mécanisme sert à réduire les problèmes de contention lorsque plusieurs processeurs essayent d'accéder à un même fichier, car il est probable que tous les processeurs n'accèdent pas aux même blocs.

On considère, par conséquent, que le *striping* de données est une caractéristique essentielle pour la haute performance dans un système de fichiers distribué.

2.4.2.2 Gestion de méta-données

La gestion de méta-données est réalisée de deux manières principales : centralisée et distribuée. Dans le cas centralisé on trouve la plupart des systèmes, grâce à sa simplicité d'implémentation, mais aussi parce que la plupart des opérations d'E/S d'une application scientifique manipulent les données proprement dites, le coût d'accès aux méta-données devenant masqué par le précédent. Au niveau de l'implémentation, NFS ne sépare pas la gestion de données et de méta-données, les deux étant gérés par le démon *nfsd*, que l'on considère le serveur proprement dit. Des systèmes comme PVFS, DPFS et Lustre, par contre, présentent un serveur dédié pour la gestion de méta-données. Cette approche est relativement simple à implémenter et ne pose pas de problèmes de performance comme nous venons d'exposer.

Dans les cas de xFS e D-NFS, aussi que pour la version 2 de PVFS, la gestion de

méta-données est distribuée. Cela signifie que les informations sur la localisation des blocs de données d'un fichier se trouvent réparties sur les machines disponibles, voire même répliquées. L'avantage de cette approche est la possibilité de tolérance aux pannes dès la réplication d'informations, ainsi que l'élimination du point centralisé d'accès aux méta-données. En revanche, ce mécanisme est plus complexe à gérer, car il faut garantir la cohérence des méta-données afin d'éviter qu'aucun client n'obtienne de données périmées.

2.4.2.3 Tolérance aux pannes

La tolérance aux pannes peut être distinguée dans différents niveaux des systèmes étudiés. On la trouve associée à la gestion de données, de méta-données, ou même dans le domaine du temps. Ceci est le cas pour NFS, où les interruptions temporaires de communication entre serveur et client peuvent être tolérées de façon transparente.

DPFS présente un mécanisme indirect de récupération de méta-données, grâce à l'utilisation d'un gestionnaire de base de données. Dans xFS et Lustre, aussi bien la gestion des données que celle des méta-données utilisent des techniques de tolérance aux pannes. xFS présente une distribution d'informations semblabe à la technique RAID. Lustre utilise également RAID et présente un serveur secondaire de méta-données pour le cas de pannes dans le serveur primaire.

De plus, tous les systèmes de fichiers étudiés peuvent bénéficier des techniques de journalisation des systèmes de fichiers locaux (e.g. ext3, ReiserFS) couramment utilisées de nos jours, ce qui permet déjà un niveau minimal de tolérance aux pannes.

2.4.2.4 Adaptation au modèle Beowulf

Ce point est assez subjectif, mais considéré important étant donnée la popularisation du modèle Beowulf de calcul parallèle. En réalité, le seul système distribué non-conformant est xFS, qui n'a pas d'implémentation pour Linux. Par contre, peu de ces versions libres peuvent être considérés mûres et présentent un cycle stable de développement, les exceptions étant PVFS, Lustre et NFS. De ce point de vue, ces derniers seraient les choix possibles pour une grappe actuelle.

2.4.3 Bilan

Après cet exposé, nous pouvons constater que les systèmes de fichiers les plus adaptés au modèle de grappe Beowulf sont ceux distribués, grâce principalement à la disponibilité du logiciel et la non-exigence de technologies spécifiques. Parmi les techniques utilisées, on peut distinguer le *striping* comme l'une des plus performantes. En ce qui concerne

TAB. 2.1 *Comparaison des caractéristiques des systèmes étudiés*

Système	Gestion de Données	Gestion de Méta-Données	Tolérance aux Pannes[a]	Caractéristique Beowulf
XFS	SAN	SAN	M	pleine
Frangipani/Petal	SAN	SAN	D/M	moyenne
GFS	SAN	SAN	D	moyenne
GPFS	SAN	SAN	D/M	aucune
NFS	fichier	centralisée	temporaire	pleine
NFSˆ2	fichier	centralisée	?	moyenne
D-NFS	fichier	distribuée	temporaire	moyenne
Bigfoot-NFS	fichier	centralisée	?	moyenne
xFS	RAID	distribuée	D/M	aucune
PVFS	*striping*	centralisée	aucune	pleine
DPFS	*striping*	centralisée	M	moyenne
Lustre	RAID	centralisée	D/M	pleine

[a] D = données ; M = méta-données

la tolérance aux pannes, bien qu'elle soit désirable à différents niveaux, elle n'est pas implémentée par tous les systèmes.

Le tableau 2.1 présente une comparaison des principales caractéristiques observées dans les systèmes de fichiers présentés.

3

NFSP : NFS Parallèle

Parmi les systèmes de fichiers étudiés, nous pouvons distinguer un sous-ensemble basé sur le protocole NFS standard. La motivation principale pour ce modèle vient des caractéristiques de stabilité et de qualité d'implémentation de NFS, acquises au long de 20 années de développement. Suivant le même esprit, le projet NFSP a été établi en 2000 au sein du Laboratoire Informatique et Distribution (ID) de l'IMAG, à Grenoble, en tant que partie d'un projet d'implantation d'une grappe de grande taille. Dans ce chapitre nous présenterons donc les principales caractéristiques de NFSP, qui fournit la base pour le développement de cette thèse.

3.1 Structure de NFSP

L'architecture de NFSP est inspirée de PVFS. Le but principal est de paralléliser au maximum les opérations d'accès aux blocs de données, toutefois sans entraîner des modifications importantes dans les procédures standards d'installation et de fonctionnement du protocole NFS.

Les clients dans un système NFSP sont des clients NFS standards (e.g. ceux disponibles dans le noyaux Linux). Cette caractéristique, envisagée depuis le début du projet, permet que des outils de gestion déjà existants puissent toujours être utilisés, et que la grappe soit encore compatible avec des solutions commerciales comme celles de MandrakeSoft ou Red Hat. L'amélioration de la performance vient donc de la parallélisation des fonctions du serveur, lequel est séparé dans deux entités : le *méta-serveur* et les *serveurs d'E/S*, ou IODs.

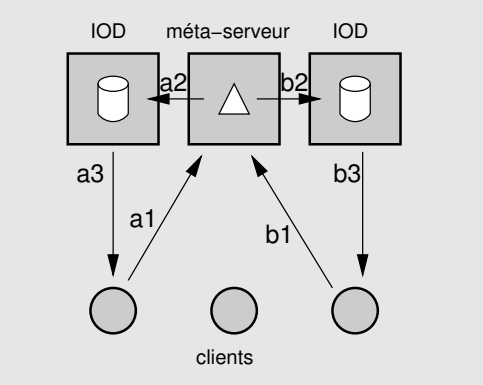

Figure 3.1 *Fonctionnement de NFSP*

Les IODs sont des processus indépendants exécutant sur plusieurs noeuds de la grappe. Leur rôle est uniquement le stockage et la récupération de données. Les informations sont enregistrées dans les disques des machines en utilisant la technique de *striping*.

Le méta-serveur est un démon, lancé sur l'un des noeuds, qui apparaît pour les clients comme le serveur NFS d'une installation traditionnelle. Cela signifie que le méta-serveur reçoit les requêtes de montage/démontage et d'accès aux blocs de données comme dans le cas standard. De même que pour NFS, NFSP exporte une ou plusieurs hiérarchies de répertoires existants dans le système de fichiers local du méta-serveur ; toutefois, la hiérarchie ne contient pas les fichiers exportés, mais plutôt un ensemble de *méta-fichiers*, où chacun contient des informations sur la localisation des données du fichier correspondant.

Lorsque le méta-serveur reçoit une requête de l'un des clients, il récupère d'abord les informations contenues dans le méta-fichier en question, et ensuite redirige la requête au(x) IOD(s) impliqué(s). L'IOD exécute l'opération désirée (lecture ou écriture de données) et renvoie le résultat au client *comme si c'était le serveur NFS* (en utilisant des techniques de *IP spoofing*, si nécessaire). Quand des requêtes sont envoyées par plusieurs clients en même temps, on obtient l'amélioration de la performance grâce à l'exécution parallèle des IODs. La figure 3.1 illustre ce modèle de fonctionnement.

3.2 Implémentation de prototypes

Plusieurs prototypes du modèle NFSP ont été implémentés depuis le début de ce projet. Motivé par la possibilité d'un cycle rapide de modifications et de tests, le premier prototype a été basé sur l'implémentation au niveau d'utilisateur de NFS (version 2.2beta47). Cette version de NFSP est par conséquent nommé uNFSP.

Une évaluation de performance a été conduite dans la grappe i-cluster, en faisant lire en parallèle par les clients un fichier de 1 Go, et en mesurant le débit global de transfert de

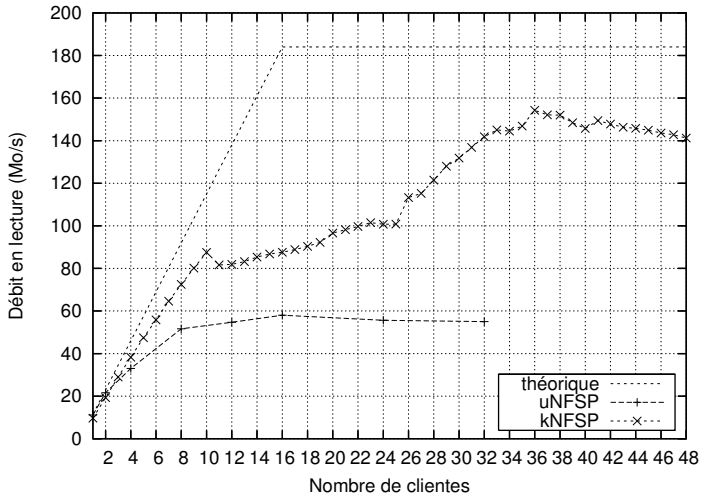

Figure 3.2 *Débit en lecture pour les deux prototypes de NFSP*

données. Les résultats obtenus ont montré une atténuation très forte par rapport au débit maximal théorique (limité par les connexions réseau selon le nombre de clients et de IODs dans chaque expérience). Une investigation détaillée du fonctionnement de ce prototype a détecté des pertes importantes de performance à cause de plusieurs copies de données entre les processus et le noyau du système d'exploitation.

L'équipe NFSP a donc décidé de porter l'implémentation dans le serveur NFS du noyau Linux, produisant kNFSP. La même évaluation de performance étant réalisée, les améliorations sont nettement visibles. La figure 3.2 présente les résultats obtenus pour les deux versions de NFSP en comparaison avec le débit théorique maximal.

Une autre expérience a été conduite en lançant des méta-serveurs sur plusieurs machines de la grappe, l'idée étant de fournir plusieurs points d'accès aux clients afin de soulager le méta-serveur unique (très chargé dans le cas des écritures, car il doit recevoir et rediriger la totalité des données écrites). Chaque méta-serveur exporte ses méta-fichiers régulièrement, mais aussi les méta-fichiers des autres serveurs, lesquels sont accédés au moyen d'un montage NFS standard.

Après la re-exécution du test de performance, en utilisant 4 méta-serveurs, on a pu constater une amélioration effective du débit de transfert, lequel a atteint jusqu'à deux fois la valeur précédente, même avec la version uNFSP. Le point faible de cette approche, toutefois, est la perte de transparence dans l'espace de nommage, car les répertoires exportés de chaque méta-serveur doivent être explicitement indiqués, comme `/meta/m0` et `/meta/m1`. Néanmoins, c'était une piste importante pour la poursuite d'activités dans le projet.

4

Le Modèle de Distribution Proposé : dNFSP

Dans le chapitre précédent, nous avons présenté les principales caractéristiques du projet NFSP, sur lequel le travail s'est basé. Comme nous l'avons montré, NFSP est en effet capable d'apporter une amélioration importante aux performances d'un système de fichiers pour grappe. Toutefois, cette amélioration ne porte que sur les opérations de lecture de données, l'écriture suivant toujours le modèle centralisé défini par NFS. Beaucoup d'applications réalisent en majorité des opérations de lecture et bénéficient donc de l'amélioration de NFSP, mais il est également désirable que le système puisse réaliser des opérations en écriture de façon optimisée. C'est dans ce but que nous avons développé cette thèse, dont la contribution principale est présentée dans ce chapitre.

Les systèmes de fichiers présentés dans le Chapitre 2 nous ont permis d'observer plusieurs techniques d'optimisation de performance. Après une période d'études réalisée en 2003 au sein du Laboratoire ID de Grenoble, correspondant au séjour en France qui fait partie de la co-tutelle, une extension à NFSP a été conçue qui permet la répartition des opérations en écriture, dont les détails sont présentés dans les pages suivantes. Il faut bien observer dès maintenant que l'extension proposé s'aligne aux principes de conception du projet NFSP, en particulier l'intrusivité minimale qui permet aux clients de rester tels quels.

Parmi les principaux objectifs de la thèse, on peut distinguer :
- aboutir à une extension du modèle NFSP qui puisse permettre l'augmentation de la performance aussi que de du passage à l'échèle à des applications plus gourmandes en opérations d'écriture de données
- permettre d'améliorer la performance d'une grappe du type Beowulf sans l'utilisa-

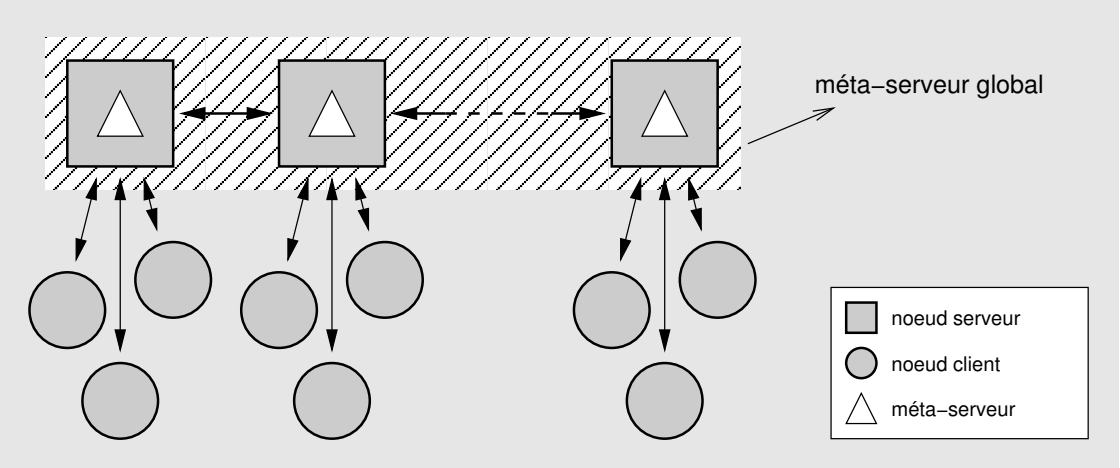

Figure 4.1 *Architecture proposé pour dNFSP*

tion des technologies coûteuses
– atteindre le délicat équilibre entre haute performance et maintien des procédures administratives de la grappe ; nous n'esperons pas atteindre la performance maximale pour le système de fichiers, mais plutôt une amélioration importante qui soit possible d'obtenir sans perdre l'esprit NFS d'utilisation et de gestion
– éviter une situation généralisé de concurrence avec d'autres systèmes et/ou interfaces d'accès à la stockage permanente, en ce qui concerne l'offre d'une solution « unique ». Par exemple, on peut citer l'interface MPI-IO, qui est probablement plus adaptée à la programmation E/S fine, mais qui par contre exige la connaissance d'une interface de programmation spécifique, tandis que le NFS peut être utilisé par l'interface POSIX standard.

4.1 Vision générale de la solution proposée

Ayant pour principal objectif l'augmentation de la performance et de la passage à l'échelle dans les opérations d'écriture de données, le modèle proposé est basé sur la distribution du méta-serveur. La figure 4.1 illustre l'architecture envisagée. La mise en place de plusieurs réplicats du méta-serveur permet la formation de groupes de clients, de façon que chaque groupe ne voie qu'un seul réplicat en tant que serveur NFS. Le nombre de clients dans chaque groupe étant inférieur au nombre total de clients, le débit demandé sur chaque réplicat est plus léger, ce qui réduit la charge sur le méta-serveur et en conséquence permet d'atteindre des meilleures performances.

Les sections suivantes détaillent le modèle proposé.

4.2 Optimisation des opérations d'écriture dans NFSP : dNFSP

Comme présenté dans le chapitre précédent, NFSP transforme le serveur NFS dans un méta-serveur, qui repasse les requêtes à des serveurs d'entrée/sortie dédiées. Bien que la lecture des données soit faite par ces serveurs, qui repassent les informations lues directement aux clients, l'écriture reste centralisée, les données devant toujours passer par le méta-serveur.

La solution la plus immédiate est donc de distribuer les points d'accès aux clients, pour offrir plus de bande passante.

Pour bien évaluer l'efficacité de cette méthode, nous avons réalisé une expérience avec le prototype NFSP originel, toutefois en faisant démarrer plusieurs copies du méta-serveur, qui accèdent toutes au même ensemble d'IODs. Cette réplication n'étant prévue dans le prototype originel, il a fallut forcer chaque client à travailler dans un sous-répertoire unique (nommé après son « hostname ») à fin d'éviter la création des fichiers de même nom.

Cette expérience a été exécuté dans la grappe i-cluster, en utilisant 8 IOD, 8 méta-serveurs et 8 clients. L'exécution consiste dans la création d'un fichier de 1 Go par chaque client. L'amélioration de l'écriture a été observé immédiatement, le débit en écriture s'élevant jusqu'aux 70 Mo/s, ce qui est bien plus performant que les 11 Mo/s que l'on peut obtenir avec un serveur centralisé.

En analysant les résultats obtenus, la conclusion est que la réplication du méta-serveur suffit pour les objectifs proposés en ce qui concerne la performance et la « philosophie » NFSP. Celui-ci a donc été choisi comme le modèle le plus adapté à la thèse. Pour bien identifier le modèle proposé, nous adoptons la dénomination *dNFSP*, avec un *d* comme *distribué*[1].

La réplication des méta-serveurs dans dNFSP a pour conséquence la possibilité de conflits d'accès aux données lorsque des clients liés à plus qu'un méta-serveur accèdent aux même fichiers. Le problème du maintien de la cohérence est l'objet de la section suivante.

[1] Ou bien *distributed*, comme on voudra

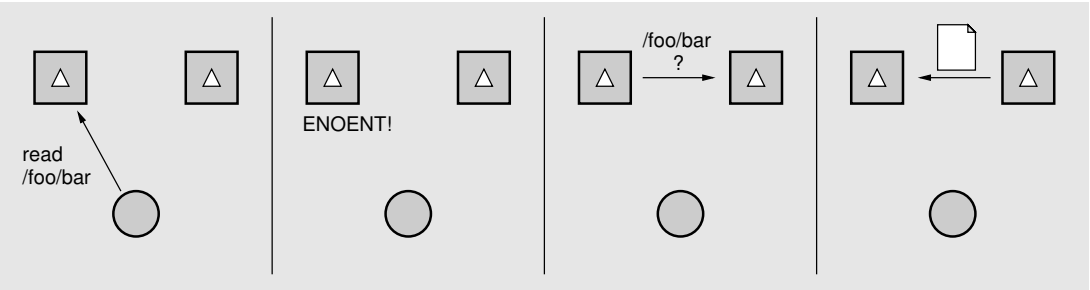

Figure 4.2 *Étapes du mécanisme de mise à jour de méta-fichiers dans dNFSP*

4.3 Maintien de la cohérence entre les méta-serveurs

Dès la simple réplication des méta-serveurs, le problème surgit dans un cas très simple : lorsqu'un client *A* crée un fichier quelconque, ce fichier ne sera pas visible sur un client *B* si celui-ci n'est pas lié au même méta-serveur. Cela vient du fait que les méta-fichiers font partie du système de fichiers local à un méta-serveur, n'étant donc visible que dans cette machine.

La solution choisie pour dNFSP se base sur le mécanisme de LRC, ou *Lazy Release Consistency* [17], classiquement utilisée dans les systèmes de mémoire partagée distribuée. Les mises à jour s'effectuent seulement en réponse à la sollicitation d'une donnée modifiée, ce qui permet d'économiser des messages. De même, dNFSP vérifie la cohérence des méta-données seulement quand un client accède à un fichier précis : le système retarde la diffusion de mises à jour de méta-données jusqu'au moment où celles-ci sont réellement nécessaires.

En plus, la vérification de cohérence n'est forcément toujours nécessaire. Le mécanisme de mise à jour de méta-données n'est déclenché que lors d'une possible erreur dans le système de fichiers, ce qui signifie que dans la plupart des cas (opérations de lecture et d'écriture) aucun procédure n'est nécessaire.

Par exemple, la figure 4.2 illustre une situation où la mise à jour est réalisé. Le client envoie au méta-serveur une requête de lecture d'un bloc de données d'un fichier quelconque. Ce fichier n'éxistant pas dans le serveur, une erreur `ENOENT` se serait normalement produite ; néanmoins, c'est plutôt une indication que le fichier a été créé dans un autre méta-serveur, et comme ça il faut récupérer le méta-fichier correspondant afin de compléter l'opération.

Dans une autre situation, on peut imaginer que deux clients (liés à deux méta-serveurs distincts) accèdent à un même fichier et que l'un des deux apporte une modification au fichier. Dans ce cas, aucune actualisation n'est nécessaire, car les modifications de données ont lieu sur les IODs, et le méta-fichier existant est toujours capable d'y accéder.

Nous envisageons au moins trois situations où l'actualisation est nécessaire :
- si le méta-fichier n'existe pas, comme nous venons d'exemplifier ;
- si le méta-fichier n'est pas à jour : par exemple, si un client essaie d'accéder au delà de la fin d'un fichier, cela peut indiquer que le fichier a été augmenté
- si l'opération exige une actualisation explicite : certains opérations comme l'effacement d'un fichier doivent être synchronisé parmi tous les méta-serveurs, afin d'éviter qu'un client puisse lire d'informations obsolètes. Bien que cette opération implique des pertes importantes, nous n'esperons pas qu'elle soit courante dans des applications parallèles

4.4 Politiques d'actualisation des méta-fichiers

Comme expliqué ci-dessus, pour actualiser un méta-fichier, le méta-serveur doit entrer en contact avec d'autres méta-serveurs et leur demander l'information nécessaire. Il doit donc déterminer quel(s) méta-serveurs contacter afin de trouver le fichier désiré. Une politique de recherche est donc nécessaire afin de minimiser le temps dépensé dans cette opération.

On distinguera ici quelques politiques de recherche :
- recherche par proximité (voisinage) : cette politique repose sur une relation de voisinage entre les méta-serveurs, en les numérotant 0 à $n-1$, de façon circulaire. La recherche commence donc par les méta-serveurs les plus proches et, si nécessaire, continue envers les plus distants. Par exemple, le méta-serveur #1 demande d'abord aux serveurs #0 et #2, puis aux serveurs #$(n-1)$ et #3, et ainsi de suite. Cette politique est plus adaptée aux cas où la connexion réseau des serveurs est plus performante que celle des clients, ce qui permet que plusieurs clients soient liés au même serveur.
- recherche hiérarchique : dans le cas où il y a un grand nombre de (méta-)serveurs, l'organisation sous forme de hiérarchie peut être intéressant pour des applications qui travaillent sur des groupes de fichiers
- recherche aléatoire : une autre possibilité est le simple choix aléatoire du méta-serveur à contacter, s'il n'y a aucun critère spécial d'allocations de noeuds et/ou des connexions réseau différents.

4.5 Considérations sur le modèle proposé

Comme présenté ci-dessus, le modèle proposé, dNFSP, vise à augmenter la performance d'une installation NFSP par la répartition du méta-serveur et la mise en place d'un mécanisme lâche de maintien de la cohérence, sans abdiquer des principes de base du projet. Les modifications apportés au modèle NFSP d'origine ont permis d'aboutir à une

proposition qui atteigne ces objectifs.

Il faut cependant insister sur le fait suivant : le modèle est dédié aux environnements de calcul haute-performance, et non à une utilisation basée sur le partage de fichiers comme d'habitude avec NFS. En conséquence, certains caractéristiques des systèmes de fichiers traditionnelles comme les « hard links » n'ont pas été considérés.

Une autre caractéristique modifiée par rapport à NFS standard est la cohérence plus lâche en ce qui concerne les méta-données. Il est néanmoins important d'observer que le propre NFS présente un mécanisme lâche de maintien de cohérence, ce qui est de la connaissance des utilisateurs.

Pour valider la proposition, une série d'expériences ont été conduites, impliquant des benchmarks et des applications réalistes. Ces expériences et les résultats obtenus, ainsi qu'une analyse de ceux-ci, font l'objet du chapitre suivant.

5

Validation de dNFSP : Implémentation et Mesures

Dans ce chapitre, nous présenterons les détails d'implémentation et les résultats expérimentaux obtenus avec un prototype du modèle dNFSP.

5.1 Implémentation sur uNFSP

Le prototype dNFSP a été implémenté à partir de la version uNFSP, dû aux raisons suivantes :

- l'implémentation étant réalisée au niveau d'utilisateur, des tâches comme lancer/arrêter des processus ou même le débogage bénéficient des facilités propres de ce niveau
- des caractéristiques existantes dans l'implémentation au niveau d'utilisateur de NFSv2, comme la réexportation de répertoires, ne sont pas disponibles dans le serveur NFS du noyau Linux
- le développement au niveau d'utilisateur permet un cycle de test/débogage plus rapide, car les modifications dans le noyau demandent souvent le redémarrage de la machine
- enfin, une investigation complémentaire des possibilités d'amélioration de performance sur la version uNFSP était intéressant pour le projet

TAB. 5.1 *Mesures de débit maximale des disques et du réseau dans les grappes utilisées*

		Disques	
Grappe	**Réseau**	**Lecture**	**Écriture**
i-cluster	11,2	42,72	35,60
LabTeC	11,1	43,22	40,63

5.1.1 Adaptations au méta-serveur

Pour bien identifier chaque méta-serveur dans le contexte dNFSP, un nouveau fichier de configuration, `dms.conf`, a été créé. Ce fichier doit contenir l'identification du méta-serveur local et une liste des adresses IP de tous les méta-serveurs, dans l'ordre correspondant à la numérotation définie.

Dans le modèle dNFSP, il faut impérativement que les méta-serveurs puissent se communiquer, afin que les uns puissent copier les méta-fichiers des autres. Le mécanisme choisi pour cette tâche est le *rcp*, par des raisons diverses : cette commande est aisément disponible dans tous les environnements Unix ; les appels à la commande peuvent être facilement intégré dans l'implémentation avec `system()` ; et la copie de fichiers avec *rcp* entraîne implicitement le maintien de plusieurs méta-données. Un problème possible avec l'utilisation de *rcp* est le coût d'exécution de cette commande ; ceci a donc fait partie de nos évaluations de performance.

En ce qui concerne les politiques d'actualisation de méta-fichiers, deux en ont été implémentées : une politique appelée *linéaire*, où la recherche commence toujours par le méta-serveur #0 s'élevant jusqu'au #$(n-1)$, et la politique par proximité. La première politique sert à mesurer des conditions de pire cas.

5.2 Mesures expérimentales

5.2.1 Disques et réseau

Ces données ont été mesures afin de définir les limites physiques de performance que pourraient être obtenues. Les expériences consistaient à la création et lecture de fichiers locaux, pour le cas des disques, et à l'exécution du « ping-pong » pour mesurer le débit maximale du réseau entre deux machines.

Deux environnements ont été considérés, correspondant aux deux grappes utilisées dans les expériences : la grappe i-cluster de l'ID/IMAG, et la grappe *labtec*, de l'Institut d'Informatique de la UFRGS. Les résultats obtenus sont présentés dans le tableau 5.1. On peut constater alors que le facteur limitant du débit pour chaque noeud est la connexion réseau.

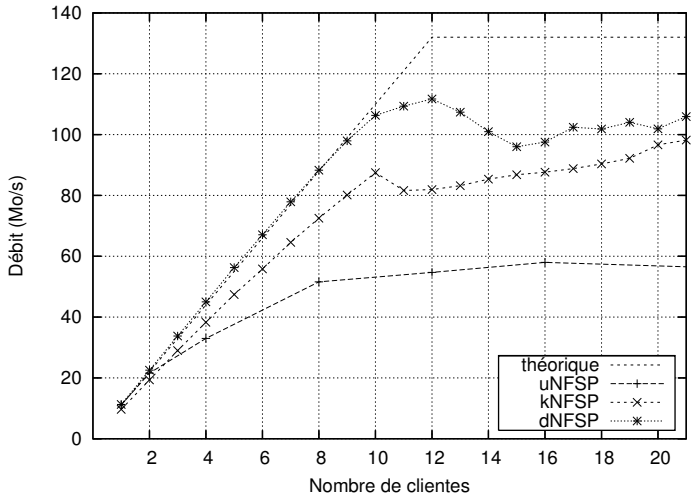

Figure 5.1 *Débit obtenu pour les opérations de lecture avec dNFSP*

5.2.2 Lectures et écritures séquentielles

Dans cette expérience, les clients réalisent, en parallèle, la création (i.e. écriture) et la lecture séquentielle de fichiers de 1 Go. L'exécution a été conduite dans la grappe i-cluster en utilisant 12 IODs, 7 méta-serveurs et 21 clients, totalisant 40 noeuds, le temps total d'exécution étant mesuré. Le débit reporté est calculé par

$$V = \frac{n_c K}{t}$$

où n_c est le nombre de clients, K est la taille du fichier (1 Go dans ce cas) et t est le temps d'exécution.

La figure 5.1 présente les résultats obtenus pour les opérations de lecture, en comparaison avec ceux des deux version de NFSP, et aussi avec le débit théorique, donné par

$$V_l = \begin{cases} n_c R & \text{si } n_c \leq n_i \\ n_i R & \text{au cas contraire} \end{cases}$$

où R est le débit du réseau (environs 11 Mo/s).

Bien que ce n'était pas son objectif, dNFSP a pu apporter une amélioration importante aux opérations de lecture par rapport aux versions uNFSP et kNFSP. Nous avons constaté que l'existence de plusieurs réplicats du méta-serveur sert a réduire la charge impliqué par un tel nombre de requêtes, et par conséquent la performance augmente.

Dans le cas des opérations d'écriture, le comportement des courbes est différent. Limitée aussi par le nombre de méta-serveurs, le débit total du réseau est fortement dégradé à chaque fois que le nombre de clients dépasse un multiple entier du nombre de méta-serveurs.

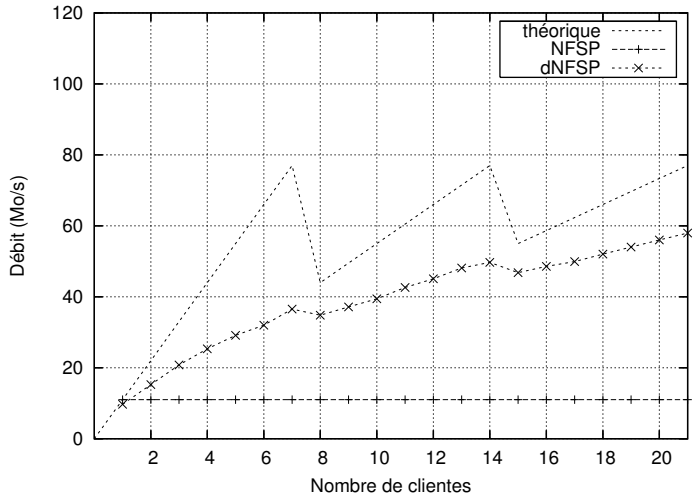

Figure 5.2 *Débit obtenu pour les opérations d'écriture avec dNFSP*

Mathématiquement, le temps total de l'écriture concurrente est donné par

$$t_e = \frac{1}{11}(((n_c - 1) \text{ div } n_m) + 1)$$

qui est le temps de transfert d'un seul fichier (1 Go / 11 Mo/s) multiplié par le nombre le plus grand de clients d'un seul méta-serveur. Le débit maximale en écriture est donc

$$V_e = \frac{n_c 1G}{t_e}$$

ou

$$V_e = \frac{11 n_c}{((n_c - 1) \text{ div } n_m) + 1}$$

Les courbes obtenues sont présentées dans la figure 5.2. L'amélioration de la performance est bien évidente, le débit s'élevant jusqu'au 60 Mo/s au lieu des 11 Mo/s dans le cas centralisé. Ce résultat confirme l'efficacité du modèle proposé dans le but d'augmenter la performance de NFSP.

Bien qu'il faille considérer la simplification du modèle mathématique appliqué, la dégradation prévue a été confirmée dans les résultats, comme observable dans les cas de 8 et de 15 clients.

5.2.3 Sur-coût de récupération de méta-fichiers

Comme nous l'avons précédemment mentionné, l'utilisation de *rcp* comme protocole de base pour la communication entre les méta-serveurs a dû être évaluée par rapport au

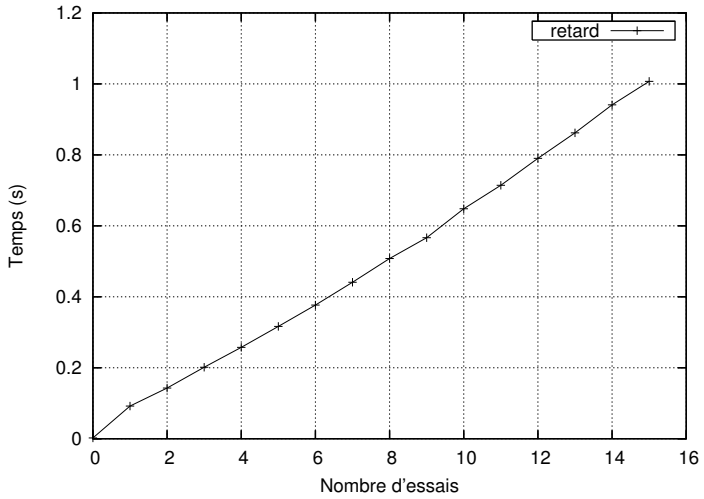

Figure 5.3 *Temps de retard d'actualisation de méta-fichiers à distance*

coût d'exécution. Une telle évaluation a donc été conduite en utilisant la politique linéaire d'actualisation de méta-fichiers.

L'expérience consiste à faire lire le contenu d'un ensemble de fichiers distants. Pour en évaluer juste le temps de récupération des méta-fichiers, la taille des fichiers (créés d'avance) est zéro.

Pour l'exécution, nous avons utilisé une configuration avec 16 clients, 16 méta-serveurs (en correspondance 1 :1) et 1 IOD seulement. Premièrement, 100 fichiers de taille nulle ont été créés sur le client #15. Ensuite, allant du client #14 jusqu'au client #0 (un client par fois), nous avons mesuré le temps nécessaire pour que chacun lise la totalité des fichiers.

Les fichiers n'existant que sur le client #15, le premier accès du client #14 déclenchera sur le méta-serveur #14 une opération d'actualisation de méta-fichier, laquelle, d'après la politique linéaire, commencera par le méta-serveur #0, et par conséquent ne trouvera le fichier que 15 essais après. Le même se produit pour chacun des 100 fichiers, et de façon similaire pour tous les clients, à l'exception du nombre d'essais qui diminue à chaque tour.

Les résultats sont présentés dans la figure 5.3. Chaque mesure a été divisée par 100 pour représenter le temps de récupération par fichier. Le comportement étant pratiquement linéaire, le retard par fichier a été mesuré à 66 ms, ce qui a été considéré acceptable par rapport au temps de transfert d'un fichier de grande taille comme utilisé dans les applications scientifiques.

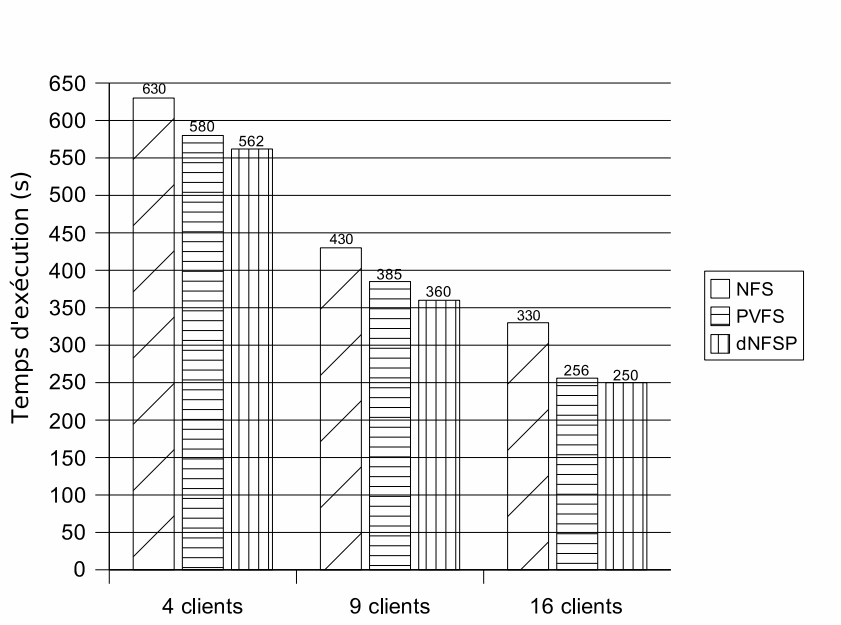

Figure 5.4 *Comparaison de performance entre dNFSP, PVFS et NFS avec BTIO*

5.3 *Benchmarks*

Nous avons également soumis le prototype de dNFSP à l'exécution de *benchmarks* traditionnellement utilisés dans l'évaluation de systèmes de fichiers, dont les résultats nous présenterons ci-après.

5.3.1 BTIO

BTIO est une adaptation du *benchmark* BT (composant du *NAS Parallel Benchmarks* [8]) pour enregistrer les résultats sur disque tout le 5ème itération. Par conséquent, il est souvent utilisé pour évaluer la performance de systèmes de fichiers parallèles.

Parmi les variantes de BTIO, nous avons utilisé *BTIO-epio*, qui n'éxige pas l'emploi de MPI-IO (qui demande à la fois NFSv3, à cause de l'utilisation de verrous). Nous avons comparé la performance de dNFSP avec celle de PVFS et de NFS (niveau utilisateur).

L'expérience a été réalisé sur la grappe Labtec en utilisant 4 noeuds pour le serveur de fichiers et 1, 4, 9 et 16 noeuds en tant que clients (BTIO demande un nombre carré de clients). La figure 5.4 montre les résultats. On peut constater la bonne performance de dNFSP dans tous les cas, s'équivalant pratiquement à celle de PVFS, et arrivant jusqu'à 32% d'amélioration par rapport à NFS.

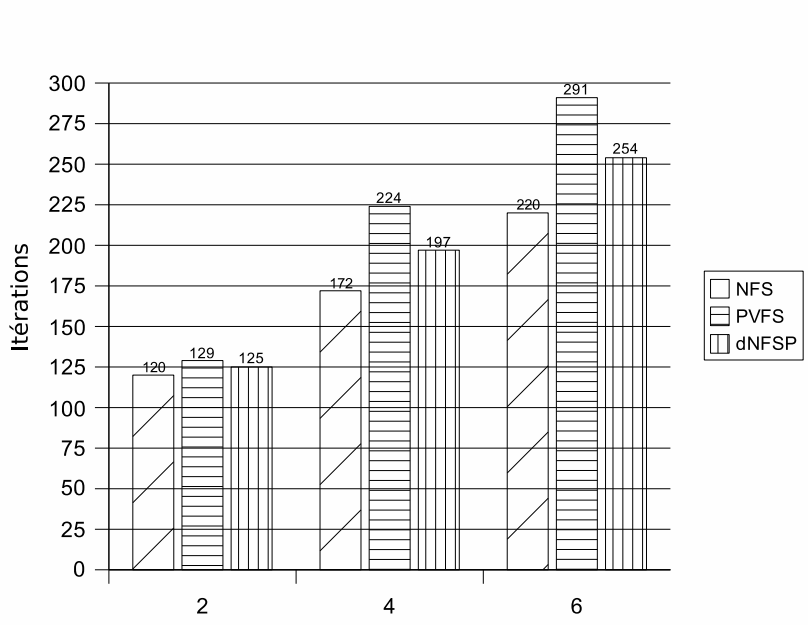

Figure 5.5 *Résultats obtenus avec GADGET*

5.4 Application : GADGET

GADGET [32] est une application de simulation cosmologique de N-corps/SPH (*Smoothed Particle Hydrodynamics*). L'objectif de son exécution n'est pas d'obtenir un résultat en tant que « valeur », mais d'observer l'évolution d'un ensemble de corps dont les forces gravitationnelles interagissent les unes avec les autres.

Pendant une exécution, il est possible de gérer des points de récupération (*snapshots*) afin de pouvoir poursuivre la simulation depuis un instant futur. La génération des points de récupération consiste dans la création de fichiers dont le contenu représente l'état des corps. Nous avons donc utilisé cette caractéristique comme forme d'évaluation de dNFSP. Le temps d'exécution étant fixé, le réflexe d'une amélioration sur le système de fichiers est la complétion d'un nombre plus grand d'itérations.

L'exécution a été conduite sur la grappe labtec, en utilisant également 4 noeuds pour le serveurs et jusqu'à 6 clients. La figure 5.5 présente les résultats, correspondant à des exécutions de 30 minutes. Encore dans cette expérience, les systèmes de fichiers parallèles ont montré meilleure performance par rapport à NFS, avec une marge d'environs 15% de gain.

5.5 Bilan

Après la réalisation de ces expériences, nous considérons que les objeticfs du travail ont été atteints. Les modifications apportées à NFSP ont effectivement permis une augmentation de performance, aussi bien dans le cas des écritures que dans celui des lectures, néanmoins sans abdiquer de la compatibilité avec les clients NFS standards et les procédures de gestion NFS en général.

6

Conclusions

L'importance du calcul parallèle avec des grappes d'ordinateurs, intéressant par des diverses raisons techniques et économiques, motive de façon constante la recherche sur des mécanismes d'amélioration de performance. La gestion d'entrée/sortie étant l'un des principaux problèmes des grappes de grande taille, le projet NFSP de l'ID propose une solution basée sur NFS afin de bénéficier de ses caractéristiques de stabilité et de procédures de gestion bien connues.

Le travail développé dans cette thèse, dNFSP, apporte une extension au modèle d'origine de NFSP, afin de permettre une amélioration de performance dans le cas des écritures distribuées. L'extension proposée s'appuie sur deux points principaux : *i*) la réplication du méta-serveur et conséquent distribution des méta-données, ce qui permet un meilleur passage à l'échelle et réduit la charge sur le serveur unique d'origine, et *ii*) un mécanisme lâche de maintien de la cohérence de méta-données, basé sur LRC, qui minimise le coût de synchronisation d'informations entre les réplicats.

Parmi les objeticfs atteints, on peut distinguer :

- Débit et passage à l'échelle plus performants dans le cas d'écriture distribuée de données, comme présenté pour le *benchmark* BTIO, avec un gain d'environs 32% par rapport à NFS, et même une amélioration sur les opérations de lecture
- Coût tolérable de maintien de cohérence, malgré l'utilisation d'un mécanisme « lourd » de transport comme *rcp*
- Maintien d'un bas niveau d'intrusivité sur les procédures d'installation, configuration et gestion d'une grappe basée sur NFS, en particulier la compatibilité avec les clients NFS standards

La confidence sur le travail développé se renforce avec la constatation que la version 2 de PVFS, lancé officiellement en novembre 2004, introduit la gestion distribuée de méta-

données, caractéristique également proposée dans cette thèse.

Parmi les contributions du travail, on distingue principalement l'investigation sur les possibilités de NFS comme un système de fichiers pour grappes et la contribution au projet NFSP de façon générale. Il faut également distinguer le renforcement de la coopération entre l'ID et la UFRGS, par la réalisation en co-tutelle de cette thèse, et par une nouvelle activité déjà établie : l'academicien Everton Hermann démarre son Master à la UFRGS avec NFSP comme sujet de travail.

Les articles suivants ont été publiés au long de la thèse :

- *CCGrid 2005*, accepté pour publication, intitulé *Evaluating the Performance of the dNFSP File System* [16]
- *SBAC-PAD 2004*, article *Performance Evaluation of a Prototype Distributed NFS Server* [6]
- *WSGPPD 2003*, intitulé *A Comparison on Current Distributed File Systems for Beowulf Clusters* [7]

Enfin, nous envisageons, comme travaux futurs : l'implémentation d'un protocole dédié de communication entre les méta-serveurs, afin de réduire encore le sur-coût d'actualisation de méta-fichiers ; l'implémentation de la tolérance aux pannes entre les IODs, pour bien garantir la correction des données ; l'intégration de dNFSP dans la version kNFSP, qui présente des améliorations implicites grâce à l'exécution au niveau du noyau ; et le passage à la version NFSv3, pour bien supporter des applications importantes comme celles basées sur MPI-IO.

Bibliographie

[1] The Berkeley NOW project, 1995. Available at : <http ://now.cs.berkeley.edu>. Access in : Jan. 2005.

[2] Free software foundation, 2005. Available at : <http ://www.fsf.org>. Access in : Jan. 2005.

[3] The Linux homepage, 2005. Available at : <http ://www.linux.org>. Access in : Jan. 2005.

[4] Open source initiative, 2005. Available at : <http ://www.opensource.org>. Access in : Jan. 2005.

[5] Thomas E. Anderson, Michael D. Dahlin, Jeanna M. Neefe, David A. Patterson, Drew S. Roselli, and Randolph Y. Wang. Serverless network file systems. In *Proc. of the 15th Symposium on Operating Systems Principles*, pages 109–126, Copper Mountain Resort, Colorado, December 1995. ACM.

[6] Rafael Bohrer Ávila, Philippe O. A. Navaux, Pierre Lombard, Adrien Lebre, and Yves Denneulin. Performance evaluation of a prototype distributed NFS server. In Jean-Luc Gaudiot, Maurício L. Pilla, Philippe O. A. Navaux, and Siang W. Song, editors, *Proceedings of the 16th Symposium on Computer Architecture and High-Performance Computing*, pages 100–105, Foz do Iguaçu, Brazil, 2004. Washington, IEEE.

[7] Rafael Bohrer Ávila, Philippe Olivier Alexandre Navaux, and Yves Denneulin. A comparison on current distributed file systems for Beowulf clusters. *Cadernos de Informática*, 3(1) :121–126, June 2003. Work presented in the 1. Workshop do Grupo de Processamento Paralelo e Distribuído.

[8] D. H. Bailey, T. Harris, W. Saphir, R. van der Wijngaart, A. Woo, and M. Yarrow. The NAS parallel benchmarks 2.0. Technical Report NAS-95-020, NASA Ames Research Center, Moffett Field, CA, December 1995.

[9] N. Boden et al. Myrinet : a gigabit-per-second local-area network. *IEEE Micro*, 15(1) :29–36, February 1995.

[10] Rajkumar Buyya, editor. *High Performance Cluster Computing : Architectures and Systems*. Prentice Hall PTR, Upper Saddle River, 1999.

[11] B. Callaghan, B. Pawlowski, and P. Staubach. *NFS Version 3 Protocol Specification : RFC 1831*. Internet Engineering Task Force, Network Working Group, June 1995.

[12] Philip H. Carns, Walter B. Ligon III, Robert B. Ross, and Rajeev Thakur. PVFS : a parallel file system for Linux clusters. In *Proc. of the 4th Annual Linux Showcase and Conference*, pages 317–327, Atlanta, GA, 2000. Best Paper Award.

[13] Cluster File Systems, Inc. Lustre : A scalable, high-performance file system, 2002. Available at http ://www.lustre.org/docs/whitepaper.pdf (July 2004).

[14] Al Geist et al. *PVM : Parallel Virtual Machine*. MIT Press, Cambridge, 1994.

[15] IEEE. IEEE 802.3z-1998 : Information technology–telecommunications and information exchange between systems–local and metropolitan area networks–specific requirements–part 3 : Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications, 1998.

[16] Rodrigo Kassick, Caciano Machado, Everton Hermann, Rafael Ávila, Philippe Navaux, and Yves Denneulin. Evaluating the performance of the dNFSP file system. In *Proc. of the 5th IEEE International Symposium on Cluster Computing and the Grid, CCGrid*, Cardiff, UK, 2005. Los Alamitos, IEEE Computer Society Press.

[17] Pete Keleher, Alan L. Cox, and Willy Zwaenepoel. Lazy release consistency for software distributed shared memory. In David Abramson and Jean-Luc Gaudiot, editors, *Proc. of the 19th Annual International Symposium on Computer Architecture*, pages 13–21, Gold Coast, Queensland, Australia, 1992. New York, ACM Press.

[18] Gene H. Kim, Ronald G. Minnich, and Larry McVoy. Bigfoot-NFS : a parallel file-striping NFS server, 1994. Available at http ://public.lanl.gov/rminnich/vecrpc/-bigfoot.ps.

[19] Wei Liu, Weimin Zheng, Meiming Shen, Xinming Ou, and Min Wu. Design and implementation of a distributed NFS server on cluster of workstations. In *Proc. of the 12th IASTED International Conference on Parallel and Distributed Computing and Systems*, pages 7–12, 2000.

[20] MPI Forum. The MPI message passing interface standard. Technical report, University of Tennessee, Knoxville, April 1994.

[21] Dan Muntz. Building a single distributed file system from many NFS servers. Technical Report HPL-2001-176, HP Laboratories, Palo Alto, July 2001.

[22] David A. Patterson, Garth A. Gibson, and Randy H. Katz. A case for redundant arrays of inexpensive disks (RAID). In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 109–116, Chicago, IL, 1988. New York, ACM Press.

[23] Kenneth W. Preslan, Andrew P. Barry, Jonathan E. Brassow, Grant M. Erickson, Erling Nygaard, Christopher J. Sabol, Steven R. Soltis, David C. Teigland, and Matthew T. O'Keefe. A 64-bit, shared disk file system for Linux. In *Proc. of the 16th IEEE Symposium on Mass Storage Systems*, pages 22–41, San Diego, California, March 1999. Los Alamitos, IEEE Computer Society.

[24] Russel Sandberg, David Goldberg, Steve Kleiman, Dan Walsh, and Bob Lyon. Design and implementation of the Sun Network Filesystem. In *Proc. of the Summer USENIX Conference*, pages 119–130, Portland, OR, USA, 1985.

[25] Mahadev Satyanarayanan. Scalable, secure, and highly available distributed file access. *IEEE Transactions on Computers*, 23(5) :9–21, May 1990.

[26] Mahadev Satyanarayanan, James J. Kistler, Puneet Kumar, Maria E. Okasaki, Ellen H. Siegel, and David C. Steere. Coda : A highly available file system for a distributed workstation environment. *IEEE Transactions on Computers*, 39(4) :447–459, April 1990.

[27] Daniel F. Savarese and Thomas Sterling. Beowulf. In Buyya [10], chapter 26, pages 625–645.

[28] Erich Schikuta and Heinz Stockinger. Parallel I/O for clusters : Methodologies and systems. In Buyya [10], chapter 18, pages 439–462.

[29] Frank Schmuck and Roger Haskin. GPFS : A shared-disk file system for large computing clusters. In *Proc. of the Conference on File and Storage Technologies*, pages 231–244, Monterey, CA, 2002.

[30] Xiaohui Shen and Alok Choudhary. Dpfs : a distributed parallel file system. In Lionel M. Ni and Mateo Valero, editors, *Proceedings of the International Conference on Parallel Processing*, pages 533–544, Valencia, Spain, 2001. Los Alamitos, IEEE Computer Society.

[31] Steve Soltis, Grant Erickson, Ken Preslan, Matthew O'Keefe, and Tom Ruwart. The design and performance of a shared disk file system for IRIX. In *Proc. of the 6th Goddard Conference on Mass Storage Systems and Technologies*, pages 41–56, College Park, Maryland, March 1998.

[32] Volker Springel, Naoki Yoshida, and Simon D. M. White. GADGET : a code for collisionless and gasdynamical cosmological simulations. *New Astronomy*, 6 :79–117, 2001.

[33] Thomas Sterling, Donald J. Becker, Daniel Savarese, John E. Dorband, Udaya A. Ranawake, and Charles V. Packer. BEOWULF : a parallel workstation for scientific computation. In *Proceedings of the 24th International Conference on Parallel Processing*, pages 11–14, Oconomowoc, WI, 1995.

[34] Thomas L. Sterling, John Salmon, Donald J. Becker, and Daniel F. Savarese. *How to Build a Beowulf : a Guide to the Implementation and Application of PC Clusters*. MIT, Cambridge, 1999.

[35] Thomas Lawrence Sterling. *Beowulf Cluster Computing with Linux*. MIT Press, Cambridge, 2002.

[36] Adam Sweeney, Doug Doucette, Wei Hu, Curtis Anderson, Mike Nishimoto, and Geoff Peck. Scalability in the XFS file system. In *Proceedings of the USENIX 1996 Technical Conference*, pages 1–14, San Diego, CA, USA, January 1996.

[37] Chandramohan A. Thekkath, Timothy Mann, and Edward K. Lee. Frangipani : A scalable distributed file system. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles*, pages 224–237, Saint Malo, France, 1997. New York, ACM Press.

Resumé Un Modèle de Distribution du Serveur de Fichiers pour Grappes

L'utilisation de grappes d'ordinateurs de grande taille comme plateforme pour le calcul parallèle trouve dans la gestion d'entrée/sortie l'un de ces principaux problèmes. L'emploi de solutions centralisés traditionnelles comme NFS forme des goulots d'étranglement qui ne sont pas acceptables sur le système de fichiers. Plusieurs solutions pour ce problème ont été proposées, aussi bien par l'utilisation de technologies comme le RAID ou la fibre optique que par la distribution des fonctions du serveur de fichiers. Suivant cette approche, NFSP est un projet qui vise à l'augmentation de la performance et du passage à l'échelle des opérations de lecture sur un serveur NFS standard tout en bénéficiant de ses aspects de stabilité et de procédures de gestion bien connues. Le modèle présenté dans cette thèse, dNFSP, est une extension de NFSP dont l'objeticf est l'amélioration de la performance des applications présentant aussi bien des opérations de lecture que celles d'écriture de données. L'extension proposée s'appuie sur deux points principaux : la gestion distribuée de méta-données, ce qui favorise le passage à l'échelle et réduit la charge sur le serveur unique d'origine, et un mécanisme lâche de maintien de cohérence basé sur LRC (*Lazy Release Consistency*), qui minimise le coût de synchronisation d'informations. Un prototype de ce modèle a été implémenté et évalué avec un ensemble de tests, *benchmarks* et applications. Les résultats obtenus confirment l'augmentation de la performance des applications sans abdiquer de la compatibilité avec les procédures standards de gestion.

Mots-clés : systèmes de fichiers, NFS, gestion de méta-données, Lazy Release Consistency et grappes de calcul

Abstract A Proposal for Distributing the File Server on Clusters

The evolution of Cluster Computing has led to the construction of each time larger and larger parallel machines, in which one of the main problems is the management of I/O, since centralised file storage solutions like NFS rapidly become the bottleneck of this part of the system. Over the last years, many solutions to this problem have been proposed, both making use of specialised hardware such as RAID and fibre optics as well as by distributing the functionalities of the file server. Following this second trend, NFSP is a proposal which extends the standard NFS server in order to provide improved read performance and scalability while taking benefit from its stability and well-known management practices. The proposal presented in this thesis, referred to as dNFSP, is an extension of NFSP with the main goal of providing better performance to applications exploring both read and write operations. The basis for the proposed system is a distributed metadata management model, which allows for better scalability and reduces the computational cost on the original NFSP metaserver, and also a relaxed mechanism for maintaining metadata coerence based on LRC (Lazy Release Consistency), which enables the distribution of the service without incurring in costly data sinchronization operations. A prototype of the dNFSP model has been implemented and evaluated by means of a series of tests, benchmarks and applications. The obtained results confirm that the model can provide better performance to applications and at the same time keep a high degree of compatibility with standard tools and procedures.

Keywords : File systems, NFS, metadata management, Lazy Release Consistency, and cluster computing