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

EVERTON HERMANN

Dinamismo de Servidores de Dados no Sistema de Arquivos dNFSp

Dissertação apresentada como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Prof. Dr. Philippe O. A. Navaux
Orientador

Porto Alegre, outubro de 2006

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Hermann, Everton

Dinamismo de Servidores de Dados no Sistema de Arquivos dNFSp / Everton Hermann. – Porto Alegre: PPGC da UFRGS, 2006.

101 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2006. Orientador: Philippe O. A. Navaux.

1. Sistema de arquivos, armazenamento, sistemas distribuídos. I. Navaux, Philippe O. A. II. Título.

“Vocês podem ter o estudo, mas eu tenho a catega.”
— SEU LAU MARQUES

AGRADECIMENTOS

Após dois anos em um novo ambiente de trabalho, é muito satisfatório perceber que por onde passamos sempre encontramos pessoas incríveis que estão sempre dispostas a nos ajudar. Desta vez, eu tive a oportunidade de reforçar grandes amizades com pessoas que conhecia anteriormente e que foram fundamentais para o resultado deste trabalho. Sem medo de cometer exageros, eu começo agradecendo ao Lucas, à Márcia e ao Righi pelas constantes leituras e incentivos, sem os quais este trabalho não teria tomado a forma atual. É grade a consideração que tenho por vocês, é tão grande que hoje eu posso chamar a Márcia de madrinha. Resumindo, vocês são "faixa dupla".

Não posso deixar de agradecer também ao professor Navaux pelas correções nos textos, e das conversas e reuniões onde pude aprender um pouco mais com a sua experiência em pesquisa. Agradeço ao Danilo e à Francieli que além de colaborarem neste trabalho, fizeram com que eu desenvolvesse novas habilidades. Com eles aprendi a ensinar e repassar o conhecimento, pude saber como é liderar um grupo de pessoas de forma que o trabalho dê bons frutos. Também quero agradecer ao Kassick, ao Bohrer e ao restante do pessoal do laboratório pelas constantes trocas de idéias que tivemos, nas noites em claro fazendo artigos e também nas noites de festividades. E ao falar em festividades é impossível não lembrar do Pezzi e do Krum, por todos os eventos e alegrias que surgiram durante este ano.

Além das pessoas que surgiram através destes quase sete anos de estudante acadêmico, existem pessoas que só por existirem já têm o seu mérito garantido. Mesmo que distantes geograficamente, e sem entender do que se tratam as coisas que eu faça, que apenas saibam que "o meu filho lida com computador", sempre têm a frase certa na hora certa, para confortar este filho desalmado que a cada ano vai mais longe de casa. Portanto, este agradecimento é pra vocês seu Jorge, o Mestre das hortaliças, e dona Ivete, a mãe mais *high-tech* do interior do estado.

Para finalizar, correndo o risco de gerar ciúmes à minha mãe, eu quero agradecer à pessoa que se tornou a mais importante na minha vida. Quem sempre foi companhia para muitas coisas, teatros, bares, ou simplesmente por estarmos juntos conversando. Hoje eu tenho o orgulho de chamá-la de minha mulher. Esta é a senhora Ana Paula Hermann, que apesar de as vezes não se dar conta, teve na sua presença uma ajuda muito importante durante o período do mestrado.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

BMI	Buffered Messaging Interface
CIFS	Common Internet File System
dNFSp	Distributed NFS Parallel
DLM	Distributed Lock Manager
ECC	Error Correction Code
FCP	Fiber Channel Protocol
IOD	I/O Daemon
LFS	Log File System
LH	Lazy Hybrid
LRC	Lazy Release Consistency
MPI	Message Passing Interface
MDS	Metadata Server
NFS	Network File System
NFSp	NFS Parallel
OBD	Object Based Disks
OST	Object Storage Targets
POSIX	Portable Operating System Interface for UniX
PVFS	Parallel Virtual File System
RAID	Redundant Array of Independent Disk
RADD	Redundant Arrays of Distributed Disks
RFC	Request for Comments
RPC	Remote Procedure Call
SAN	Storage Area Networks
SMB	Server Message Block
SSI	Single System Image
TCP	Transport Control Protocol

UDP	User Datagram Protocol
VIOD	Virtual IOD
XDR	eXternal Data Representation
XOR	Exclusive OR
XPN	Expand File System

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Um dos maiores desafios no desenvolvimento de sistemas de alto desempenho é a questão da transferência e armazenamento de grandes quantidades de dados dentro do sistema. Diferentes abordagens tentam solucionar este problema. Entre elas, tem-se os sistemas de arquivos voltados para *cluster*, como PVFS, Lustre e NFSp. Eles distribuem as funções de armazenamento entre os nós do *cluster*. Na maioria dos casos, os nós do sistema de arquivos são divididos em duas categorias: servidores de dados e servidores de metadados. Assim, fica a cargo do administrador determinar como estes servidores são dispostos dentro do *cluster*. No entanto, esta tarefa nem sempre é óbvia, pois grande parte dos sistemas de arquivos exige que os nós destinados ao sistema sejam determinados na sua instalação, sem a possibilidade de alterações posteriores. Uma má configuração inicial pode exigir a reinstalação do sistema, e o fato de não fazer esta reinstalação pode resultar em um serviço que não satisfaz às necessidades dos usuários.

O objetivo deste trabalho é propor um modelo de tratamento do dinamismo de servidores de dados em um sistema de arquivos para *cluster*. Três cenários foram estudados, e para cada um deles foram analisadas estratégias de autoconfiguração do sistema de arquivos em tempo de execução. O primeiro caso tratado foi a adição de servidores de dados por parte do administrador para expandir a capacidade do sistema de arquivos. Testes sobre este caso mostraram que, nas situações onde a distribuição de carga entre os servidores de dados é homogênea, pode-se extrair os melhores resultados do sistema. O segundo caso tratado foi a inserção por parte do usuário de servidores temporários de dados. Esta inserção tem como objetivo suprir as necessidades temporárias de algumas aplicações. Foram realizados testes comparando o desempenho de aplicações com e sem a utilização de servidores temporários. Em todos os casos, a aplicação com servidores temporários teve maior desempenho, atingindo até 20% de ganho. O último cenário tratado combina técnicas de replicação com o dinamismo de nós. Assim, foi possível manter o sistema de arquivos em funcionamento mesmo após a perda de um servidor de dados. Os resultados mostraram que a perda de servidores de dados pode resultar em desequilíbrio de carga entre servidores, comprometendo o desempenho do sistema de arquivos.

Palavras-chave: Sistema de arquivos, armazenamento, sistemas distribuídos.

Data Servers Dynamism in the dNFSp File System

ABSTRACT

One of the most important challenges to high performance systems designers is storing and transfering large amounts of data between the nodes on the system. Different approaches have been proposed to solve this storage performance problem. Cluster file systems, like PVFS, Lustre and NFSp are examples of such systems, as they distribute the functionality of a file system across the nodes of cluster, achieving a high level of parallelism and offering a larger storage space than centralized solutions. Usually the file system nodes are of two types: metadata servers and data servers. The placement of those services on a cluster is left to the cluster administrator. Such configuration is not an obvious task, as most file systems do not allow changing the configuration after the installation. A suboptimal initial configuration may result on a file system that does not fit the users need and changing such configuration may require a file system reinstall.

The objective of this work is to propose a model to treat the dynamism of data servers on a cluster file system. Three scenarios were studied and for each one we have designed suitable reconfiguration strategies. The first case has its origin on the system administrator's actions, adding or removing data servers to change the capacity of the file system. The tests have shown that with an homogeneous load distribution across the servers it was possible to obtain the best results. The second scenario treats the temporary data server insertion by the user. This case aims to provide extra storage capacity to a specified application. Tests were performed comparing applications with and without temporary data servers. On all the cases the application with temporary data server has had better performance results, reaching 20% of performance gain. The last scenario, combines replication techniques with server dynamism. This way, it was possible to keep the file system working even on data servers failure. The tests have shown that the losts of a node may result on load unbalancing on data servers, degrading the overall file system performance.

Keywords: File systems, storage, distributed systems.

1 INTRODUÇÃO

A utilização de *cluster* de computadores (BAKER; BUYYA, 1999) tem crescido muito rapidamente dentro da área de processamento paralelo. Além do aumento na demanda de processamento, um número significativo de aplicações científicas executadas neste tipo de plataforma necessitam efetuar uma grande quantidade de operações de entrada e saída em armazenamento secundário (discos, sistemas de arquivos em rede, etc.). Por isso, o desempenho das operações desse tipo é crucial, podendo ser o principal fator na determinação do tempo de execução de uma aplicação (ZHU et al., 2003).

Nesse contexto, pode-se observar que o aumento da diferença de desempenho entre processador e dispositivos de armazenamento ainda é uma realidade. Isso transforma as operações de entrada e saída (E/S) no maior custo pago por aplicações que fazem uso massiço do sistema de armazenamento. Uma das alternativas de baixo custo e que permite amenizar este problema é a utilização do armazenamento dos nós de um *cluster* através de um sistema de arquivos paralelo. Estes sistemas possibilitam agregar a capacidade de transferência de rede e de disco de vários nós. Assim, é possível oferecer uma melhoria de desempenho significativa no tratamento de arquivos.

Para tirar proveito das vantagens de sistemas de arquivos para *clusters*, surgiram vários projetos de pesquisa propondo soluções de armazenamento escaláveis e com bom desempenho. Alguns exemplos de sistemas que possuem estas características são o *Parallel Virtual File System* (PVFS) (LATHAM et al., 2004), o Lustre (SCHWAN, 2003) e o *Parallel Network File System* (NFSp) (LOMBARD; DENNEULIN, 2002). O PVFS divide as funcionalidades de armazenamento em dois tipos de nós: os servidores de armazenamento e servidores de metadados. Os metadados são informações a respeito dos dados armazenados em um arquivo, eles podem ser utilizadas para diversos fins, como para localizar os dados ou manter estatísticas sobre o acesso aos arquivos. O alto desempenho do PVFS é obtido através da distribuição dos dados entre vários nós, possibilitando o acesso paralelo a eles. O Lustre possui uma estrutura de distribuição semelhante à do PVFS. Em adição, ele oferece a possibilidade de ter nós operando em pares replicados, o que aumenta a disponibilidade do sistema.

O NFSp (LOMBARD; DENNEULIN, 2002; KASSICK et al., 2005; ÁVILA et al., 2004), desenvolvido no laboratório ID em Grenoble, França, realiza a comunicação entre cliente e servidor através de um protocolo largamente difundido no armazenamento remoto, que é o *Network File System* (NFS) (SUN MICROSYSTEMS, 1989). As funcionalidades de um servidor NFS tradicional são distribuídas entre os nós de um *cluster*. Ele armazena dados e metadados em servidores diferen-

tes, inspirado no modelo do PVFS. Porém, um dos principais pontos frágeis do NFSp é a existência de apenas um servidor de metadados, que torna-se o gargalo das operações de escrita de dados. Como uma alternativa para solucionar esta limitação, foi desenvolvida no Grupo de Processamento Paralelo e Distribuído da UFRGS uma variante do NFSp chamada dNFSp. O seu princípio é distribuir os metadados entre vários nós, e utilizar mecanismos de sincronização para manter a coerência de informações entre os servidores.

A utilização de protocolos de armazenamento bem conhecidos, como os utilizados no dNFSp, facilita a instalação do sistema de arquivos em um ambiente de *cluster*. Apenas os nós servidores precisam ser alterados, enquanto que os clientes continuam intactos. Mesmo assim, definir quais nós fazem parte do sistema de arquivos continua sendo uma tarefa complexa. Dentre os métodos de instalação, uma alternativa comumente adotada é dedicar alguns nós do *cluster* ao sistema de arquivos de forma exclusiva. Neste caso, um bom dimensionamento dos nós destinados para este fim é essencial para garantir o uso eficiente dos recursos.

Como nem todas as aplicações possuem o mesmo perfil de acesso ao sistema de armazenamento, um sistema de arquivos contento poucos nós pode ser insuficiente para as aplicações que fazem uso massiço do armazenamento. Por outro lado, quando são disponibilizados muitos nós, são consumidos recursos que poderiam ser utilizados para outros fins. Um sistema que possui uma configuração estática obriga o administrador a determinar a quantidade ideal de nós no momento da instalação, o que nem sempre é possível. Por isso, é desejável ter a alternativa de alterar a configuração do sistema de arquivos sem a necessidade de reinstalação.

O objetivo deste trabalho é explorar diferentes aspectos do dinamismo de servidores de dados em um sistema de arquivos para *cluster*. Serão estudados três casos, o primeiro deles refere-se à possibilidade do administrador redimensionar o sistema de arquivos em tempo de execução, sem a exigência de que o sistema seja reconfigurado. O segundo caso tratado é a inserção de servidores de dados temporários por parte do usuário visando um maior desempenho de armazenamento para suas aplicações. Além disso, é tratada a perda de servidores de dados devido a falhas. Este tratamento, combinado com mecanismos de replicação, pode aumentar a disponibilidade do sistema.

Um protótipo que estende as funcionalidades do dNFSp foi implementado para validar o modelo de dinamismo de servidores de dados. A avaliação do protótipo foi feita utilizando ferramentas de análise de desempenho comumente usadas em sistemas de arquivos, tais como escrita seqüencial de dados e *benchmarks* de E/S. Dessa forma, foi possível obter medidas tanto das operações básicas de entrada e saída quanto em casos semelhantes ao de aplicações científicas, onde operações de armazenamento são combinadas com operações de processamento.

O restante deste trabalho está dividido em 6 capítulos. No final de cada capítulo são apresentados os principais pontos que foram tratados no seu decorrer, sob a forma de um resumo. A organização dos capítulos deste trabalho pode ser vista a seguir.

Capítulo 2: Sistemas de Armazenamento de Alto Desempenho

Neste capítulo é apresentada uma breve introdução sobre sistemas de armazenamento, seguida pela análise do relacionamento entre esses sistemas

de arquivos e o presente trabalho. Nele é feito um comparativo entre os sistemas apresentados, destacando-se vantagens que podem ser obtidas com a incorporação da inserção dinâmica de servidores de dados em um sistema de arquivos para *cluster*.

Capítulo 3: Modelo de Dinamismo de Servidores de Dados

Neste capítulo é feita uma análise dos pontos onde é possível tirar proveito do dinamismo de servidores de dados em um sistema de arquivos distribuído. Com base nesta análise foi construído um modelo, que é o ponto principal deste capítulo. Neste modelo, procurou-se abordar situações que não são completamente tratadas pelos sistemas de arquivos apresentados no Capítulo 2. Numa primeira etapa são analisados os eventos que podem causar o dinamismo de nós em um sistema distribuído. Estes eventos são associados ao contexto do sistema de arquivos dNFSp. Para cada situação são estabelecidas estratégias e ações para que o sistema de arquivos possa reagir aos eventos de inserção e remoção de servidores de dados.

Capítulo 4: Implementação do Protótipo

Este capítulo apresenta a implementação de um protótipo do modelo apresentado no Capítulo 3. Esta implementação é baseada no sistema de arquivos dNFSp. Inicialmente é apresentada a adaptação do sistema de arquivos para permitir a inserção das novas funcionalidades. Logo após, são mostrados os detalhes da implementação de cada cenário do dinamismo de servidores de dados tratado pelo modelo.

Capítulo 5: Validação e Testes

Este capítulo apresenta a validação e avaliação do protótipo através de testes de sua utilização em diferentes situações. Inicialmente foi avaliado o desempenho das adaptações necessárias para o desenvolvimento do dinamismo de servidores de dados. Em seguida, cada um dos tipos de dinamismo foi avaliado. Foram feitos testes básicos das operações de sistemas de arquivos utilizando *micro benchmarks*. Também foram feitas avaliações que simulam o comportamento de aplicações científicas, com o intuito de testar o sistema em um cenário mais próximo da realidade.

Capítulo 6: Conclusão e Trabalhos Futuros

O objetivo deste capítulo é revisar as principais contribuições resultantes deste trabalho. Nele são discutidos o modelo, a implementação e os resultados obtidos através dos testes de desempenho. Além disso, são feitas sugestões de trabalhos futuros, que servem para dar continuidade a este trabalho, melhorando alguns de seus aspectos ou implementando novas funcionalidades.

2 ARMAZENAMENTO DE DADOS

Este Capítulo tem como objetivo servir de embasamento teórico para o restante do trabalho. Ele está dividido em três seções principais. Em um primeiro momento são apresentados os conceitos básicos de armazenamento de dados, englobando a noção de dados e metadados na organização das informações. Em seguida são discutidos os principais sistemas de arquivos relacionados com este trabalho. Para finalizar, tem-se um comparativo entre os sistemas apresentados, juntamente com a exposição do problema tratado neste trabalho.

2.1 Dados e Metadados

A estrutura lógica de sistema de armazenamento poder ser vista como uma pilha, onde uma camada superior mascara o funcionamento da inferior. Assim, pode-se considerar o nível mais baixo desta pilha como a camada de suporte físico, que é freqüentemente um dispositivo magnético. Acima dessa camada temos o suporte lógico, que abstrai o funcionamento da camada física. No caso de um disco rígido, esta tarefa é executada pela controladora da interface, que oferece às camadas superiores uma visão linear do disco. Sobre essas duas camadas podem existir outras camadas como, por exemplo, um sistema de arquivos ou um sistema de redundância RAID, adicionando mais funcionalidades ao armazenamento. Na Figura 2.1 temos uma ilustração das camadas que podem ser encontradas no armazenamento de dados desde a camada física até às aplicações (SILBERSCHATZ; GALVIN; GAGNE, 2001).

No nível mais próximo da aplicação, em grande parte dos casos, existe o sistema de arquivos. Ele é utilizado para controlar a distribuição dos dados e oferecer ao usuário meios de acesso de alto nível, tais como nome de arquivos, permissões, entre outras funcionalidades. Para proporcionar uma visão hierárquica do sistema de arquivo, utiliza-se uma estrutura de diretórios. Os diretórios, na prática, são arquivos especiais que, ao invés de possuírem dados, possuem a lista dos arquivos que fazem parte deste diretório. Outros arquivos também podem ter um significado particular para o sistema operacional. Alguns exemplos são os dispositivos, ligações simbólicas e túneis de comunicação.

Para viabilizar essa idéia de arquivos em um sistema de armazenamento, é preciso guardar outras informações, que servem para identificar os dados e manter outros atributos do arquivo. Esse tipo de informação é chamado de metadado. Eles podem conter qualquer tipo de informação considerada importante pelo sistema de arquivos. Entre as principais informações armazenadas nos metadados tem-se:

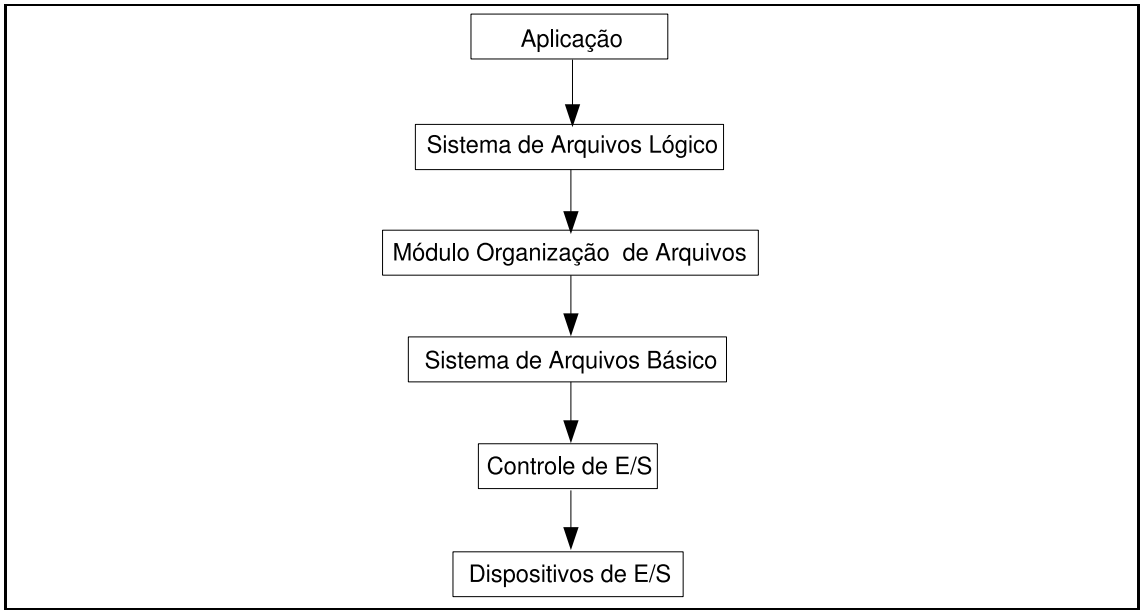

Figura 2.1: Camadas de *Software* e *Hardware* envolvidas no armazenamento de dados

- **A maneira de encontrar os dados:** pode ser desde um bloco inicial e um tamanho, até estruturas complexas descrevendo os blocos de dados utilizados;

- **O tamanho do arquivo:** é preciso armazenar o tamanho do arquivo para sabermos a posição em que os dados serão anexados quando se quiser inserir dados no final do arquivo. Além disso, nem sempre o tamanho do arquivo corresponde ao espaço ocupado por ele na camada física;

- **O tipo do arquivo:** é necessário para os sistemas de arquivos que suportam diferentes tipos de arquivos, com ligações simbólicas e túneis de comunicação;

- **Propriedades adicionais:** são utilizadas para gerenciar permissões, propriedades dos dados, versões, comentários, entre outras;

- **Dados estatísticos:** alguns exemplos são a data de criação, de último acesso e freqüência de acesso. Isso pode ser utilizado por sistemas de arquivos para otimizar o acesso aos dados, colocando os que são requisitados com mais freqüência em regiões de acesso mais rápido. Também podem ser utilizados para controle de cópias de segurança (*backup*).

Apesar de existir uma grande variedade de atributos de metadados, nem todos têm o mesmo impacto no acesso aos dados. Alguns servem de meio de obtenção de maiores informações sobre os arquivos, como data de acesso e criação. Por outro lado, outros atributos são cruciais para o acesso ao conteúdo, como a localização e o tamanho. Devido a esta forte associação entre metadados e dados, é importante otimizar o acesso e a localização destas informações para reduzir o tempo necessário para obter os dados de um arquivo. Diferentes técnicas podem ser usadas dependendo do tipo de sistema de arquivos proposto. Por exemplo,

em um sistema de arquivos em rede, deve ser levado em consideração o desempenho da transferência de dados entre os nós, para se extrair o melhor da arquitetura disponível. Considerando os conceitos básicos apresentados nessa seção, serão mostrados a seguir alguns exemplos de sistemas de arquivos para arquiteturas de alto desempenho que possuem relacionamento com este trabalho.

2.2 Trabalhos Relacionados

Com o constante uso de redes locais para interconectar computadores, tornou-se cada vez mais comum o uso de servidores para armazenar os dados dos usuários. Isso permite que os dados possam ser acessados de qualquer computador conectado à rede que possua as devidas permissões. Além disso, em sistemas de alto desempenho, as operações de entrada e saída normalmente são as operações mais onerosas (ROSARIO; CHOUDHARY, 1994). Portanto, soluções centralizadas de sistemas de arquivos em rede, projetados para estações de trabalho, podem afetar negativamente o desempenho final das aplicações.

Uma alternativa aos sistemas de arquivos em rede centralizados são os sistemas de arquivos paralelos e distribuídos. Eles possibilitam agregar a capacidade de armazenamento e de transferência de dados de vários servidores e, ao mesmo tempo, obter uma visão única do sistema. Em alguns sistemas de arquivos também é possível fracionar o conteúdo de um arquivo em diversos servidores, possibilitando o acesso paralelo a diferentes blocos de dados.

Nas seções seguintes apresentaremos os sistemas de arquivos que possuem relacionamento direto com o presente trabalho.

2.2.1 PVFS

O PVFS é um sistema de arquivos paralelo que divide as funcionalidades de armazenamento em dois tipos de nós: os Servidores de E/S (*I/O Servers*), que armazenam os dados, e o Gerente (*Manager*), que faz o papel de servidor de metadados. Um dos nós do sistema de arquivos deve ser configurado como Gerente, enquanto os demais têm a função de Servidor de E/S.

O alto desempenho do PVFS é obtido através da distribuição do conteúdo dos arquivos entre os Servidores de E/S. Esse fracionamento do conteúdo dos arquivos em diversos nós possibilita um acesso paralelo às partes disjuntas de um arquivo. Cabe ao Gerente controlar a maneira com que os arquivos são fracionados entre os Servidores de E/S. Além de ter esta função, ele é responsável por armazenar as propriedades dos arquivos e a localização dos dados. Porém, o Gerente não participa diretamente do armazenamento, nem da transferência dos dados de um arquivo. Após obter do Gerente as informações de localização e distribuição dos dados, os clientes acessam diretamente os Servidores de E/S para escrever ou ler blocos de arquivos.

Existem duas maneiras de acessar um servidor do sistema de arquivos PVFS: através de uma biblioteca ou através de um módulo específico para o sistema operacional. A utilização de um módulo permite ao usuário montar o sistema de arquivos na árvore de diretórios do sistema. Com isso, é possível acessar os arquivos no PVFS da mesma maneira que um sistema de arquivos local.

A arquitetura do PVFS utilizando um módulo no núcleo do sistema operacional é ilustrada na Figura 2.2. Nesse caso existe um processo (pvfs_client) execu-

tando no espaço do usuário em cada cliente do sistema de arquivos. Este processo é responsável por fazer o cliente se comunicar com os servidores de dados e de metadados. A parte do PVFS que executa no núcleo do sistema (VFS/pvfs_mod) serve como um meio de permitir a montagem do sistema na árvore de diretórios local. Portanto, a implementação no núcleo apenas repassa as requisições do cliente para o processo em modo usuário.

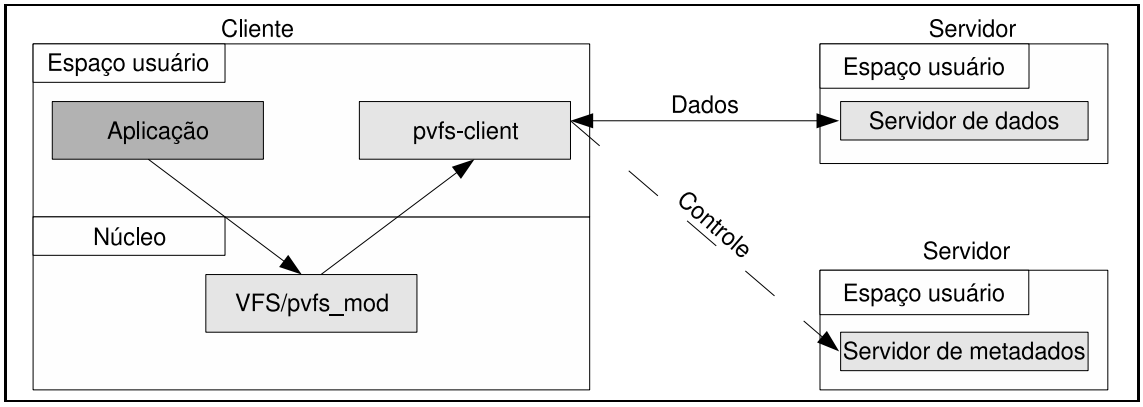

Figura 2.2: Arquitetura do sistema de arquivos PVFS

O PVFS versão 2 é uma reformulação da versão anterior e inclui novas funcionalidades ao sistema de arquivos. Uma das modificações na nova versão é o uso de protocolo sem estado entre os nós do sistema distribuído, facilitando o tratamento de falhas no sistema. Os metadados também passaram a ser distribuídos, o que evita a existência de um ponto único de falhas. Isso também ajuda a amenizar o gargalo criado pela existência de um único servidor de metadados na versão anterior.

Numa tentativa de aumentar a confiabilidade do sistema, foi incluído suporte à existência de dados e metadados redundantes. Entretanto, esta opção exige um armazenamento compartilhado entre as duas máquinas envolvidas, pois o PVFS não implementa nenhum mecanismo de replicação. Outra alteração importante foi a inclusão de um sistema modular, que permite adicionar novos padrões de distribuição de dados. Até este momento, o único padrão implementado é o *round-robin* (HAHNE, 1991).

2.2.2 NFSv2

O protocolo de armazenamento NFS (SUN MICROSYSTEMS, 1989; CALLAGHAN; PAWLOWSKI; STAUBACH, 1995; SHEPLER et al., 2003), desenvolvido pela Sun Microsystems, fornece acesso remoto transparente para compartilhamento de arquivos através da rede. Ele foi projetado para ser independente de arquitetura, sistema operacional ou protocolo de comunicação. Esta independência é obtida através do uso de Chamada Remota de Procedimento (*Remote Procedure Call*, RPC) (SRINIVASAN, 1995), e também de Representação Externa de Dados (*eXternal Data Representation*, XDR) (EISLER, 2006). Uma grande variedade de arquiteturas possuem implementações do protocolo NFS, desde computadores pessoais até computadores de grande porte. Isso faz com que o NFS seja um dos protocolos de sistemas de arquivos mais utilizado em redes locais.

O NFS versão dois, ou NFSv2, é um protocolo cliente-servidor síncrono e sem estado. Um cliente envia uma requisição, que contém tudo o que é necessário

para o seu tratamento, e espera uma resposta. A norma que especifica o protocolo NFS versão dois é o RFC 1094 e está disponível para as redes baseadas em TCP e UDP. Porém, a natureza sem estado do NFS, faz com que o protocolo UDP seja utilizado com maior freqüência.

O fato do NFSv2 ser um protocolo sem estados traz várias vantagens, principalmente em casos de expiração de tempo (*timeout*) de resposta ou de falha de um dos componentes. Quando uma requisição é perdida ou o servidor deixa de responder por um certo período, é possível reenviar a requisição até a obtenção de uma resposta. Algumas requisições idempotentes podem ser enviadas várias vezes sem a necessidade de tratamento de mensagens duplicadas.

A leitura e escrita de dados é feita de forma síncrona. Dessa maneira, o cliente envia uma requisição de escrita ou leitura e aguarda, bloqueado, uma resposta da confirmação da operação. Este tipo de comportamento, em termos de desempenho, acaba desperdiçando tempo de processamento. Durante o tempo em que está bloqueado, o cliente poderia estar efetuando operações computacionais, aproveitando de uma maneira mais eficiente os recursos disponíveis.

A simplicidade do NFS faz com que ele deixe a desejar em alguns fatores importantes. O servidor não tem meios de garantir a consistência de *cache* de conteúdo de arquivos feita pelo cliente, porque ele não mantém informações sobre o estado dos clientes. Isso significa que é possível que um cliente acesse um dado inconsistente de sua *cache* quando algum outro cliente modifica o conteúdo sem seu conhecimento.

O NFS possui um mecanismo para diminuir a probabilidade de leitura de dados inconsistentes. Ele obriga o cliente a verificar periodicamente se existem versões mais atualizadas dos dados no servidor. Tornando o intervalo das verificações pequeno o suficiente, a probabilidade de ocorrer uma leitura de dados inconsistentes passa a ser tolerável para algumas aplicações. Além disso, o NFS utiliza sincronização ao fechar o arquivo, desta forma o cliente escreve no servidor todos os dados que foram modificados entre a abertura e o fechamento.

2.2.3 NFSv3

O NFS versão 3, ou NFSv3, destaca-se por corrigir algumas limitações da versão anterior, principalmente para melhorar o desempenho. Entre as modificações é possível destacar algumas, como: suporte a operações em disco assíncronas e aumento do tamanho dos blocos de dados transmitidos.

O suporte a operações assíncronas permite ao servidor responder antes de consolidar as informações em armazenamento estável. Os dados são realmente escritos no disco quando o processo termina ou fecha o arquivo no cliente. Isso resulta no envio de uma operação especial para o servidor, chamada COMMIT. Esta chamada faz com que o cliente fique bloqueado, aguardando a consolidação das escritas, antes de continuar sua execução. Apesar de ficar bloqueado após o COMMIT, o tempo total de espera das operações de E/S assíncronas tende a ser menor que o obtido com operações síncronas.

O NFSv3 também oferece suporte a arquivos maiores. A quantidade de bits utilizada nos campos destinados a especificar o tamanho do arquivo e o deslocamento de leituras e escritas dobrou. Estes atributos passaram a conter 64 bits. Além disso o número de mensagens trocadas em um conjunto de operações foi reduzido. Com o NFSv3, a transmissão dos atributos dos arquivos é enviada

junto com as demais informações de resposta de cada operação.

2.2.4 NFSv4

Uma das principais modificações realizadas pelo NFS versão 4 (PAWLOWSKI et al., 2000; SHEPLER et al., 2003), foi a eliminação de protocolos auxiliares. Nas versões 2 e 3, um protocolo de montagem foi empregado para obter o diretório inicial do sistema de arquivos, enquanto que o tratamento de acesso exclusivo aos arquivos (*lock*) era realizado por um protocolo de gerenciamento de *lock* em rede. O NFS versão 4 define um protocolo único ao qual os sistemas de *lock* e de montagem foram completamente integrados.

Outra diferença estrutural entre o NFS versão 4 e seus predecessores é a introdução de uma chamada de procedimentos compostos, chamada COMPOUND. Este procedimento permite agrupar várias operações em uma única requisição enviada ao servidor. Ao receber uma requisição composta, o servidor agrupa as respostas de cada operação para formar a mensagem que será retornada ao cliente. O objetivo do uso de procedimentos compostos é reduzir o tempo de tratamento das operações causado por um grande número de mensagens trocadas.

A introdução de operações com estado, como OPEN e CLOSE, é uma outra diferença estrutural importante. O NFS versões 2 e 3 é essencialmente sem estado. A operação análoga à abertura de um arquivo nas versões anteriores é o LOOKUP. No entanto, esta não cria nenhum estado no servidor. A introdução de operações com estado para fechamento e abertura de arquivos possibilita ao servidor dar maior liberdade ao cliente, permitindo que este faça *cache* de dados dos arquivos de forma agressiva, além de poder gerenciar estados de *lock*.

2.2.5 pNFS

Apesar das grandes melhorias existentes no NFS versão 4, alguns problemas relativos ao desempenho continuam a existir, visto que um servidor possui rede, CPU, memória e acesso a discos limitados. O acesso a qualquer arquivo através do NFSv4 é destinado a um único servidor. Apesar do NFSv4 permitir migração entre sistemas de arquivos, ele não oferece um mecanismo que suporta múltiplos servidores exportando simultaneamente a mesma imagem do sistema. Grupos de trabalho têm estudado maneiras de paralelizar o serviço de armazenamento através de uma extensão do protocolo NFSv4 chamada pNFS.

O pNFS particiona o protocolo de sistema de arquivos em duas partes: comunicação de controle e de dados. O controle é implementado pelo servidor de arquivos NFSv4 estendido, enquanto que a transferência de dados pode ser implementada através de comunicação direta entre o cliente do sistema de arquivos e o dispositivo de armazenamento.

As novas operações do pNFS retornam descritores de acesso, chamados de *Layout*, que definem como os dados estão dispostos em um ou mais servidores. É possível que existam tipos diferentes de *Layout*, que variam de acordo com o protocolo de armazenamento utilizado para transferir os dados. Eles também podem diferir segundo o esquema de agregação, que define como o conteúdo do arquivo é distribuído entre os servidores disponíveis. Os esquemas de agregação descritos nos *Layouts* podem ser simples, como o mapeamento direto de um arquivo para um servidor ou mais complexos, como o fracionamento. O tipo de agregação padrão do pNFS é o fracionamento, que permite aos clientes acessarem

os dispositivos de armazenamento em paralelo.

Existe um protótipo do pNFS desenvolvido pela Universidade de Michigan. Esse protótipo utiliza o PVFS como base de armazenamento e comunicação (HILDEBRAND; HONEYMAN, 2005). Na Figura 2.3 é possível observar a arquitetura destes sistema. Alguns elementos foram adicionados à arquitetura padrão do NFS versão 4. Entre eles, um gerenciador de *Layout*, um gerenciador de E/S e uma interface para obtenção do *Layout*.

Ao tratar uma requisição, o servidor pNFS obtém o mapa da disposição dos dados do arquivo no sistema e o transfere para o cliente pNFS. Este, por sua vez, o transmite para o seu gerenciador de *Layout*, para que assim, os dados possam ser acessados diretamente. Os dados transferidos utilizando pNFS são transmitidos diretamente entre os clientes e os servidores de dados. No caso da versão original do NFSv4, os dados devem trafegar para o servidor central e dele serem transmitidos para os servidores de dados.

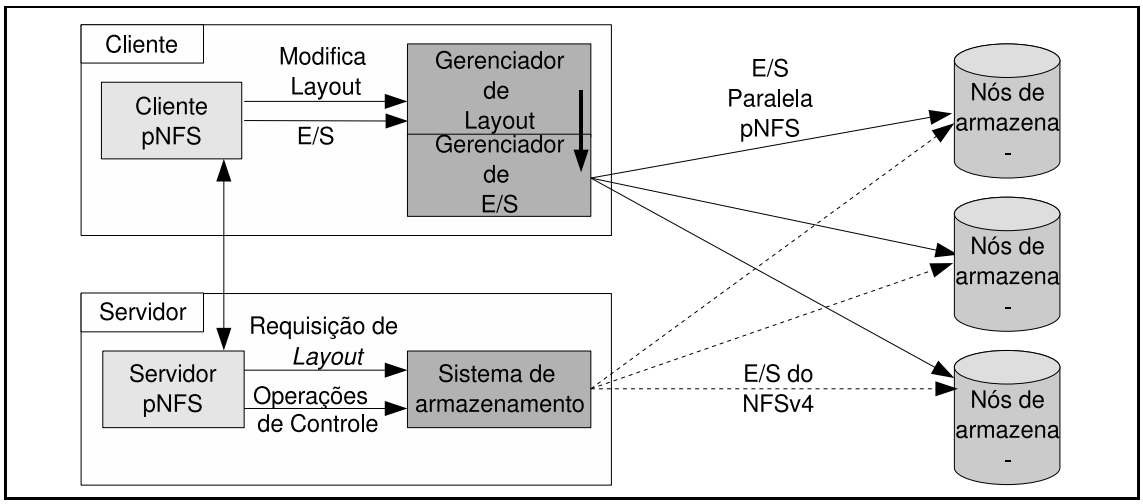

Figura 2.3: Arquitetura de um protótipo do pNFS utilizando o PVFS

2.2.6 NFSp

O NFSp (LOMBARD, 2003; LOMBARD et al., 2003) é um sistema de arquivos paralelo que implementa o protocolo NFS. Seu principal objetivo é oferecer acesso paralelo aos dados sem a necessidade de modificar a instalação do sistema nem o funcionamento do protocolo. Na instalação do sistema de arquivo NFSp, os clientes necessitam apenas de um sistema operacional que possua uma implementação do protocolo NFS tradicional. Os clientes de um *cluster* que utilizam o NFSp não precisam ser alterados em nenhum aspecto para acessar os dados.

O ganho de desempenho do NFSp é obtido distribuindo as funções de um servidor NFS. Como no PVFS, os nós envolvidos no serviço do NFSp são divididos em dois tipos: os servidores de entrada e saída, também chamados de IODs, e o servidor de metadados, chamado de metasservidor.

A técnica utilizada no NFSp para armazenar o conteúdo dos arquivos nos IODs é o fracionamento de dados, que consiste em subdividir os arquivos em blocos, armazenando cada um destes blocos em IODs distintos. O algoritmo de distribuição utilizado é o *round-robin*, onde a distribuição é feita de forma circular entre os IODs. No NFSp os blocos de dados são de tamanho fixo, o qual é

definido no momento da inicialização do sistema. O metasservidor é executado em apenas um dos nós, que do ponto de vista dos clientes é um servidor NFS tradicional. Apesar da informação estar distribuída entre os IODs, os clientes não têm conhecimento de que o sistema de arquivos possui vários servidores. Todas as operações de acesso ao sistema de arquivos feitas pelos clientes são enviadas ao metasservidor.

Apenas o metasservidor conhece a maneira com que os dados estão distribuídos entre os servidores de dados. Ao receber uma requisição de leitura de um bloco de dados, ele analisa os metadados associado ao arquivo desejado e obtém as informações de localização dos dados. Com esta informação ele repassa a requisição ao IOD responsável pelo bloco em questão. O IOD, ao receber uma solicitação de leitura de dados, acessa a informação diretamente do seu armazenamento local e constrói uma mensagem de resposta ao cliente. A mensagem enviada pelo IOD ao cliente terá como remetente o metasservidor, escondendo do cliente a existência dos demais nós. Este tipo de técnica de manipulação do remetente da mensagem é chamada de *spoofing*.

No caso acima, os dados são transmitidos apenas entre o IOD e o cliente. O único tipo de mensagem que trafega entre o cliente e o metasservidor é o de controle. Nas operações de escrita, os dados são transmitidos do cliente para o metasservidor, que do ponto de vista do cliente é quem oferece o serviço NFS. Após receber os dados do cliente, o metasservidor os repassa aos IODs. Esta retransmissão de dados acaba agregando um custo maior às escritas quando compradas às leituras. Em conseqüência disso, a existência de um único metasservidor transforma-se num gargalo de escrita de dados para o sistema.

2.2.7 dNFSp

O dNFSp (KASSICK et al., 2005; ÁVILA et al., 2004; ÁVILA, 2005) é um projeto desenvolvido no Instituto de Informática da Universidade Federal do Rio Grande do Sul (UFRGS) em conjunto com o laboratório *Informatique et Distribution* (ID) associado ao *Institut d'Informatique et Mathématiques Appliquées de Grenoble* (IMAG) com o objetivo de melhorar o desempenho da escrita de dados. Ele procura eliminar o gargalo causado pela existência de um único metasservidor no NFSp. Para isso, o NFSp foi expandido para possibilitar o uso de vários servidores de metadados. A associação entre clientes e servidor de metadados continua a mesma, visto que cada cliente conhece apenas um servidor de metadados. Porém é possível distribuir os clientes entre os vários servidores de metadados e assim dividir entre os metasservidores a carga das operações realizadas pelos clientes.

A sincronização de metadados empregada pelo dNFSp implementa uma adaptação do modelo de consistência relaxada chamado LRC (*Lazy Release Consistency*) (KELEHER, 1995). Neste tipo de algoritmo, as alterações feitas por um elemento não precisam ser propagadas imediatamente aos demais. Quando um outro componente do sistema deseja acessar algum dado que ele ainda não possui, é preciso contatar os demais até encontrar os metadados e assim obter a informação necessária. A versão adaptada do LRC permite a existência de nós que armazenem uma versão desatualizada dos dados, fazendo com que o sistema não esteja sempre sincronizado.

Na adaptação do LRC utilizada pelo dNFSp, a ocorrência de sincronização é menor, se comparado com o modelo LRC original. No momento da criação

de um metadado, o metasservidor armazena uma cópia local dos metadados. Quando um outro metasservidor necessita acessar o arquivo, é preciso solicitar a informação a todos os metasservidores até encontrar os metadados. Uma vez obtidos os metadados, os demais metasservidores não são mais contatados em acessos futuros.

Toda informação envolvida na obtenção do conteúdo de um arquivo é encontrada nos metadados. Com o passar do tempo, algumas informações armazenadas em um metasservidor podem divergir dos demais, sem que isso altere a localização dos dados. Alguns exemplos são os dados estatísticos e permissões. Para um grande número de aplicações, este nível de sincronismo é suficiente para manter o sistema funcionando corretamente. A necessidade de pouca troca de mensagens minimiza a influência do mecanismo de comunicação no desempenho final do sistema.

O mecanismo de transferência de dados empregada pelo protótipo do dNFSp para sincronizar os metadados é o rcp/rsh, que realizam, respectivamente, cópia remota de arquivos e *shell* remoto. Para sincronizar os metadados de dois servidores é preciso executar um comando rcp especificando o metarquivo a ser copiado remotamente. No metarquivo são armazenados os metadados de um determinado arquivo. De forma semelhante, quando é necessário remover ou renomear um arquivo, o metasservidor deve executar um comando rsh comunicando a mudança a todos os metasservidores do sistema. Como o dNFSp utiliza um algoritmo baseado em sincronização relaxada, o efeito de usar rsh não é tão visível no desempenho geral. Testes realizados e publicados anteriormente mostram que o dNFSp pode se equiparar aos demais sistema de arquivos em termos de desempenho, mesmo utilizando o rsh (KASSICK et al., 2005; HERMANN et al., 2006).

A Figura 2.4 mostra a arquitetura contendo vários servidores de metadados no dNFSp. O cliente envia uma requisição de leitura do NFS para o metasservidor ao qual está associado. O metasservidor passa a requisição ao IOD. Ao recebê-la o IOD envia os dados diretamente ao cliente.

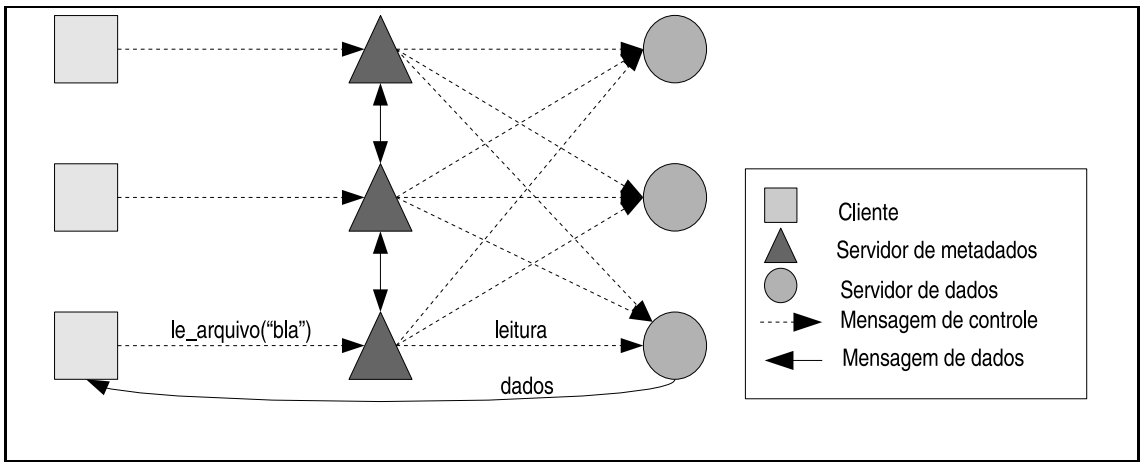

Figura 2.4: Arquitetura do sistema de arquivos dNFSp, composta por clientes, servidores de metadados e servidores de dados

2.2.8 Zebra

O sistema de arquivos Zebra (HARTMAN; OUSTERHOUT, 2001) distribui os dados dos arquivos em diversos servidores, assegurando que a perda de um único servidor não afete a disponibilidade dos dados. O esquema de fracionamento utilizado pelo Zebra é semelhante ao do RAID nível 5, onde um disco armazena o bloco de paridade enquanto que os outros armazenam os dados. O bloco de paridade é obtido a partir de uma operação XOR (*Exclusive OR*) dos blocos de dados. A cada ciclo de distribuição de dados é escolhido outro disco para armazenar a paridade, o que faz com que a carga de acesso seja distribuída de forma uniforme entre os servidores. Para contornar o problema de escritas de dados pequenos, existente no RAID 5, o sistema de arquivos Zebra utiliza *logs* estruturados para agrupar vários arquivos e os escreve em uma única operação. Desta forma é possível economizar operações de escrita através da agregação de arquivos e também amenizar o problema do desempenho do cálculo de paridade em operações menores que o tamanho do bloco.

O gerenciamento do *log* estruturado e de agregação é feito pelo cliente. Quando um cliente contém dados suficientes para fazer uma escrita, eles são transmitidos para o servidor e o *log* é esvaziado. Para que se mantenha a coerência de dados durante o período em que estão armazenados no cliente, existe um gerenciador central de arquivos. Ele gerencia os metadados e supervisiona a interação entre os clientes. Esse gerenciador também é responsável por tratar todas as operações de arquivos que envolvem metadados. Os dados dos arquivos são armazenados nos servidores de armazenamento, enquanto que o *File Manager* armazena apenas os metadados. Além de gerenciar os *logs* estruturados, o cliente é responsável pelo fracionamento dos dados entre os servidores de dados, determinando os servidores que devem armazenar os dados.

2.2.9 xFS

O Berkeley xFS (ANDERSON et al., 1995) apresenta uma abordagem de sistema de arquivos "sem servidor". Sua estrutura é completamente distribuída, os dados e metadados são difundidos entre as máquinas disponíveis e possuem a capacidade de migrar entre elas. O sistema é composto de quatro tipos de entidades: servidores de armazenamento, gerenciadores, limpadores e clientes. Os servidores de armazenamento são servidores de E/S, responsáveis pela leitura e escrita de blocos de dados nos discos. Os gerenciadores são servidores de metadados e os clientes são as entidades que usam o sistema de arquivos. Por fim, os limpadores exercem a função de defragmentar e reciclar o espaço de armazenamento de modo a prover espaço para futuras operações de escrita.

O xFS também foi projetado para suportar a inclusão em tempo de execução de novos nós ao sistema. Quando uma nova máquina se junta ao sistema, o xFS pode adicioná-la ao mapa de gerenciadores. Cada nó do sistema possui um mapa de gerenciadores que mantém as informações necessárias para encontrar os dados de um determinado arquivo. A cada mudança na configuração do sistema de arquivos este mapa deve ser redistribuído entre os nós, para que os arquivos possam ser acessados corretamente. A funcionalidade de reconfiguração não foi implementada nos protótipos desenvolvidos pela equipe do xFS. Porém ela faz parte do modelo proposto pelo sistema de arquivos.

2.2.10 Vesta

O sistema de arquivos Vesta (CORBETT; FEITELSON, 1996) foi inicialmente projetado para o computador da IBM chamado Vulcan, com o objetivo de explorar o acesso paralelo oferecido por esta arquitetura. A distribuição dos dados entre os nós é específica para cada arquivo. Isso permite ao usuário determinar o número de nós e a granularidade no momento da criação do arquivo.

Uma das principais inovações do Vesta é a possibilidade de dividir os dados de um arquivo em particionamentos lógicos disjuntos. A disposição do particionamento lógico é especificada na abertura do arquivo, isso permite que um mesmo arquivo possa ser acessado pelos nós com uma distribuição lógica diferente a cada abertura. Por exemplo, acessando o arquivo como uma abstração de matriz, onde o particionamento lógico pode representar linhas, colunas ou blocos de dados.

O acesso aos metadados é feito de forma direta através do cálculo de uma função de *hash* sobre o nome do arquivo para determinar o processador que contém os metadados. Nas operações de escrita e leitura não existe necessidade de acessar os metadados, pois o sistema de arquivos garante que o cliente possui localmente toda a informação necessária para acessar os dados. Todas estas vantagens são obtidas devido à forte integração do Vesta com a arquitetura do computador Vulcan, o que torna o sistema de arquivos pouco portável.

2.2.11 Expand

O sistema de arquivos Expand (XPN) (GARCIA-CARBALLEIRA et al., 2003) combina vários servidores NFS para oferecer um particionamento distribuído de arquivos, como mostra a Figura 2.5. Esta solução não inclui alterações no servidor NFS. Ele implementa uma biblioteca que oferece uma interface POSIX (GALLMEISTER, 1995) ao programador, contendo funções que realizam abertura, leitura, escrita e fechamento de arquivos, entre outras funções. Desta forma, não se trata realmente de um sistema de arquivos que pode ser montado em um diretório da estrutura hierárquica do sistema. Seus dados não são acessados da mesma maneira que um arquivo local. Para o seu uso, o programador deve utilizar a interface de programação oferecida pelo Expand através de sua biblioteca de acesso a arquivos.

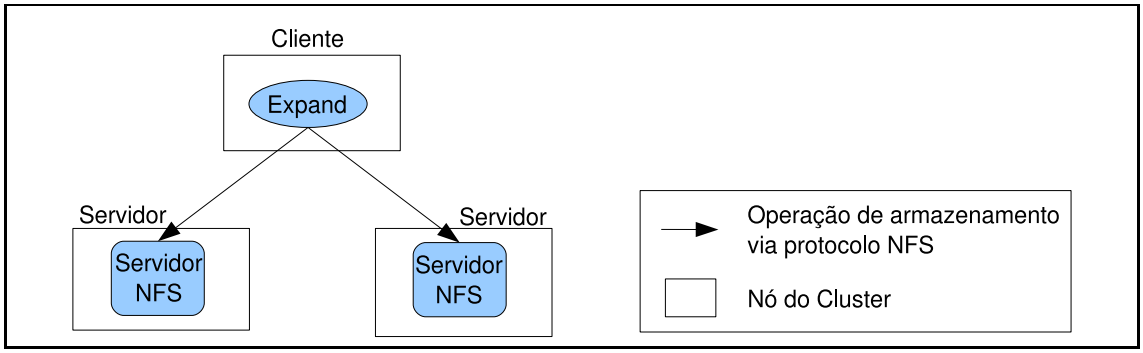

Figura 2.5: Funcionamento do sistema de arquivos Expand utilizando vários servidores NFS

Um arquivo no sistema Expand é composto de vários sub-arquivos. Cada um destes sub-arquivos é armazenado em um servidor NFS do sistema. Esta divisão

em sub-arquivos é transparente ao usuário, e ele acessa as informações como se estivessem armazenados em um único arquivo. O particionamento distribuído permite ao usuário criar vários arquivos que possuem algoritmos diferentes de distribuição, como por exemplo, arquivos particionados de forma cíclica ou arquivos redundantes utilizando RAID 4 ou RAID 5.

Cada sub-arquivo de um arquivo Expand contém um pequeno cabeçalho em seu início, onde ficam os metadados. Apesar dos metadados do arquivo estarem em todos os sub-arquivos, apenas um deles armazena os metadados atuais. Este nó, chamado de nó mestre, é determinado através de uma função de *hash* sobre o nome do arquivo, como utilizado pelo Vesta, descrito na seção anterior.

Uma das funcionalidades projetadas para o Expand é permitir o redimensionamento do sistema de arquivos através da inserção de novos servidores. Porém para que os dados estejam adequados à nova configuração é necessário redistribuí-los quando um novo servidor é inserido. Apesar de fazer parte dos projetos, até o momento não existem publicações onde constam resultados da implementação desta funcionalidade.

2.2.12 Lustre

O sistema de arquivos Lustre (SCHWAN, 2003) foi projetado com o objetivo de remover os gargalos tradicionalmente encontrados em sistemas de arquivos para *cluster*. Sua arquitetura é dividida em diferentes tipos de serviços, como mostra a Figura 2.6. Os servidores de metadados (MDSs, *Metadata Servers*) contêm o esquema de diretórios do sistema de arquivos, permissões e atributos estendidos para cada objeto. Os OSTs (*Object Storage Targets*) são responsáveis pelo armazenamento e a transferência dos dados de um arquivo. Cada OST executa uma instância do Gerenciador de *Lock* Distribuído (*Distributed Lock Manager*, DLM) que é responsável pelo controle de consistência dos dados. Tanto OSTs quanto MDSs podem operar em pares (*active / failover*) onde o nó *failover* automaticamente ocupa o lugar do ativo em casos de falha (CLUSTER FILE SYSTEMS, INC, 2002), podendo assim existir versões replicadas tanto de MDSs quanto de OSTs.

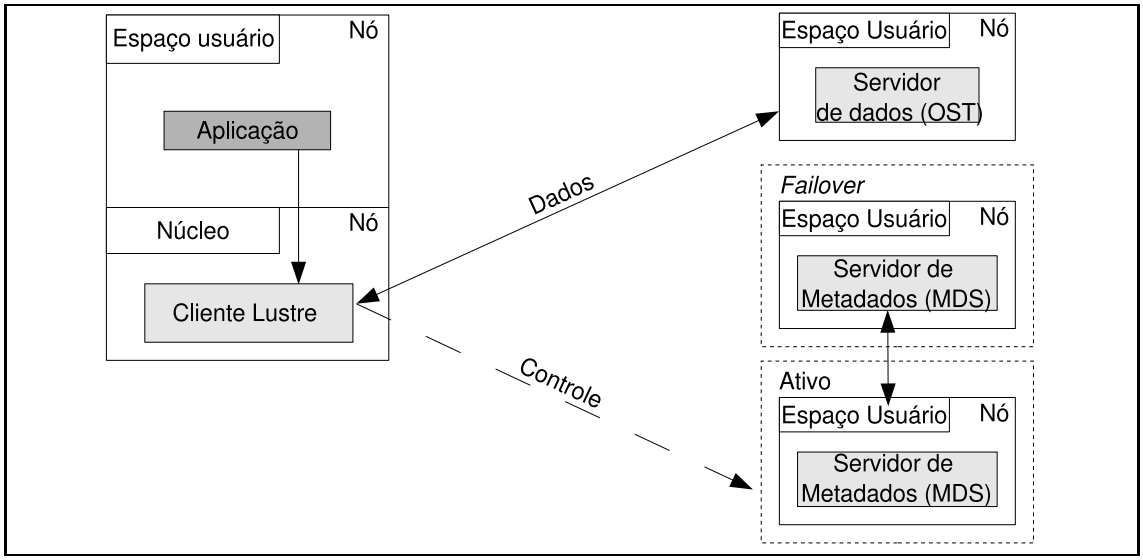

Figura 2.6: Arquitetura do sistema de arquivos Lustre com a existência de nós atuando em pares: *Failover* e Ativo

Para acessar os dados de um determinado arquivo, o cliente contata o servidor de metadados e assim ele obtém informações sobre a localização do arquivo. Uma vez obtida a localização dos dados, o cliente pode contatar diretamente o OST para ler ou escrever dados. Desta forma o servidor de metadados não faz parte do processo de transferência de dados. Acessos subseqüentes ao servidor de metadados podem ocorrer com o objetivo de alterar alguns atributos do arquivo. Um exemplo é quando o cliente deseja anexar conteúdo ao final do arquivo, alterando o seu tamanho. Para que o tamanho do arquivo seja modificado e visível para os demais clientes, é preciso notificar o servidor de metadados que tal modificação ocorreu.

2.3 Análise dos Sistemas Apresentados

O uso de um sistema de arquivos voltado para *cluster* permite um aumento da vazão disponível aos clientes. Isso ocorre pois a maioria deles oferece um tratamento paralelo das operações de armazenamento. Apesar de existirem diferentes arquiteturas de sistemas de arquivos para *cluster*, a maioria delas possui algumas características em comum. A separação de dados e metadados é feita em grande parte dos sistemas, como Lustre, Zebra, PVFS e NFSp. Isso permite modelar um servidor de dados relativamente simples e rápido, enquanto que operações mais complexas, envolvendo atributos de arquivos e permissões, são tratados pelo servidor de metadados. O algoritmo de distribuição de dados utilizado na maioria dos casos é o fracionamento (*striping*). Ele divide o conteúdo dos arquivos entre os nós de armazenamento em frações de tamanhos iguais.

Na tabela 2.1 podemos ver algumas características dos sistemas de arquivos apresentados neste trabalho. Três aspectos foram colocados em evidência. O primeiro deles é a intrusão no momento da instalação. Neste contexto, a intrusão diz respeito às modificações necessárias no núcleo do sistema operacional. O segundo aspecto é a possibilidade de redimensionar e expandir o sistema de arquivos após a instalação. E finalmente foi avaliado se os sistemas de arquivos implementam algum método de tolerância a falhas.

Tabela 2.1: Comparação dos sistemas de arquivos distribuídos relacionados

Sistema de Arquivos	**Intrusão**	**Servidor de Metadados**	**Redimensionamento**	**Tolerância a Falhas**
NFS	Nenhuma	Único	Não	Não
dNFSp	Nenhuma	Múltiplos	Não[1]	em Implementação[1]
PVFS	Média	Múltiplos	Não	em Implementação
Lustre	Alta	Múltiplos	Não	Replicação
Zebra	Média	Único	Não	RAID
Expand	Nenhuma	Múltiplos	Planejado	Não
xFS	Alta[2]	Múltiplos	Sim	Não
Vesta	Alta[2]	Múltiplos	Sim	Não

[1]Com o desenvolvimento deste trabalho estas funcionalidades passaram a ser implementadas pelo dNFSp

[2]Os sistemas de arquivos Vesta e xFS foram projetados para sistemas específicos, Vulcan e Solaris respectivamente, portanto não existe implementação para o núcleo Linux

Parte dos sistemas de arquivos avaliados possui um nível de intrusão elevado no momento da instalação. Alguns sistemas de arquivos, como o Lustre, exigem a recompilação do núcleo do sistema operacional para serem instalados. Enquanto outros, por se basearem em protocolos bem difundidos, não exigem alterações nos clientes para o seu funcionamento. O NFSp é um exemplo deste tipo de sistema de arquivos, pois não modifica a semântica do protocolo de comunicação, fazendo com que clientes NFS possam fazer uso de um servidor de arquivos distribuído sem a necessidade de serem alterados.

Com o aumento do número de nós envolvidos no sistema de arquivos, a probabilidade de ocorrerem falhas no sistema também cresce. Para garantir o funcionamento do sistema é preciso tomar medidas para implementar tolerância a falhas no sistema. Entre os sistemas de arquivos apresentados que se preocupam com tolerância a falhas, a maioria utiliza mecanismos baseado em RAID para manter dados redundantes.

O tipo de configuração do sistema de arquivos é outra característica importante na sua manutenção ao longo do tempo. Uma configuração estática nem sempre será adaptada a todas as situações. Se são disponibilizados muitos nós para o sistema de arquivos, serão consumidos recursos que poderiam ser utilizados para outros fins. Se o sistema de arquivos for distribuído em um número pequeno de máquinas, é possível não estar disponibilizando o grau de paralelismo necessário às aplicações que façam uso massiço do armazenamento.

Entre os sistemas de arquivos apresentados, alguns levam em consideração o dinamismo de nós. Grande parte destes o fazem através do tratamento de falha de nós, aumentando assim a disponibilidade do sistema. Outros consideram a inserção de novos nós para redimensionar o sistema de arquivos quando o administrador achar necessário. Porém estes dois aspectos nem sempre são suficientes. Mesmo podendo redimensionar o sistema, o administrador tem dificuldades em prever ou medir as necessidades dos usuários. A diversidade de aplicações executadas na arquitetura de *cluster* faz com que coexistam aplicações com perfis diferentes no que diz respeito às necessidades de armazenamento.

Nos demais capítulos deste trabalho pretende-se tratar três aspectos diferentes do dinamismo de servidores de dados. A exemplo de outros sistemas de arquivos, são utilizadas técnicas de replicação e detecção de falhas para suportar a perda de nós. Também será levada em consideração a expansão do sistema de arquivos controlada pelo administrador, podendo assim redimensionar o sistema sem a necessidade de reinstalação e restauração de dados. Por fim, para suprir as necessidades específicas de cada aplicação, serão desenvolvidos mecanismos que permitem aos usuários do *cluster* especificarem suas necessidades quanto ao sistema de armazenamento. Com isso, os usuários podem adaptar o sistema de arquivos aos requisitos de suas aplicações.

2.4 Resumo do Capítulo

Em sistemas de arquivos paralelos, a inserção e remoção dinâmica de servidores de dados é considerada e disponibilizada por algumas implementações. Os principais motivos são expandir o sistema e suportar tolerância a falha. Entre os exemplos de sistemas de arquivos que fazem esse tipo de tratamento temos o xFS, o Expand, o Zebra e o Lustre.

O dinamismo de nós no sistema de arquivos xFS é suportado através do uso de um mapa de gerenciadores. Ele está presente em todos os nós do sistema e mantém as informações necessárias para encontrar os dados de um determinado arquivo. Para que os arquivos possam ser acessados corretamente, a cada mudança na configuração do sistema de arquivos este mapa deve ser redistribuído entre os nós. A funcionalidade de reconfiguração não foi implementada nos protótipos desenvolvidos pela equipe do xFS, porém faz parte do modelo proposto pelo sistema de arquivos.

O sistema de arquivos Expand combina vários servidores NFS para oferecer um particionamento distribuído de arquivos. Esta solução não inclui alterações no servidor NFS. O Expand é implementado através de uma biblioteca que oferece uma interface de programação ao programador. Uma das funcionalidades projetadas para o Expand é permitir o redimensionamento do sistema de arquivos através da inserção de novos servidores. No entanto a redistribuição dos dados não foi implementada até o momento.

O sistema de arquivos Lustre tem sua arquitetura dividida em dois tipos principais de serviços. Os servidores de metadados (*Metadata Servers*, MDS) contêm o esquema de diretórios do sistema de arquivos, permissões e atributos estendidos para cada objeto. Os *Object Storage Targets* (OST) são responsáveis pelo armazenamento e a transferência dos dados de um arquivo. Tanto OSTs quanto MDSs permitem a existência de réplicas que funcionam em pares (*active/failover*) (CLUSTER FILE SYSTEMS, INC, 2002) e podem dinâmica e automaticamente ocupar o lugar um do outro em casos de falha.

O sistema de arquivos Zebra distribui os dados dos arquivos em diversos servidores, assegurando que a perda de um único servidor não afeta a disponibilidade dos dados. O esquema de fracionamento utilizado pelo Zebra é semelhante ao do RAID nível 5, onde um disco armazena o bloco de paridade enquanto que os outros armazenam os dados. O bloco de paridade é obtido a partir de uma operação booleana "ou exclusivo" (XOR) dos blocos de dados. Através do uso desta técnica, o Zebra pode suportar a perda de nós, visto que existem dados redundantes.

Decorrente das carências e das funcionalidades existentes em outros sistemas de arquivos, será apresentado no próximo capítulo um modelo de dinamismo de servidores de dados voltado a sistemas de arquivos para *cluster*. Entre os características consideradas está a inserção de servidores temporários por parte do usuário, que não é implementada pelos sistemas de arquivos para *cluster* mais populares. Além disso, foram previstos tratamentos para situações já levadas em consideração pelos sistemas de arquivos estudados neste trabalho. Como por exemplo, tolerância a falhas e redimensionamento do sistema por parte do administrador.

3 MODELO DE DINAMISMO DE SERVIDORES DE DADOS

Grande parte dos sistemas de arquivos para *cluster* oferecem uma configuração estática. Assim, os nós disponíveis para o armazenamento e a função de cada um deve ser definida no momento da instalação do sistema. Modificações posteriores exigem que o sistema de arquivos seja reinstalado e, possivelmente, que os dados sejam restaurados de forma não automática. Este tipo de restrição obriga o administrador a dimensionar o sistema de arquivos no momento da instalação de forma que ele continue a satisfazer as necessidades do usuário por um longo período de tempo. Porém nem sempre é possível encontrar uma solução ótima nestes casos.

A dificuldade em determinar o número ideal de nós destinados ao sistema de arquivos transforma a configuração estática em uma restrição que delimita o melhor uso dos recursos disponíveis. Para amenizar os problemas causados por uma configuração estática, é proposto neste trabalho um modelo que apresenta pontos onde se pode tirar proveito do dinamismo de servidores de dado em um sistema de arquivos para *cluster*. Inicialmente são avaliados os eventos que originam o dinamismo de servidores de dados. De acordo com os cenários possíveis, são estabelecidas estratégias para o sistema reagir a estes eventos fazendo com que os recursos sejam melhor utilizados.

Para contemplar o modelo proposto, o sistema de arquivos pode ser expandido para oferecer maior capacidade e desempenho. Para isso, é apresentado um modo de inserção de nós por parte do administrador, dando-lhe a liberdade de realocar recursos. É possível remover servidores de dados do sistema de arquivos e utilizar os recursos para outros fins, quando constatado que o sistema de arquivos está super-dimensionado. Ou então, adquirir novos equipamentos e associá-los ao sistema de arquivos para oferecer maior desempenho e capacidade de armazenamento.

Apenas a intervenção do administrador no sistema de armazenamento não garante um bom desempenho a todos os tipos de aplicações. Na maioria dos casos, o administrador tem um conhecimento restrito sobre o comportamento de aplicações executando no *cluster*. A possibilidade de ajustar o sistema de arquivos de acordo com o perfil de cada aplicação pode resultar em um ganho de desempenho. Assim, outro característica tratado pelo modelo é a inserção dinâmica de servidores de dados por parte do usuário do *cluster*. Para isso, parte dos recursos alocados por ele devem ser dedicados ao armazenamento de dados. Estes servidores de dados funcionam como recursos temporários e estão disponíveis apenas

durante a execução das aplicações de um determinado usuário. Desta forma, o sistema de arquivos pode ser configurado com um número mínimo de servidores permanentes de dados e, ao mesmo tempo, ser expandido de acordo com as necessidades de cada usuário.

Finalmente, o dinamismo de servidores de dados é associado à redundância de dados. Esta modelagem permite que o sistema de arquivos continue funcionando corretamente após a perda de servidores de dados. O uso de réplicas possibilita o acesso aos dados na existência de servidores com falha. Combinando estas três características tratadas pelo modelo é possível desenvolver um sistema de arquivos que, além de poder ser redimensionado para atender as demandas do administrador e dos usuários, pode oferecer maior disponibilidade dos dados.

Este capítulo está organizado da seguinte forma. Na primeira seção é apresentada a maneira utilizada para abstrair o dinamismo de servidores de dados, permitindo que estes sejam alterados de forma transparente ao restante do sistema. Em seguida, são apresentados os eventos que causam o dinamismo de nós em um sistema distribuído. A partir disso, foi modelado primeiro caso de dinamismo, que é a inserção de servidores de dados pelo administrador. Após, é tratada a inserção de nós temporários por parte do usuário. O último caso tratado é a perda de nós resultante de falhas. Para finalizar, é apresentado um resumo do capítulo apresentando os principais pontos de cada um desses casos.

3.1 Abstração do Dinamismo

A noção de grupos de armazenamento é utilizada para isolar o dinamismo dos servidores de dados (IODs) do restante do sistema de arquivos. Um grupo de armazenamento representa uma fração dos dados segundo o padrão RAID 0 (PATTERSON; GIBSON; KATZ, 1988). Este tipo de distribuição de dados foi escolhido pois atualmente o dNFSp utiliza este método de fracionamento para distribuir os dados entre os servidores. No contexto deste trabalho, os discos podem ser representados pelos nós do *cluster*. A Figura 3.1 mostra um exemplo de distribuição de dados através do algoritmo *round-robin* utilizando o padrão RAID 0. Neste esquema o conteúdo do arquivo é dividido em blocos de tamanhos iguais. Esses blocos são distribuídos de maneira circular entre os grupos disponíveis. A forma com que os dados são armazenados dentro de um grupo é invisível para o restante do sistema. Assim, ao invés de especificar um servidor ao qual os dados são escritos ou lidos, o servidor de metadados solicita esta operação a um grupo. Isso permite que o armazenamento dos dados possa ser modificado sem a necessidade de alterar o algoritmo empregado na distribuição dos dados.

O relacionamento entre servidores de dados e grupos é N para N. É possível ter um grupo com vários servidores, e ter um servidor presente em vários grupos. Os diferentes tipos de associação são ilustrados na Figura 3.2.

No primeiro caso, onde existe N **grupos por IOD**, cada IOD é responsável por armazenar dados de mais de uma porção do fracionamento do arquivo. Devido à abstração oferecida pela utilização de grupos, não é preciso que cada fração esteja armazenada em um servidor diferente. Por exemplo, um determinado servidor de dados pode ao mesmo tempo fazer parte de 2 grupos de armazenamento. Além disso, a abstração permite que o servidor responsável por uma determinada fração possa ser substituído por outro de forma transparente. Assim, a exis-

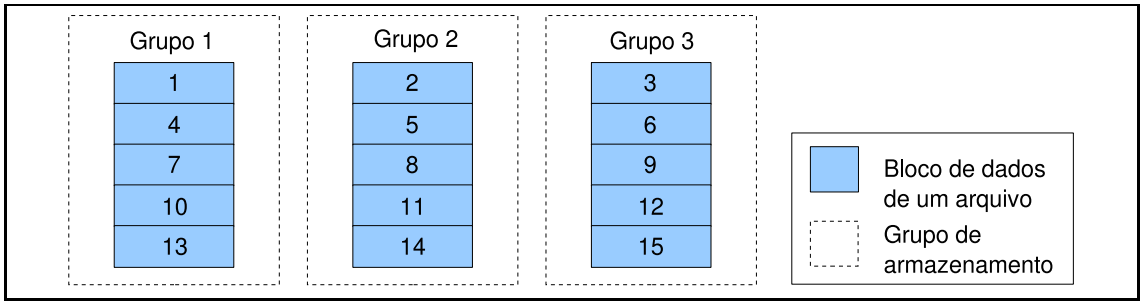

Figura 3.1: Disposição dos blocos de dados utilizando grupos de armazenamento em um sistema baseado no padrão RAID 0

tência de vários grupos por IOD possibilita a expansão do sistema, visto que mais IODs podem ser inseridos para assumir o gerenciamento de dados de algum dos grupos.

O caso onde existe **um IOD por grupo** representa um cenário igual ao obtido quando não é empregada a noção de grupos. Ou seja, cada fração de dados é armazenada por um servidor de dados diferente. Esta configuração não permite a expansão do sistema visto que todos os grupos já possuem um IOD exclusivo. O sistema é expansível somente quando o número de grupos é maior que o de IODs. Por outro lado, quando existe mais de um IOD por grupo, o sistema pode passar a conter IODs funcionando como réplicas.

Quando não é possível encontrar um grupo livre, o sistema de arquivos associa o IOD a um grupo que já contém nós. E neste caso, passa a existir N **IODs por grupo**. Os IODs de um mesmo grupo trabalham como réplicas. Em caso de falha de um o outro assume sua função. A quantidade de IODs do grupo com menor número de IODs determina quantas falhas simultâneas de servidores de dados são suportadas pelo sistema. Por exemplo, se um grupo possui 2 IODs e todos os demais possuem 3, o sistema garante o seu funcionamento se ocorrer apenas uma falha de servidores de dados. Em outras palavras, o grupo com 2 IODs pode perder somente um deles e seguir trabalhando normalmente. Portanto o número de falhas simultâneas suportadas pode ser determinado através da expressão: $\min_{i=1...n}\{IOD_i\} - 1$, onde $IOD_i, i = 1 \ldots N$ são os N IODs no Grupo.

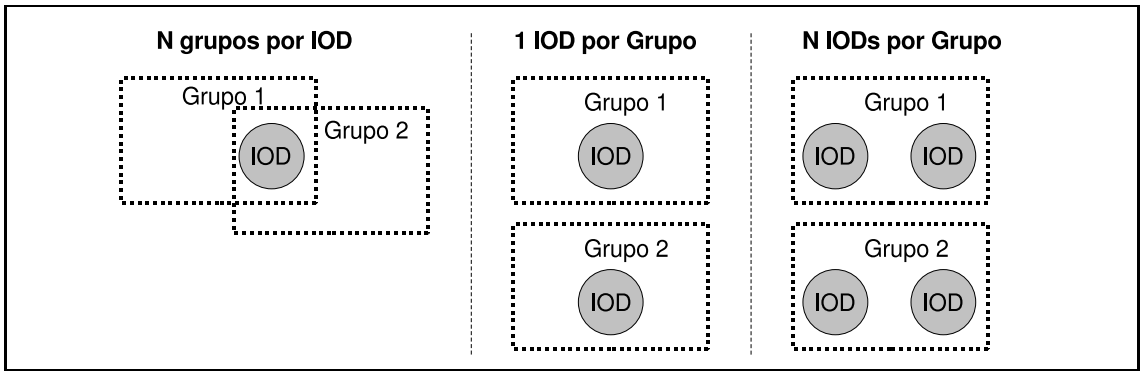

Figura 3.2: Tipos de associações possíveis entre grupos de armazenamento e servidores de dados (IODs)

O sistema é considerado em desequilíbrio quando constatado que a quantidade de grupos aos quais um nó faz parte, difere da quantidade de algum outro.

Em um sistema em desequilíbrio os IODs que fazem parte de um número maior de grupos são considerados **sobrecarregados**. Por exemplo, se um IOD esta associado a 2 grupos, e todos os demais estão associados a apenas um grupo, ele é considerado sobrecarregado. A partir da noção de grupos e de sobrecarga é que o gerenciador de servidores de dados toma as decisões de como associar um novo servidor de dados ao sistema.

3.2 Origem e Tratamento do Dinamismo

O dinamismo de nós em sistemas distribuídos pode ter origem em três eventos principais: **associação** (*join*), **desassociação** (*leave*) e **falha** (*fail*) (IAMNITCHI et al., 2002). No caso do dNFSp é possível subdividir a associação segundo a finalidade desejada. Por exemplo, uma nova associação pode ser feita pelo administrador para expandir a capacidade do sistema ou então substituir um nó falho. A mesma associação pode ser realizada por um usuário específico para oferecer, temporariamente, maior capacidade de armazenamento para sua aplicação.

A associação para expansão é o caso onde o administrador deseja agregar mais nós de armazenamento ao sistema de arquivos. Isso pode ser ocasionado por uma deficiência observada no desempenho e na capacidade de armazenamento oferecido aos usuários do *cluster*. O administrador pode solicitar a aquisição de novos equipamentos ou então realocar recursos materiais e, em seguida, notificar a existência deles ao sistema de arquivos.

Outro caso semelhante ao anterior, que exige um tratamento maior, é a associação pós-falha. Após ser detectado que um servidor de dados teve seu funcionamento comprometido, o administrador insere outro para suprir esta deficiência. Esta inserção tem por objetivo substituir um nó específico. Isso exige que os dados armazenados pelo nó que falhou sejam restaurados no novo servidor de dados. Em casos onde o nó inserido é o mesmo que falhou, é possível implantar mecanismos que atualizem apenas os dados que diferem entre o momento da falha e o momento do restabelecimento, para que o novo nó esteja disponível no menor espaço de tempo possível.

A desassociação de nós do sistema de arquivos ocorre quando se deseja explicitamente remover um nó que pertence ao sistema de arquivos. Isso permite que recursos subutilizados possam ser realocados para outros fins. Um exemplo disso ocorre quando a necessidade dos usuários é menor que a capacidade e o desempenho oferecidos pelo sistema de arquivos. Neste caso os dados que estão armazenados no servidor devem ser transferidos para outro servidor de dados. Com esta migração, os dados continuam disponíveis ao usuário após a remoção do servidor de dados.

A perda de nós por falha é um evento que ocorre em sistemas de arquivos distribuídos com probabilidade proporcional ao número de nós. Mecanismos para amenizar os efeitos da perda de nós devem ser levados em consideração quando se deseja oferecer um serviço confiável e de alta disponibilidade. Diferente do caso anterior, não existe uma desassociação explícita do nó. Isso impede que os dados sejam transferidos para outro servidor de dados antes da remoção. Para garantir o acesso futuro aos dados é preciso utilizar mecanismos que mantêm informações redundantes em diferentes servidores de dados. Estes dados redundantes podem ser utilizados para recuperar os dados de um servidor de dados

que falhou. Outra consequência da inexistência de desassociação explícita é a necessidade de um mecanismo de detecção de falhas. É ele o responsável por monitorar os nós e detectar quando algum não responde mais por suas funções.

3.2.1 Inserção de Servidores de Dados Iniciada pelo Administrador

Os principais objetivos da inserção de servidores de dados em um sistema de arquivos são a substituição de um nó falho e a expansão da capacidade do sistema. Em ambos os casos o nó inserido deve obter os dados já existentes no seu grupo de armazenamento para poder ser integrado ao sistema. Após a sua inserção, ele é capaz de tratar as requisições do usuário da mesma maneira que os servidores de dados já presentes no sistema.

Para inserir um novo servidor no sistema, o administrador deve comunicar este fato ao sistema de arquivos, como ilustra a Figura 3.3. Esta notificação permite ao sistema de arquivos escolher a melhor maneira de integrar o novo servidor de dados (IOD). Ao receber um novo nó o sistema de arquivos deve dar preferência ao grupo cujos nós estão mais sobrecarregados e associar o novo servidor a este grupo. Quando existir sobrecarga, esta escolha permite que ela seja aliviada.

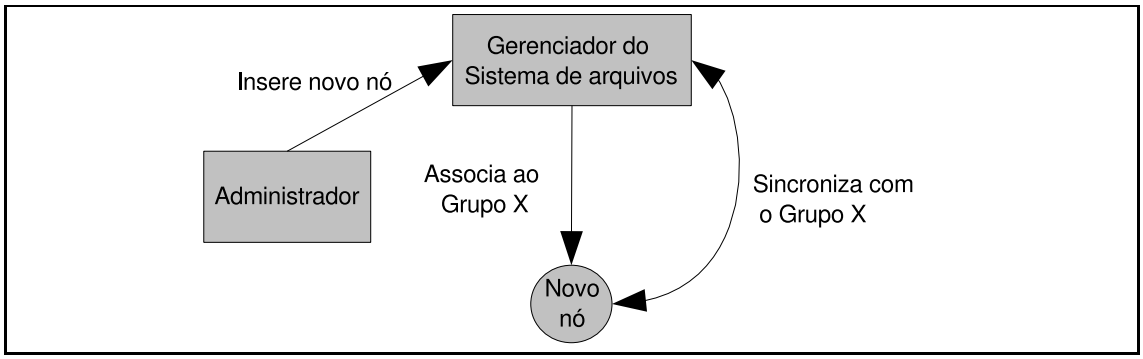

Figura 3.3: Inserção de novos servidores: o administrador anuncia o novo servidor; o gerenciador de servidores de dados notifica a qual grupo o nó deve se associar; o nó sincroniza com um dos membros do grupo

Após tomada a decisão sobre a qual grupo ele deve se associar, o novo servidor de dados deve ser notificado sobre este evento para que ele saiba quais são os dados que ele deve tratar. Além disso é preciso fornecer meios para que o novo servidor obtenha os dados já existentes no grupo. Após a obtenção dos dados ele passa a estar apto a tratar as requisições dos clientes e a ter as mesmas funções de um servidor já existente no sistema.

No tratamento do caso onde o servidor de dados está retornando de uma falha, é preciso dar preferência ao grupo que ele pertencia antes dela. Portanto, o sistema de arquivos deve ser informado de qual grupo o servidor de dados pertencia anteriormente. Esta medida pode diminuir o tempo de integração de um nó, visto que os dados que não foram alterados entre o momento da falha e o retorno não precisam ser transferidos.

3.2.2 Inserção de Servidores de Dados Iniciada pelo Usuário

Devido à grande variedade de aplicações que são executadas em um *cluster*, nem sempre é possível satisfazer às necessidades de armazenamento de todos os

usuários. O pouco conhecimento prévio que o administrador tem sobre as aplicações executadas torna o dimensionamento do sistema de arquivos uma tarefa bastante complexa. Uma má escolha na quantidade de servidores destinados a este fim pode resultar num desperdício de recursos quando o sistema de arquivos é super-dimensionado. Por outro lado, a disponibilização de poucos nós de armazenamento pode implicar em um baixo desempenho de algumas aplicações, principalmente as que efetuam um número significativo de operações de E/S.

O usuário, por outro lado, possui maior controle que o do administrador do *cluster* sobre o perfil de suas aplicações. Isso lhe permite determinar com maior precisão o padrão de acesso aos dados antes mesmo do início da execução. Com isso, um melhor uso dos recursos pode ser obtido se o usuário tiver a possibilidade de redimensionar o sistema de arquivos de acordo com as necessidades de suas aplicações.

Para possibilitar esta escolha, é preciso oferecer ao usuário ferramentas que permitam definir o número de nós que uma aplicação deseja dedicar ao armazenamento. Este número deve ser contabilizado junto com os nós de processamento no momento da reserva de recursos junto ao escalonador do *cluster*, (como PBS, OAR, etc.). Além disso, o redimensionamento do sistema de arquivos por parte de um usuário não pode afetar a configuração disponível aos outros usuários.

Como ilustrado na Figura 3.4, o usuário aloca um número de nós suficiente para executar sua aplicação e para o armazenamento. No momento em que estes recursos são disponibilizados, é preciso informar ao sistema de arquivos que uma aplicação deseja dedicar alguns de seus nós para o armazenamento. Fazendo uso das ferramentas de configuração, o usuário indica quais nós serão responsáveis pelo processamento e quais vão estar armazenando dados. Ao receber a solicitação do usuário, o sistema de arquivos faz as associações necessárias para que os novos nós sejam integrados ao sistema.

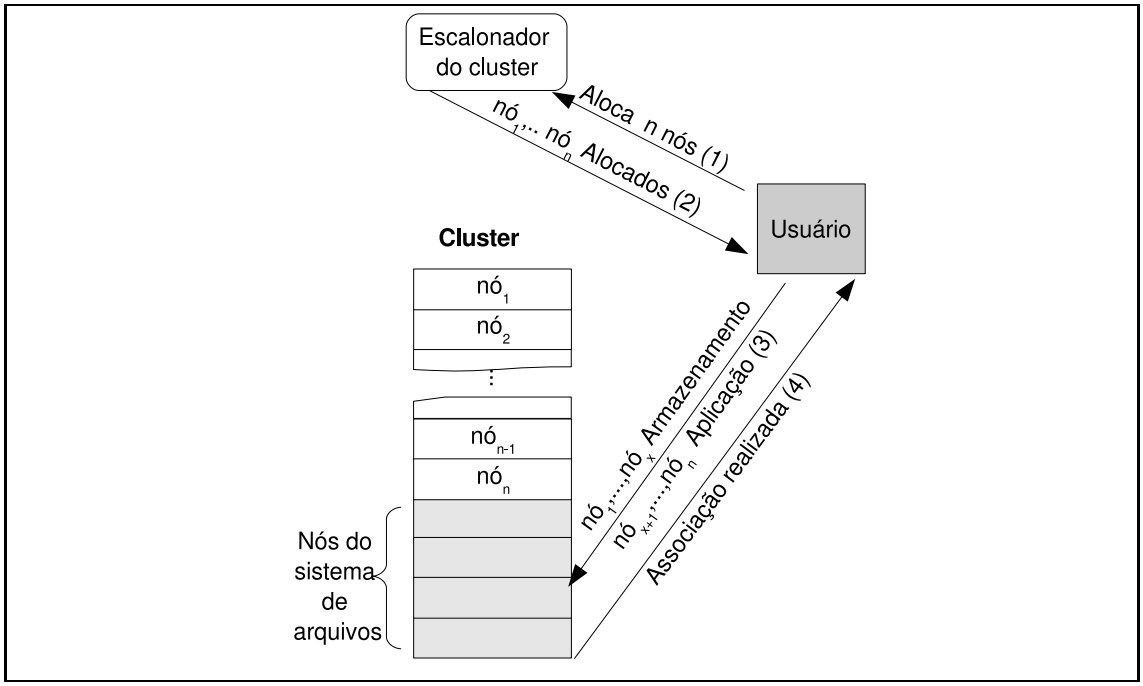

Figura 3.4: Etapas da alocação de nós de um *cluster* destinando x nós para servidores temporários de dados

Neste cenário, os nós de armazenamento inseridos pelo usuário são chamados de **servidores temporários de dados**. Estes nós armazenam informações pertencentes a uma aplicação durante a sua execução. Os demais servidores de dados, inseridos pelo administrador no momento da instalação do sistema de arquivos são chamados de **servidores permanentes de dados**. É neles que ficam armazenados os dados durante todo o tempo de vida do sistema de arquivos.

Para relacionar servidores temporários e servidores permanentes, também faz-se uso da noção de grupo de armazenamento. Cada servidor inserido pelo usuário é associado a um servidor permanente, e ambos passam a pertencer ao mesmo grupo. Porém não existe replicação de dados entre membros do grupo e o servidor temporário, e não existe suporte a falhas de servidores temporários. As únicas informações escritas nos servidores temporários são os dados criados pela aplicação associada a eles. Assim, dados gerados pela aplicação podem ser lidos e escritos diretamente desses servidores. Aplicações baseadas em iterações podem tirar proveito deste tipo de comportamento, uma vez que os dados produzidos em uma etapa são utilizados na etapa seguinte.

No momento da leitura, assim como no da escrita, os dados são solicitados aos servidores temporários. Como não existe replicação entre servidores permanentes e temporários, é possível que alguns dados requisitados pela aplicação não sejam encontrados no servidor temporário. Isso ocorre principalmente na leitura de dados criados em um momento anterior à execução da aplicação, ou então, dados criados por outras aplicações. Neste caso, a requisição de leitura é repassada ao servidor permanente que pertence ao mesmo grupo do servidor que recebeu a requisição. Toda a informação que não é encontrada nos servidores temporários e que não pertence a outras aplicações em execução é encontrada nos servidores permanentes.

O acesso aos servidores temporários de dados é restrito aos nós da aplicação associada a eles. Isso permite minimizar a interferência das operações de E/S de uma aplicação em outra. Portanto, aplicações que possuem nós temporários têm acesso exclusivo a seus servidores temporários de dados, enquanto que as demais apenas têm acesso aos servidores permanentes. Uma conseqüência deste isolamento é que os dados criados pelas aplicações que possuem servidores temporários de dados somente serão acessíveis às demais aplicações quando forem transferidos para os servidores permanentes de dados.

A Figura 3.5 possui a modelagem da taxa de transferência disponível na leitura de dados ao utilizar servidores temporários de dados. Considera-se que todos os clientes possuem o mesmo comportamento, ou seja, executam a mesma aplicação. As mensagens de controle, por possuírem um tamanho pequeno, não são levadas em consideração. A ilustração possui apenas um servidor de metadados. As mensagens que são enviadas e recebidas por ele são apenas mensagens de controle, o que não gera um gargalo significativo de dados. A vazão total de dados do canal de comunicação é representada por V_t e o número de clientes por n_c. Neste cenário o Cliente A possui um servidor temporário de dados, enquanto que os demais possuem acesso apenas aos servidores permanentes de dados. Com isso, é possível fazer uma aproximação da vazão de dados disponível para cada cliente, tanto para os que possuem servidores temporários de dados quanto para os demais.

A taxa de transferência de dados entre um servidor de dados permanente

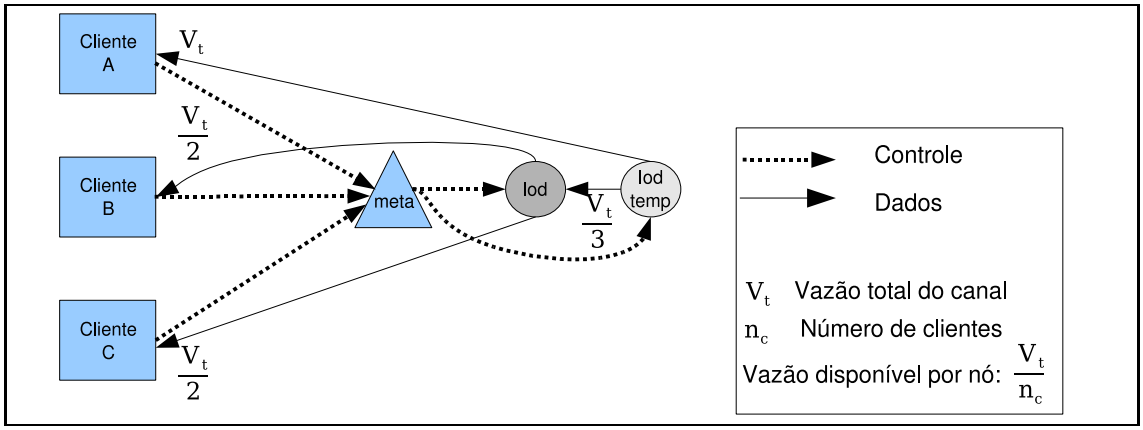

Figura 3.5: Modelagem da taxa de transferência disponível durante a leitura em cada conexão com e sem servidores temporários de dados

e um cliente depende do número de clientes no sistema. É possível fazer uma aproximação da vazão de dados de um cliente baseado na vazão total do canal. Isso pode ser obtido através da expressão: $\frac{V_t}{n_c}$. Isso ocorre porque todos os clientes concorrem pelo acesso aos servidores permanentes de dados. Já para clientes que fazem parte de uma aplicação que possui servidores temporários de dados o acesso é exclusivo. Conseqüentemente isso resulta em um maior desempenho de acesso aos dados, pois a vazão é compartilhada apenas pelos nós que pertencem a esta aplicação.

Como nem todos os dados estão armazenados nos servidores temporários, é possível que alguma informação necessite ser obtida dos servidores permanentes. Neste caso, mesmo que a aplicação possua servidores temporários, a vazão de dados é compartilhada com todos os nós do sistema. Dependendo da probabilidade (p) de ocorrer uma leitura de um servidor permanente, é possível estimar a vazão média de uma aplicação através da expressão: $p\frac{V_t}{n_c} + (1-p)V_t$.

Durante a escrita, os dados são transferidos entre cliente e servidor de metadados e, em seguida, entre o servidor de metadados e o servidor de dados. Portanto, existem dois fatores que podem limitar o desempenho deste tipo de operação. O primeiro deles é o número de servidores de metadados, o outro é o número de servidores de dados.

A existência ou não de servidores temporários de dados não afeta o desempenho quando o fator limitante for o número de servidores de metadados. Por outro lado, se o número de servidores de metadados é suficiente para não se tornar o gargalo da escrita, é possível fazer uma estimativa do ganho de desempenho que pode ser obtido utilizando servidores temporários de dados. A Figura 3.6 ilustra esta situação, onde vários clientes enviam dados para serem escritos nos servidores de dados. A situação é bastante semelhante à leitura, porém ao invés de depender do número de clientes, a vazão disponível depende da quantidade de servidores de metadados tentando escrever em um mesmo servidor de dados. Se considerarmos que as aplicações são idênticas, realizando as mesmas operações simultaneamente, podemos obter uma aproximação da vazão disponível através da expressão: $\frac{V_t}{n_m}$. Onde n_m é o número de servidores de metadados tentando escrever no servidor de dados.

Antes de liberar os recursos é preciso transferir os dados dos servidores tem-

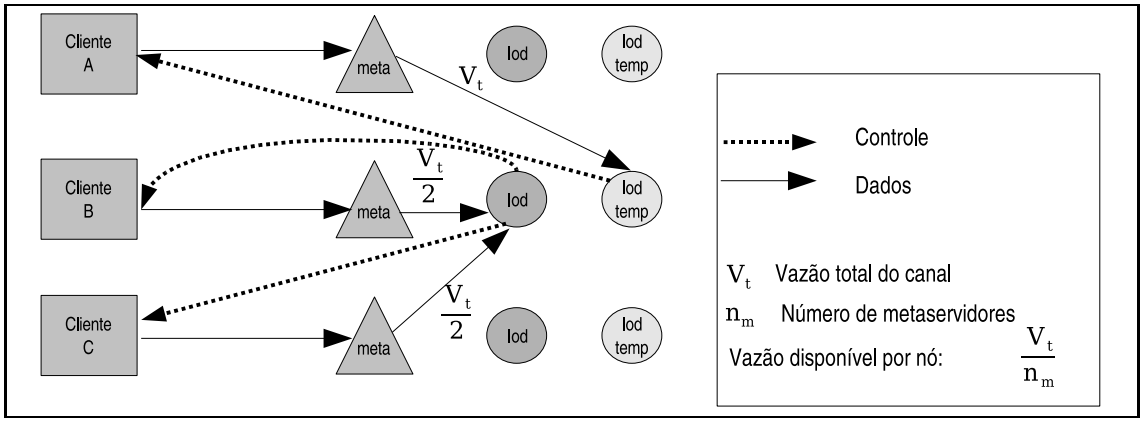

Figura 3.6: Modelagem da taxa de transferência disponível durante a escrita em cada conexão com e sem servidores temporários de dados

porários para os servidores permanentes. Este procedimento garante que os dados criados pela aplicação possam ser acessados no futuro. O gerenciador de servidores de dados deve informar ao IOD temporário quais são os IODs permanentes para os quais ele deve enviar os dados. Após o final da transferência os servidores temporários de dados são finalizados, e os recursos tornam-se disponíveis para a próxima alocação.

3.2.3 Perda de Servidores de Dados

A replicação de dados, tanto em sistemas de propósito geral quanto em sistemas distribuídos, é utilizada com dois objetivos principais: proporcionar maior confiabilidade ao sistema e aumentar o desempenho de acesso aos dados (POPEK et al., 1990). O aumento da confiabilidade pode ser obtido através da existência de cópias dos dados em mais de um local no sistema. Os dados continuam disponíveis enquanto pelo menos uma das réplicas estiver acessível, inclusive quando algum dos nós falhar. Do ponto de vista de desempenho, ter várias cópias de dados permite evitar sobrecargas na leitura, acessar a réplica mais próxima ou então, fazer acesso paralelo a diversos nós que contém réplicas dos mesmos dados. Para que estes objetivos possam ser atingidos, é necessário encontrar um método adequado para determinar a distribuição das réplicas e a sua localização no momento do acesso. Também é preciso garantir a existência de pelo menos uma réplica em funcionamento para oferecer confiabilidade aliada ao alto desempenho de acesso (BOLOSKY et al., 2000).

Neste trabalho não serão desenvolvidas novas técnicas de replicação. Isso porque algumas delas já foram estudadas e desenvolvidas na versão original do NFSp. As garantias de replicação e coerência de dados são de responsabilidade destes mecanismos. Portanto a perda de pacotes, incoerência de dados e a ordenação de mensagens não são tratadas no modelo de dinamismo de servidores de dados.

O modelo de falhas considerado neste trabalho, é o de falha silenciosa (*fail stop*) (SCHLICHTING; SCHNEIDER, 1983; JALOTE, 1994). Nesse modelo, um servidor de dados que entra em um estado de falha para de se comunicar com os demais membros do sistema. Isso significa que um servidor em estado inconsistente não envia mensagens incoerentes aos demais nós. Os membros do sistema

de arquivos que possuem comportamento bizantino (LAMPORT; SHOSTAK; PEASE, 1982) também não são considerados neste trabalho, devido à complexidade de detectá-los. Além disso, as falhas são apenas consideradas nos servidores de dados. Falhas no servidor de metadados ou nos clientes não são levadas em consideração, apesar de algumas técnicas utilizadas neste trabalho poderem ser aplicadas neste contexto.

Baseando-se no modelo de falhas abordado acima, é possível desenvolver um mecanismo de detecção de falhas que tem como função notificar o gerenciador do sistema de arquivos quando for encontrada uma falha em um servidor de dados. Após a detecção de um nó falho é preciso que o sistema de arquivo tenha a capacidade de reagir à perda deste servidor de dados (ver Figura 3.7). A primeira medida a ser tomada é o isolamento do servidor de dados. Para isso, todas as solicitações que seriam enviadas àquele nó devem ser tratadas a partir das informações redundantes destes dados. Por exemplo, os outros nós que fazem parte do mesmo grupo de replicação podem receber e processar as requisições que seriam destinadas ao nó falho.

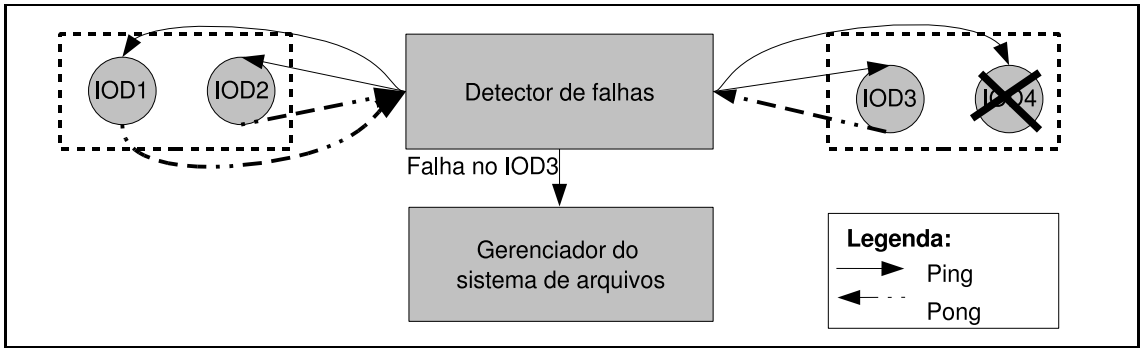

Figura 3.7: Mecanismo de detecção de falha de servidores de dados

No entanto, apenas o isolamento do servidor de dados que falhou não é suficiente para garantir o funcionamento do sistema de arquivos num longo espaço de tempo. Para minimizar a probabilidade de interrupção do funcionamento do sistema de arquivos é preciso garantir a existência de um número mínimo de réplicas de dados. Assim, ao detectar um nó falho, é preciso eleger um outro elemento para substituí-lo e dar continuidade à replicação de dados. Isto pode ser feito através da utilização de um nó reserva (*spare*), ou então sobrecarregar um outro servidor de dados que não pertence ao mesmo grupo de replicação.

3.3 Ciclo de Vida do Sistema de Arquivos

Nesta seção é discutida a progressão do sistema de arquivos através do tempo, utilizando o tratamento do dinamismo de servidores de dados. Para isso, é tomado o exemplo de um *cluster* cujo administrador decide instalar um sistema de arquivos distribuído para oferecer aos seus usuários algo mais eficiente que um servidor NFS centralizado. Neste exemplo são utilizadas quantidades de servidores de dados ilustrativas, que podem não representar um caso real.

Neste exemplo, a primeira atitude do administrador é instalar um sistema de arquivos com replicação, para que os dados dos usuários sejam armazenados com maior segurança. Para isso, ele utiliza 2 nós, um como réplica do outro.

Isso permite que um dos servidores falhe, sem comprometer o acesso aos dados. Parte dos usuários contenta-se com esta configuração e utiliza o sistema de arquivos sem nenhum ajuste especial. Outros, por terem aplicações que exigem maior desempenho do armazenamento, passam a utilizar os servidores temporários de dados. Uns fazem isso para diminuir a interferência de outras aplicações no desempenho da sua e terem uma maior garantia do sistema de arquivos, enquanto outros desejam expandir o sistema temporariamente.

Ao perceber que grande parte dos usuários está utilizando servidores temporários de dados, o administrador decide expandir a capacidade do sistema, duplicando o número de servidores permanentes de dados. Com isso, alguns dos usuários que antes utilizavam servidores temporários têm seus requisitos satisfeitos pela configuração permanente do sistema. Assim, o tempo de transferência de dados dos servidores temporários para os permanentes, que era gasto por estes usuários, pode ser utilizado para outras alocações. Os usuários que utilizavam servidores temporários para obter maior isolamento de suas aplicações vão continuar utilizando este mecanismo. Além deles, alguns continuarão achando insuficiente o número de servidores permanentes instalados pelo administrador e vão continuar utilizando servidores temporários.

Com o passar do tempo, é possível que alguns servidores de dados falhem. Estas falhas resultam em reconfiguração do sistema e conseqüentemente em perda de desempenho. Se os nós não forem repostos, a perda sucessiva pode reduzir sucessivamente a quantidade de servidores de dados permanentes. Esta redução faz com que o desempenho piore, e os usuários que antes utilizavam apenas os nós permanentes poderão ter a necessidade de utilizar servidores temporários para obter resultados satisfatórios quanto ao armazenamento.

Com o emprego deste mecanismo de servidores temporários foi possível dar maior autonomia aos usuários. O desempenho do sistema de arquivos deixa de depender apenas do administrador. Porém, o administrador deve observar o comportamento dos usuários para identificar suas necessidades e adaptar o sistema para que os recursos sejam utilizados da melhor forma. Também é preciso realizar a reposição de nós perdidos para evitar a degradação do desempenho do sistema.

3.4 Resumo do Capítulo

O dinamismo de nós em sistemas distribuídos tem origem em três principais eventos. O primeiro deles é a associação, onde um novo nó é inserido no sistema. O segundo é a desassociação, que acontece quando um nó é explicitamente retirado do sistema. O terceiro evento é a falha de um nó, onde ele deixa de fazer parte do sistema por ter seu funcionamento comprometido.

Partindo das diferentes origens do dinamismo este trabalho propõe um modelo de tratamento do dinamismo de servidores de dados voltado para sistemas de arquivos distribuídos. Nele não considera-se a inserção e remoção de servidores de metadados. Dentre os eventos causadores de dinamismo, o evento de associação foi subdividido em duas classes distintas, dependendo de quem teve a iniciativa de inserir: o administrador do sistema ou algum usuário.

Um esquema de grupos de dados é utilizado para abstrair a associação e desassociação de nós. Estes grupos representam unidades de armazenamento, onde

os dados podem ser escritos ou lidos. Um grupo pode conter vários servidores de dados, e um servidor de dados pode participar de mais de um grupo. Esta diversidade de situações permite que o sistema se reconfigure de acordo com os nós disponíveis. Quando mais de dois servidores pertencem ao mesmo grupo de armazenamento, um funciona como réplica do outro.

Entre os casos de dinamismo de servidores de dados tratados no modelo apresentado neste capítulo temos a inserção de nós por parte do administrador. Esta alternativa é utilizada normalmente para expandir a capacidade do sistema. Assim, mais recursos podem ser associados ao armazenamento para oferecer maior desempenho ou maior capacidade de armazenamento. Um servidor de dados, ao ser inserido, é associado a um grupo de armazenamento. Porém, antes de ser utilizado efetivamente, ele deve obter os dados do grupo ao qual ele vai se associar, para que em seguida ele possa tratar as requisições dos clientes.

O segundo caso tratado no modelo é a inserção de nós iniciada pelo usuário. Ao tratar este caso é possível oferecer um serviço de maior qualidade aos usuários, visto que nem sempre o administrador tem conhecimento sobre as necessidades de armazenamento de cada usuário que utiliza o *cluster*. A possibilidade da inserção de servidores de dados por parte do usuário permite que ele dedique ao armazenamento uma parte de seus nós alocados. Estes nós se mantêm associados ao sistema de arquivos apenas durante a execução da aplicação do usuário. Por serem acessíveis unicamente por um certo período, eles são chamados de servidores temporários de dados. Além disso, os nós temporários são exclusivos ao usuário. Assim, aplicações de um usuário não têm acesso aos servidores temporários de outro usuário. Isso aumenta o isolamento entre as aplicações, o que em alguns casos pode oferecer maior desempenho.

Finalmente, ao combinar o dinamismo de servidores de dados com a replicação de dados, é possível manter o sistema de arquivos acessível mesmo após a falha de um dos servidores de armazenamento. Para isso, é necessário que os dados que pertenciam ao servidor que falhou possam ser recuperados a partir das informações encontradas em outros servidores. Isso é feito através da associação de vários servidores de dados a um mesmo grupo de armazenamento. Esta configuração faz com que os servidores de dados de um mesmo grupo funcionem como espelho um do outro. Detectada a perda de um servidor de dados, o sistema de arquivos pode se reconfigurar com os nós que restaram. Depois do período de reconfiguração, ele está apto a suportar a falha de mais um servidor de dados.

No capítulo seguinte, será apresentada a implementação de um protótipo de sistema de arquivos que funciona segundo o modelo desenvolvido neste capítulo. Cada um dos cenários é implementado tendo como base o sistema de arquivos dNFSp. Algumas ferramentas e mecanismos já existentes neste sistema de arquivos foram adaptados, e outras novas funcionalidades foram desenvolvidas. Assim, foi possível ter uma versão funcional que segue o modelo proposto.

4 IMPLEMENTAÇÃO DO PROTÓTIPO

No capítulo anterior, foi apresentado um modelo de dinamismo de servidores de dados para sistemas de arquivos distribuídos. Este modelo considera três caraterísticas principais: a inserção de servidores efetuada pelo administrador, a inserção por iniciativa do usuário e a perda de servidores de dados por falha. As etapas da implementação das diferentes ações tomadas são apresentadas neste capítulo. Inicialmente são analisadas as adaptações necessárias no dNFSp para oferecer maior sincronismo entre os servidores de metadados e assim permitir a implementação do dinamismo de servidores de dados. Nesta adaptação, o protocolo utilizado pelo dNFSp foi reformulado. Além disso, houve uma remodularização do código do sistema de arquivos para facilitar o desenvolvimento de novas funcionalidades.

Para evitar ambigüidade, a nova implementação do dNFSp será chamada de dNFSp versão 2, ou dNFSp2. As referências à versão original, resultante da tese de doutorado intitulada "Uma Proposta de Distribuição do Servidor de Arquivos em Clusters" (ÁVILA, 2005), serão feitas através de dNFSp versão 1, ou dNFSp1.

O restante deste capitulo será organizado em 7 seções. Na primeira parte, que compreende a Seção 4.1, é apresentada a reformulação do algoritmo de sincronização de metadados. Em seguida, na Seção 4.2, são apresentadas as adaptações necessárias para a implementação do dNFSp2 feitas no mecanismo de servidores de dados virtuais e no sistema de replicação. Na Seção 4.3, é apresentada a implementação da inserção de servidores de dados iniciada pelo administrador. Já na Seção 4.4 é ilustrada a interface entre o usuário e o sistema de arquivos, que possibilita a inserção de servidores temporários de dados. Após, na Seção 4.5, é explicado o mecanismo de detecção de falhas e de reconfiguração. As alterações na arquitetura e estrutura do sistemas de arquivos também são ilustradas neste capítulo, na Seção 4.6. Por fim é feito um resumo da fase de implementação do dinamismo de servidores de dados no dNFSp.

4.1 Adaptação do Protocolo de Sincronização do dNFSp1

O uso de uma adaptação do mecanismo de consistência relaxada no dNFSp1 pode ocasionar inconsistência em metadados armazenados em múltiplos servidores. Isso ocorre quando dois metasservidores acessam o mesmo arquivo. A chance de ocorrer inconsistência aumenta se a informação a ser escrita depender de algum atributo contido nos metadados. Por exemplo, uma incoerência no tamanho do arquivo pode resultar na perda de dados se a posição onde os dados devem ser escritos for relativa ao final do arquivo. Neste caso, um cliente pode

sobrescrever dados de outro, enquanto que o comportamento desejado seria adicionar conteúdo ao final do arquivo. Este tipo de comportamento faz com que algumas aplicações possam não executar corretamente no dNFSp1, por exigirem maior coerência de metadados.

A inserção de novas funcionalidades no sistema de arquivos também pode exigir maior sincronismo entre os servidores de metadados. Na implementação do dinamismo de servidores de dados, é preciso que os servidores de metadados se comuniquem com maior freqüência. Alguns exemplos de novas funcionalidades que têm sua implementação facilitada com o maior sincronismo entre servidores de metadados são: controle de versão de arquivos e de blocos, para facilitar a recuperação de falhas; distribuição do conteúdo dos arquivos entre os servidores de dados de forma heterogênea; restrição do acesso aos nós de armazenamento incluídos pelo usuário. Por isso, a primeira etapa deste trabalho foi a adaptação do dNFSp1 para que fosse possível desenvolver a inserção dinâmica de servidores de dados. Com isso, fez-se necessária a reformulação do algoritmo de sincronização de metadados.

4.1.1 Sincronização de Metadados Baseada em *Hash*

Dentre as principais abordagens empregadas na distribuição de dados e metadados em sistemas distribuídos, é possível destacar duas grandes classificações: particionamento de sub-árvore de diretórios e distribuição em *hash*. No primeiro caso, os metadados pertencentes a uma mesma sub-árvore encontram-se agrupados em um mesmo servidor. Para acessar um determinado arquivo, o cliente deverá acessar os metasservidores responsáveis por cada sub-árvore, de acordo com o caminho do arquivo. Os servidores de metadados são pesquisados hierarquicamente, até encontrar a informação desejada. Sistemas de arquivos como AFS (Andrew File System) (HOWARD et al., 1988; MORRIS et al., 1986), Coda (BRAAM, 1998), NFS (Network File System) (PAWLOWSKI et al., 1994; SANDBERG et al., 1985) e LFS (Log-Structured File System) (ROSENBLUM; OUSTERHOUT, 1992) são implementados por servidores que podem ser responsáveis por uma ou mais sub-árvores do sistema.

Em trabalhos como o dos autores Levy e Silberschatz (LEVY; SILBERSCHATZ, 1990), é possível encontrar mais detalhes sobre o funcionamento e meios de implementar o mecanismo de sub-árvore. Nestas implementações, a principal desvantagem é o balanceamento de carga. Quando uma sub-árvore é acessada com maior freqüência que as demais, surge um desequilíbrio entre os servidores de metadados. Isso resulta em sobrecarga do servidor responsável pela sub-árvore mais concorrida.

A segunda maneira comumente utilizada para gerenciar metadados foi introduzida pelo sistema de arquivo Vesta (CORBETT; FEITELSON, 1996; CORBETT; BAYLOR; FEITELSON, 1993). Esse sistema aplica uma função de *hash* sobre o caminho dos arquivos para determinar a localização dos metadados. A utilização de uma função de *hash* apropriada diminui as chances de sobrecarga de um servidor do sistema. Com esta técnica é possível contornar o desbalanceamento de carga, que é o principal problema da implementação utilizando sub-árvore. No Vesta, os clientes calculam o *hash* baseado em todo o caminho do arquivo. A partir do resultado deste cálculo o cliente determina qual nó é preciso contatar para obter os metadados do arquivo desejado.

O sistema de arquivos PVFS2 (CARNS et al., 2000; LATHAM et al., 2004) faz uso de técnica semelhante para dar suporte a múltiplos metasservidores e distribuir a carga entre eles. Isto é feito distribuindo faixas de valores que podem ser resultados da função de *hash* entre os metasservidores. Esta configuração é feita na inicialização do sistema e é chamada de *range*.

O mecanismo de gerenciamento de metadados LH (*Lazy Hybrid*) (BRANDT et al., 2003) faz uma combinação entre os dois métodos, ele distribui os diretórios através do método hierárquico, e os arquivos utilizando *hash*. Uma implementação utilizando servidores NFS baseada no algoritmo do sistema de arquivos Vesta também foi implementada pelo sistema de arquivos Expand (GARCIA-CARBALLEIRA et al., 2003). No entanto, o Expand tem como princípio fazer modificações apenas no cliente, deixando intacto o servidor. O que se opõe à proposta do NFSp de fazer apenas modificações no servidor.

Para oferecer um maior sincronismo entre os metasservidores do dNFSp, foi desenvolvida uma variante da técnica de *hash*, combinada com algumas características da divisão em sub-árvore. Esta abordagem híbrida é necessária visto que a árvore de diretórios está em todos os metasservidores e apenas os arquivos são encontrados através do *hash*. Diferente da maioria dos sistemas de arquivos, a utilização do cálculo do *hash* no dNFSp2 é realizada no servidor de metadados, e não no cliente. Isso mantém sua conformidade com a proposta do NFSp de fazer modificações apenas no lado do servidor, sem alterar a semântica do protocolo NFS.

O dNFSp2 garante que os metadados vistos por um metasservidor sejam os mais recentes no sistema. Isso é feito através da existência de apenas uma versão válida dos metadados de um arquivo no sistema. Cada arquivo possui um meta servidor principal, chamado de *Hash*, que mantém a versão mais recente dos metadados. Para um metasservidor acessar os metadados de um arquivo, é preciso antes contatar o *Hash* para obter os metadados mais recentes. O uso da função de *hash* para localizar os metadados contribui para distribuição de carga entre os metasservidores, visto que o resultado da função de *hash* não será o mesmo para todos os arquivos. Com isso consegue-se uma implementação que esta completamente dentro do modelo LRC.

Os passos envolvidos na criação e acesso ao arquivo de metadados (metarquivo) são ilustrados na Figura 4.1. Para a criação do metarquivo (Figura 4.1a), o metasservidor calcula a função de *hash* através do nome do arquivo e obtém o identificador do metasservidor *Hash*. Com este identificador ele obtém o endereço do *Hash* e o envia uma mensagem notificando a criação do metarquivo. O *Hash*, ao receber a mensagem, armazena localmente os metadados. Quando um cliente necessita acessar um arquivo (Figura 4.1b), o metasservidor deve calcular o *hash* e verificar com o metasservidor *Hash* a existência do metaarquivo. Se ele existe, o *Hash* responde com a versão mais recente dos metadados. Caso contrário, a resposta será uma mensagem de erro.

Neste algoritmo, o número de mensagens trocadas entre os metasservidores é constante: uma solicitação e uma resposta. Ao analisarmos este número de mensagens trocadas, podemos considerar a complexidade do dNFSp2 como $O(1)$, pois, independentemente da quantidade de servidores de metadados, o número de mensagens trocadas é constante. Já o dNFSp1 possui uma complexidade $O(n)$, pois é preciso contatar em média $\frac{n}{2}$ servidores de metadados até encontrar

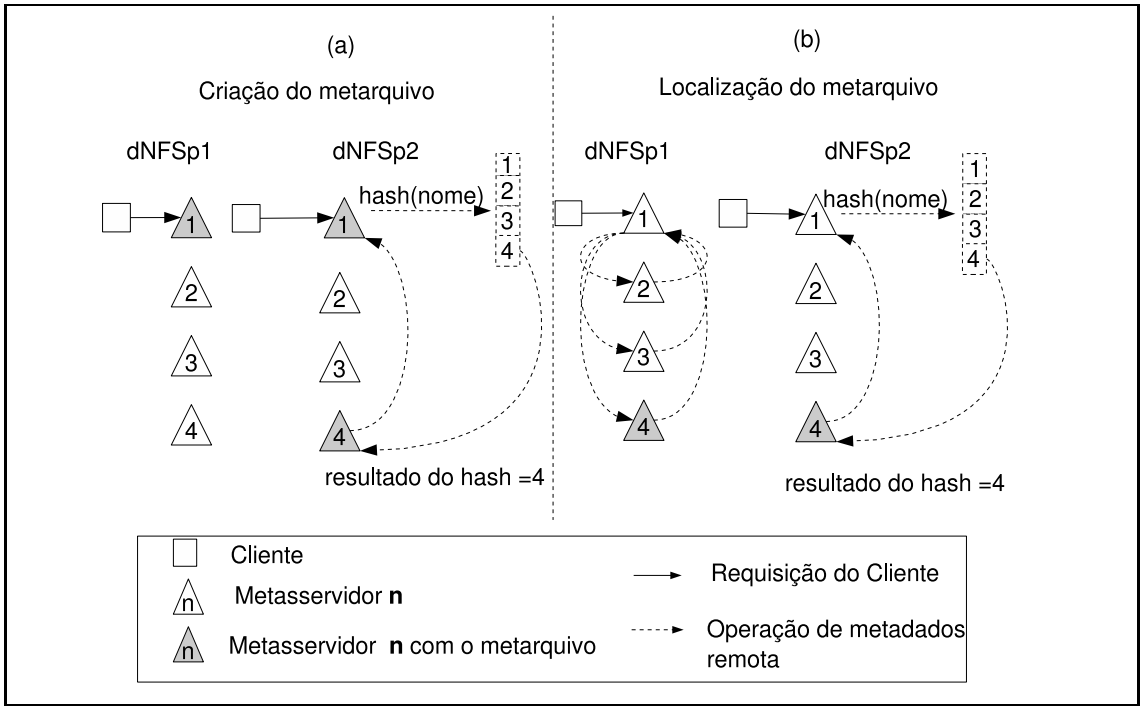

Figura 4.1: Comparação entre os métodos de criação e acesso aos metadados do dNFSp1 e dNFSp2

a informação desejada.

Outra mudança no dNFSp2 é o protocolo usado para transferir dados entre dois metasservidores. Nessa nova implementação, o sistema de transferência rcp foi substituído pelo protocolo de chamada de procedimentos remotos (*Remote Procedure Call*, RPC). A comunicação utilizando rcp tem um custo elevado, pois ele exige a execução de um comando externo a cada transferência de dados. A interferência do baixo desempenho do rcp no novo algoritmo seria mais visível, pois, para obter maior sincronismo, os servidores de metadados devem comunicar-se com maior freqüência. A utilização de RPC também coloca o mecanismo de sincronização de metadados no mesmo nível de abstração do NFS, pois tanto cliente e servidor do NFS quanto os metasservidores do dNFSp2 utilizam o mesmo protocolo de comunicação. Com isso, a manutenção do código e a depuração são facilitadas.

4.1.2 *Cache* de Metadados Baseado no Último Acesso

O uso de *cache* de dados e metadados é uma técnica freqüentemente utilizada em sistemas de arquivos distribuídos para aumentar o desempenho do acesso à informação. Isso é obtido através do armazenamento de uma cópia dos dados no momento do primeiro acesso. Nos acessos subseqüentes esta cópia é utilizada, evitando a troca de mensagens desnecessárias entre os nós.

O emprego de *cache* em sistemas de arquivos distribuídos pode ser feito em diferentes elementos do sistema: clientes, servidores de metadados ou servidores de dados. Além da localização, o tipo de informação armazenada também pode variar, por exemplo: *cache* de dados ou *cache* de metadados. Estas variações podem ser combinadas de diferentes maneiras, permitindo tirar proveito de *cache* de informações em diversos pontos do sistema.

O mais freqüente é encontrar *cache* do conteúdo dos arquivos feita pelos clientes. Esta técnica é implementada pela grande maioria dos sistemas de arquivos distribuídos. Neles são empregados mecanismos para manter a coerência entre as *caches* de diferentes nós. A técnica utilizada para manter a coerência dos dados em *cache* varia de um sistema de arquivos para outro. O NFS por exemplo, possui um mecanismo de consistência de *cache* baseado em *timeout*. Não há invalidação explicita da *cache* dos clientes, devido à natureza sem estado do protocolo.

Alguns sistemas mantêm também *cache* de metadados. Um exemplo é o Archipelago (JI et al., 2000) que faz *cache* dos metadados do conteúdo dos diretórios. Os atributos são armazenados de acordo com a necessidade de utilização. Nem todas as informações dentro do metarquivo são replicadas no Archipelago, apenas as informações essenciais para os demais servidores de metadados poderem encontrar os dados. Isso reduz o número de vezes que é preciso contatar os servidores de metadados para manter a coerência entre eles.

No sistema de gerenciamento de metadados *Lazy Hybrid* (LH) também existe um sistema de *cache*. Quando algum diretório hierarquicamente superior ao que se deseja acessar encontra-se em um outro servidor de metadados, é preciso contatar este servidor para obter a informação desejada. Neste caso, o LH armazena os metadados em *cache*, evitando comunicações a mais em acessos futuros. Em caso de modificação dos metadados, uma mensagem de invalidação é enviada a todos os metasservidores para que estes desconsiderem as informações inválidas em sua *cache*.

A solução adotada pelo dNFSp2 faz o controle de validade da *cache* através de um esquema de *token*. Ele é semelhante à **posse na escrita** (*write-ownership*) utilizada pelo xFS. Para implementar este mecanismo, foi introduzido um novo atributo aos metadados. Ele serve para indicar qual foi o último servidor de metadados a acessar o metarquivo. O novo atributo faz o papel do *token*, que pode estar de posse de qualquer um dos metasservidores. Um metasservidor não precisa contatar os outros servidores antes de acessar o metarquivo se ele foi o último a acessá-lo. Isso porque a sua cópia está atualizada. O metasservidor *Hash*, funciona como um ponteiro para o metasservidor que possui o *token* de último acesso ao metarquivo. A transmissão do *token* de um servidor de metadados para outro sempre feita por intermédio do *Hash*.

O algoritmo usado para gerenciamento de *cache* de metadados baseado em *token* é representado pelo fluxograma da Figura 4.2. O caso mais simples ocorre quando os metadados solicitados são encontrados localmente pelo metasservidor e este possui o *token* dos metadados (1). Neste caso, não é preciso contatar outros servidores de metadados antes de responder ao cliente. Se por outro lado, os metadados não existem, ou o metasservidor não possui o *token*, é preciso contatar o *Hash* (Meta B) antes de prosseguir o tratamento da requisição (2). O *Hash* pode responder de duas formas. A primeira é quando o próprio *Hash* é o dono do *token*, neste caso ele responde com os metadados do arquivo (3). Caso contrário (4), ele responde com o endereço do servidor de metadados que é o dono atual do *token* (Meta C). Ao receber este endereço, o servidor de metadados que esta tratando a requisição do cliente deve solicitar os metadados ao dono do *token* (5). Após receber os metadados, ele pode continuar tratando a solicitação do cliente. A partir deste momento, ele passa ser o novo dono do *token*.

Em um mesmo instante apenas um metasservidor pode possuir o *token* de um

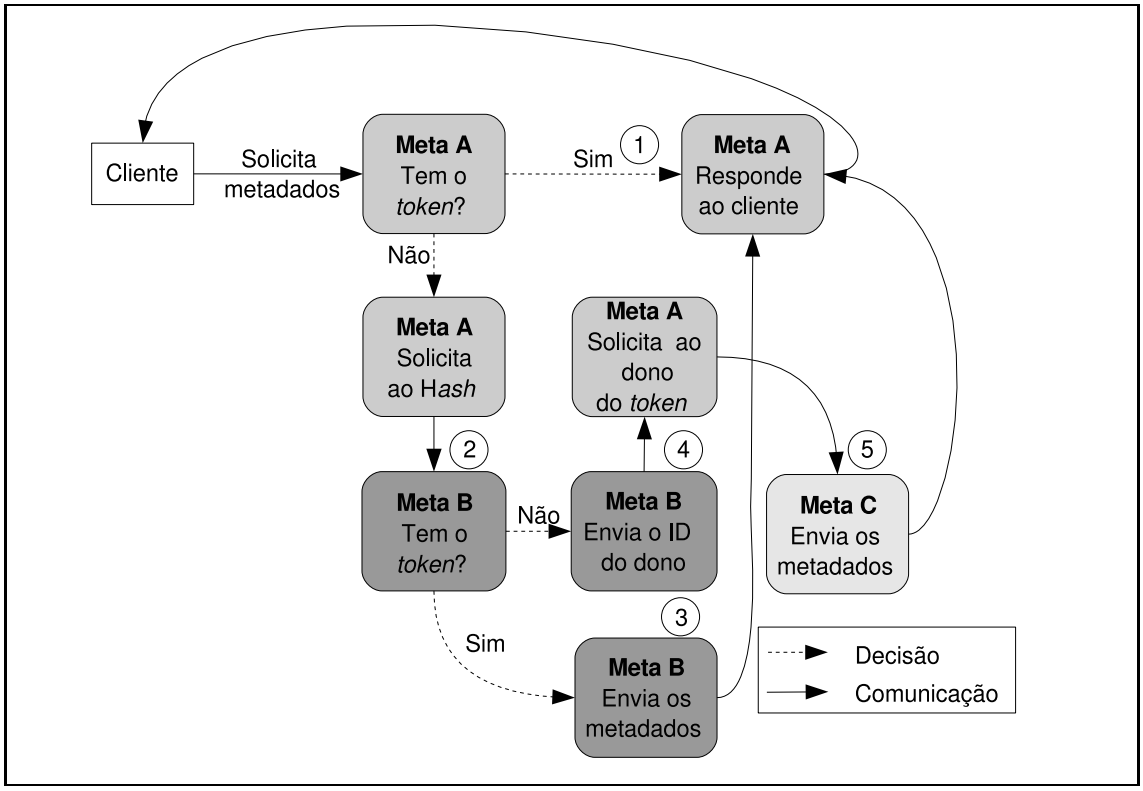

Figura 4.2: Fluxograma do algoritmo de gerenciamento de *cache* de metadados no protocolo de sincronização do dNFSp2

metarquivo. Não existe maneira de dois ou mais metasservidores serem considerados como o último a ter acessado um arquivo. O fato de não possuir o *token* é um indicativo de que a cópia local dos metadados pode estar divergente da versão mais recente do sistema.

4.2 Mecanismos Auxiliares

Vários mecanismos e conceitos existentes no dNFSp1 foram mantidos durante a implementação deste trabalho. Porém, alguns tiveram que ser adaptados para se adequarem aos requisitos do modelo proposto. Estes é o caso do **Gerenciador Distribuído de Servidores de Dados**, do sistema de **Servidores de Dados Virtuais** e do mecanismo de **Replicação de Dados**. O funcionamento de cada um deles e as modificações feitas para sua adaptação ao modelo de dinamismo serão apresentadas a seguir.

4.2.1 Gerenciamento Distribuído de Servidores de Dados

O NFSp possui um gerenciador de servidores de dados que tem como objetivo facilitar a manutenção destes servidores. Ele é implementado através de uma ferramenta chamada *iod_mgmt*. Esta ferramenta funciona como uma interface entre o administrador e o sistema, evitando a necessidade de definir arquivos de configuração. Além disso, a existência deste mecanismo permite alterar o sistema durante a sua execução sem ter que reinicializar todos os serviços para que as modificações sejam efetivadas.

Alguns exemplos de operações do gerenciador de servidores de dados que

permitem a alteração da configuração do sistema em tempo de execução são: a adição e a remoção de servidores de dados, a verificação do estado atual do sistema e a definição de parâmetros do fracionamento de arquivos e da detecção de falhas. Além destas operações, existe a opção de carregar um arquivo descrevendo as modificações desejadas. Assim, é possível adicionar ou remover vários servidores de dados de uma só vez.

O gerenciador de servidores de dados é um serviço que executa no mesmo nó do servidor de metadados. Ele utiliza mecanismos de comunicação inter-processos para receber os comandos de configuração enviados por um lançador de comandos. De acordo com as solicitações ele transmite as alterações necessárias ao servidor de metadados.

O tipo de comunicação entre *iod_mgmt* e servidor de metadados depende de qual implementação do NFSp é utilizada. Uma das implementações existentes executa como um servidor no espaço do usuário, e outra versão é implementada como um módulo do núcleo do sistema operacional. No primeiro caso, a troca de mensagens entre o *iod_mgmt* e o servidor de metadados é feita através de canais de comunicação inter-processos. Este tipo de comunicação é o mesmo utilizado entre a interface do usuário e o *iod_mgmt*. Já na versão implementada no núcleo do sistema operacional, a comunicação é feita através da uma interface de acesso ao sistema operacional, no caso, a interface "proc". Reunindo estes dois tipos de acesso em um só servidor, o *iod_mgmt* passa a funcionar como uma interface única de configuração do sistema de arquivos. Porém, o mecanismo de comunicação utilizado na implementação exige que o servidor de metadados, o *iod_mgmt* e o lançador de comandos sejam executados no mesmo nó.

Com o desenvolvimento do dNFSp1, o sistema passa a suportar vários servidores de metadados. Porém, o dNFSp1 utiliza uma configuração estática de servidores de dados, o que faz com que ele não necessite de um gerenciador de servidores de dados. Por isso, foi necessário desenvolver uma versão distribuída do *iod_mgmt*. Nesta nova versão, todos servidores de metadados são notificados sobre as alterações na configuração, feitas em tempo de execução.

A Figura 4.3 mostra como é feito o tratamento das requisições do usuário na versão distribuída do gerenciador de servidores de dados. Em cada metasservidor é executada uma instância do *iog_mgmt*. O usuário solicita as operações de configuração através do lançador de comandos. Este lançador as transmite para todos os *iod_mgmt* do sistema, utilizando uma lista de gerenciadores de servidores de dados. Cada *iod_mgmt* processa as solicitações e notifica as alterações ao seu servidor de metadados.

A comunicação entre o lançador de comandos e o servidor de metadados deve ser feita via rede para que as informações possam chegar a todos os *iod_mgmt*. Com isso, o mecanismo de comunicação inter-processos deixa de ser uma solução adequada, pois ele suporta apenas a transferência de mensagens entre processos de um mesmo nó. Na comunicação entre o lançador de comandos e as instâncias do *iod_mgmt*, o canal de comunicação inter-processos foi substituído por um canal TCP. A comunicação entre os *iod_mgmt* também é feita utilizando este tipo de canal de comunicação.

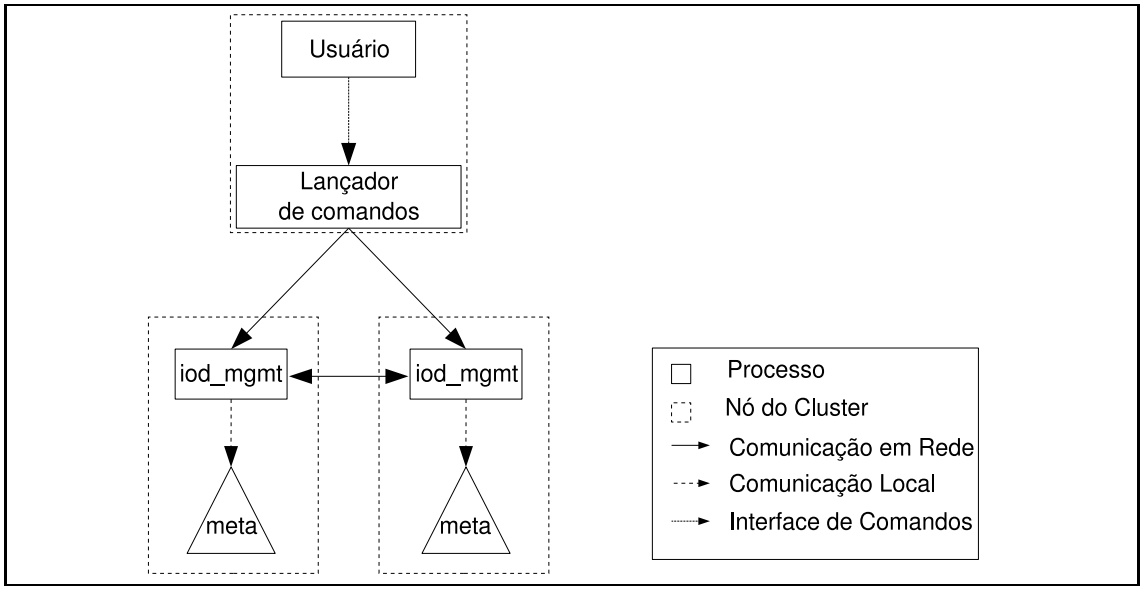

Figura 4.3: Arquitetura do gerenciador de servidores de dados distribuído

4.2.2 Servidor de Dados Virtual

Um servidor de dados virtual (VIOD), no NFS, representa um serviço de armazenamento, onde o metasservidor armazena uma fração do arquivo. Isto permite adaptar a maneira que os dados são realmente escritos nos servidores de dados, de forma transparente ao servidor de metadados. Os VIODs combinados com o gerenciador de servidores de dados, funcionam como uma camada de configuração entre os metasservidores e os servidores de dados.

Para a implementação dos VIODs, a associação estática entre servidores de dados e blocos de dados foi substituída por uma função. Esta função retorna um servidor de dados responsável pelo bloco de dados, de acordo com a disposição atual dos nós. Com esta função, o dinamismo de servidores de dados é isolado do restante do código do servidor de metadados.

No dNFSp2, o esquema de VIODs foi adaptado para implementar a noção de grupos de armazenamento apresentado na Seção 3.1. A principal mudança realizada na implementação já existente de VIODs foi permitir que um servidor de dados possa fazer parte de mais de um grupo de armazenamento. Na versão original, um servidor de dados estava associado a no máximo um VIOD.

Um novo atributo foi adicionado às mensagens entre o metasservidor e o IOD, permitindo assim que um IOD seja associado a mais de um VIOD. Este novo atributo indica a qual VIOD pertencem os dados. Além disso o armazenamento dos dados é feito em um diretório separado para cada VIOD. Isso evita que os dados de um VIOD sejam sobrepostos por dados de outros quando um IOD pertence a vários VIODs.

Outra característica dos VIODs é a possibilidade de incluir replicação de dados entre servidores de dados de um mesmo VIOD. Além disso, é possível redefinir quais os nós associados a um determinado VIOD de forma transparente ao restante do sistema. O número de VIODs existentes no sistema deve ser determinado no momento da instalação do sistema de arquivos. Os servidores de dados que se associam dinamicamente ao sistema de arquivos passam a fazer parte de um desses VIODs de acordo com as necessidades do sistema de arquivos.

4.2.3 Replicação de Dados

A existência de servidores de dados virtuais no NFSp possibilita a replicação entre servidores de dados que pertencem ao mesmo servidor de dados virtual. A existência de dois níveis de abstração (IODs e VIODs) permite que sejam criadas estruturas semelhantes aos níveis de RAID compostos. O esquema utilizado no NFSp é o RAID $1+0$, que é ilustrado na Figura 4.4. Neste caso os dados dos IODs que fazem parte de um mesmo VIOD estão replicados entre si utilizando RAID 1. Por outro lado, o conjunto de VIODs representa o RAID 0, armazenando frações diferentes dos dados.

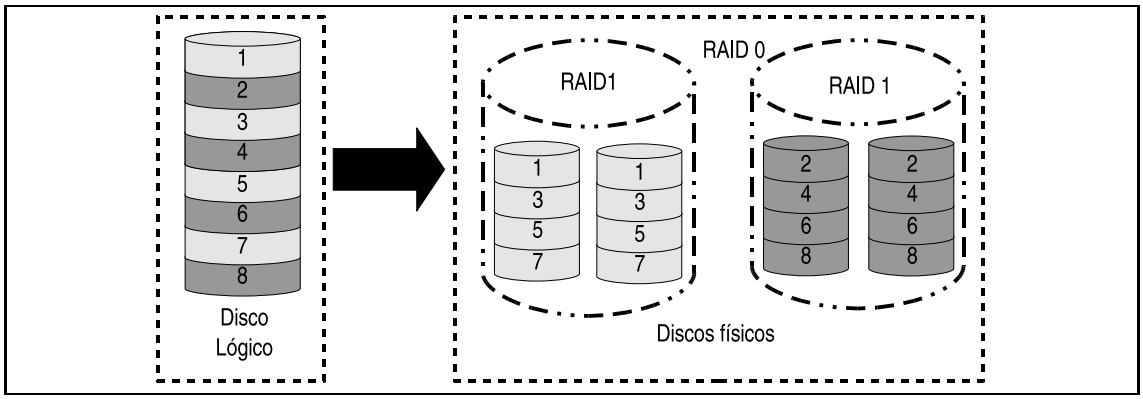

Figura 4.4: Distribuição de dados em uma sistema baseado no padrão RAID $1+0$

A maneira de replicar dados implementada no NFSp é o repasse de escritas (*write forward*). Quando configurado para trabalhar neste modo, o metasservidor deve repassar a requisição a um dos IODs que pertence ao VIOD responsável pelos dados. Juntamente com o conteúdo dos blocos, é enviada a lista dos IODs que pertencem a este VIOD. O IOD que recebe os dados os retransmite ao próximo IOD da lista. A retransmissão ocorre até que o último IOD receba os dados. Este último é o responsável por enviar a confirmação de escrita ao cliente, como mostra a Figura 4.5.

O número de IODs envolvidos na operação de escrita com replicação corresponde ao número de IODs que fazem parte do mesmo VIOD. Portanto, este número depende da quantidade de IODs e de VIODs no sistema. Nem todos os IODs do sistema são utilizados durante uma operação de escrita. Apenas os que pertencem ao mesmo VIOD recebem a retransmissão dos dados. Na maioria dos casos, os IODS envolvidos representam uma pequena porcentagem dos IODs do sistema.

A principal vantagem do repasse de escritas é deixar o controle de perda de mensagens para o protocolo de nível superior, que no caso é o RPC. Passado um tempo limite (*timeout*) após o envio da solicitação, o cliente considera que a mensagem foi perdida e reenvia a solicitação, esperando que esta solicitação seja recebida. O número de mensagens trocadas por este método é a sua principal desvantagem. Isso porque, a quantidade de transmissões de dados necessária é proporcional ao número de servidores de dados que pertencem a um mesmo VIOD. Este fato faz com que o desempenho da escrita com replicação seja pior que a escrita sem o uso de réplicas, onde os dados são transmitidos a apenas um servidor de dados.

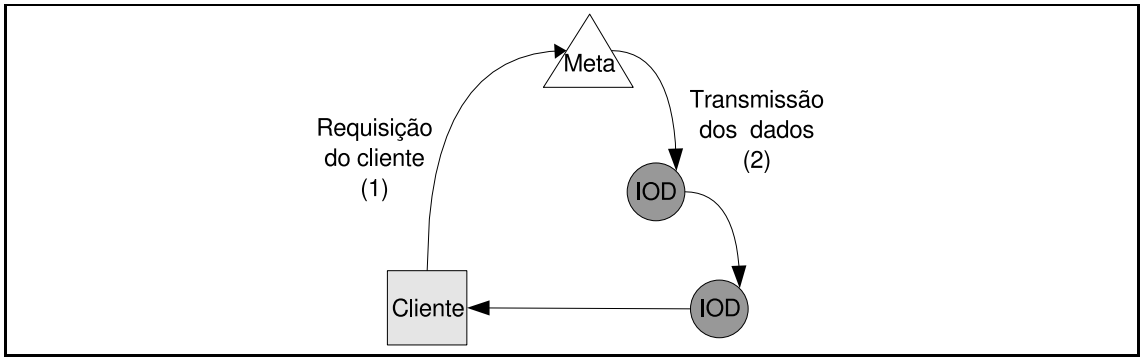

Figura 4.5: Replicação de dados entre servidores de dados do dNFSp utilizando repasse de escritas

Por outro lado, a eficiência da leitura de dados é maior quando existem réplicas. Isso é obtido porque um IOD diferente é escolhido a cada operação de leitura destinada a um mesmo VIOD. Esta escolha é feita de forma circular entre os IODs de um mesmo VIOD, distribuindo assim a carga entre estes servidores de dados.

Para que os mecanismos de coerência de réplicas desenvolvido para o NFSp pudessem ser utilizado no dNFSp2 foi preciso adaptar a sincronização entre servidores de metadados. Esta adaptação foi apresentada na Seção 4.2. Além disso, foi necessário distribuir as informações sobre os VIODs entre os servidores de metadados, pois o dNFSp1 não utiliza o mecanismo de VIODs. Isso foi obtido através do desenvolvimento da versão distribuída do gerenciador de servidores de dados.

4.3 Expansão do Sistema de Arquivos pelo Administrador

A abstração oferecida pelo mecanismo de VIODs permite que os servidores de dados sejam inseridos e removidos do sistema de arquivos de forma transparente ao servidor de metadados. Por intermédio do gerenciador de IODs o administrador pode gerenciar o dinamismo dos servidores de dados. Os nós podem ser inseridos manualmente pelo administrador através da opção addiod. Esta opção recebe como parâmetro o endereço do servidor de dados e o identificador do VIOD ao qual o nó será acoplado. Este método permite fazer um ajuste fino na configuração do sistema de arquivos. Ele deixa a critério do administrador a decisão de qual VIOD será associado ao novo nó.

Por outro lado, é possível deixar a cargo do sistema de arquivos a decisão de como inserir um novo servidor de dados. Para isso, o servidor de dados deve se auto-anunciar ao sistema de arquivos, como indicado na Figura 4.6. Esta notificação é feita através do envio, em *broadcast*, de uma mensagem UDP, que é tratada por um dos gerenciadores de IOD. Para que esta mensagem seja enviada, é preciso iniciar o servidor de dados com a opção notify. Ao receber a mensagem, o gerenciador analisa a configuração atual do sistema e determina qual é o VIOD que possui maior necessidade de um novo IOD. Feita a escolha, o IOD é notificado de seu novo grupo de servidores de dados.

Normalmente o VIOD escolhido é o que possui o IOD mais sobrecarregado. Esta escolha tem como objetivo oferecer uma melhor distribuição de carga entre os servidores, como foi apresentado na Seção 3.2.1. Uma vez escolhido o VIOD,

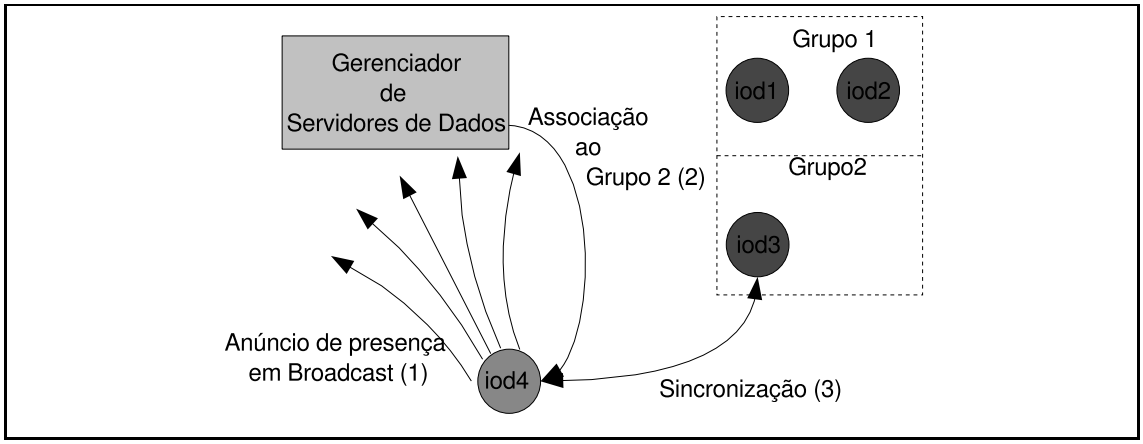

Figura 4.6: Inserção de novos servidores de dados: (1) o servidor anuncia a sua presença; o gerenciador de servidores de dados notifica a qual grupo o nó deve se associar (2); sincronização com um dos membros do grupo (3)

os dados são migrados de um dos IODs deste VIOD para o novo IOD. O uso de um controle de versão de blocos permite a existência de escritas durante a etapa de transferência de dados. Após o final da sincronização, o servidor de dados inserido pode tratar as requisições do usuário de maneira igual aos demais servidores de dados.

A operação de associação é repetida enquanto a diferença entre o número de VIODs do novo IOD e o número de VIODs do IOD mais sobrecarregado for maior que um. Após esta condição de parada ser satisfeita, pode-se considerar que a distribuição de VIODs entre os IODs está equilibrada. A repetição da operação de associação pode fazer com que um IOD seja associado a mais de um VIOD. Isso permite oferecer um balanceamento de carga melhor que o obtido com a associação do novo servidor de dados a apenas um VIOD.

Na notificação em *broadcast*, a perda de mensagens não é tratada pelo mecanismo de dinamismo de servidores de dados. Portanto, fica a cargo de quem insere o novo servidor de dados conferir se ele foi inserido corretamente no sistema. Isso pode ser feito verificando o estado do sistema, através da ferramenta *iod_mgmt*, ou observando as mensagens impressas pelo IOD no momento da execução.

4.4 Inserção de Servidores Temporários de Dados

A inserção de nós iniciada pelo usuário (HERMANN et al., 2006) tem grande parte de sua implementação realizada no gerenciador de servidores de dados. Através de comandos enviados ao *iod_mgmt* é possível especificar quais nós fazem parte de uma aplicação que contém servidores temporários de dados. Além disso, ele permite notificar o sistema de arquivos da criação e finalização da execução das aplicações. O *iod_mgmt* recebe as solicitações do usuário e avalia a situação atual do sistema de arquivos. A partir delas, ele toma as decisões de como tratar esta solicitação. O resultado é repassado aos servidores de dados e metadados para que eles sejam postos em prática. Apesar de todas as operações de configuração serem tratadas pelo *iod_mgmt*, são os servidores de dados e de metadados os responsáveis por determinar como as requisições de E/S do usuá-

rio são tratadas, utilizando pra isso, as configurações repassadas pelo *iod_mgmt*.

4.4.1 Interface com o Usuário

A interface entre o usuário e o gerenciador de servidores de dados é feita através da linha de comando. Três novas opções foram criadas para definir a associação dos nós a uma aplicação. O primeiro deles é o create-execution, que tem como objetivo notificar o sistema de arquivos que se deseja executar uma aplicação com servidores temporários de dados. Esta opção cria um espaço para o gerenciamento desta nova aplicação e retorna ao usuário um identificador único para a execução criada. Este identificador é utilizado nos passos subseqüentes para determinar a qual aplicações as operações são destinadas.

A definição dos nós de processamento que pertencem a uma aplicação é feita através da opção application-node. Esta opção recebe como parâmetro o endereço do nó de processamento e o identificador da aplicação. Ao receber esta solicitação, o *iod_mgmt* insere este nó na lista de clientes da aplicação e transmite esta informação aos servidores de metadados.

A inserção de servidores temporários é feita pelo próprio servidor de dados no momento de sua inicialização. A opção application-iod foi adicionada para diferenciar um IOD temporário dos demais. Esta opção recebe como valor o identificador da aplicação à qual o IOD está associado, valor este que foi obtido através da opção create-execution. Ao inicializar o servidor de dados com esta opção, o sistema de arquivos é notificado de que um novo servidor temporário deve ser inserido.

Após inserir todos os servidores de dados e nós de processamento, o usuário pode iniciar a execução da aplicação. A existência de servidores temporários de dados é transparente à aplicação. Portanto, não é preciso alterá-la para executar num ambiente com servidores temporários de dados.

Quando uma aplicação finaliza sua execução, o usuário deve notificar este fato ao sistema de arquivos. Isto é feito através do comando delete-execution, seguido do identificador da aplicação. O *iod_mgmt* ao processar esta operação envia uma mensagem aos servidores temporários indicando qual é o servidor permanente responsável por receber os seus dados. Finalizada a sincronização entre os servidores de dados, a representação da aplicação é excluída dos metasservidores e os nós podem ser liberados. Como meio de oferecer maior conforto aos usuários, possível integrar esta etapa de finalização ao escalonador do *cluster* (PBS, OAR), para que , se acaso o usuário não informar o fim da execução, o fato de liberar os nós acabe disparando o processo de transferência de dados entre servidores temporários e permanentes.

Um exemplo de seqüência de comandos utilizando a interface do *iod_mgmt* é mostrado na Figura 4.7. Para facilitar a utilização desta ferramenta foi desenvolvido um lançador de aplicações chamado dnfsprun. Este lançador interage com o escalonador do *cluster*, no caso testado o OAR. Com isso, basta informar o número de nós destinados ao armazenamento, juntamente com as informações necessárias para executar a aplicação.

4.4.2 Caminho de Dados com Servidores Temporários de Dados

Todas as decisões de configuração são feitas pelo *iod_mgmt*. Porém, durante a execução, o caminho percorrido pelos dados é definido pelos servidores de meta-

```
iod_mgmt --create-execution
0  #resultado do "create execution"
iod_mgmt --application-node="client1 0"
ssh iod1 iodng --applicationiod=0
ssh client1 mytests.sh
Resultados da aplicação
iod_mgmt --delete-execution=0
```

Figura 4.7: Exemplo de inserção de servidores temporários de dados utilizando a interface de comandos do gerenciador de servidores de dados: *iod_mgmt*

dados. Isso é feito através de uma tabela de mapeamento entre os nós de armazenamento e nós de processamento. Ela permite determinar de forma direta quais os servidores de dados e clientes que pertencem a uma mesma aplicação.

A tabela de mapeamento é construída assim que o servidor de metadados recebe as configurações enviadas pelo *iod_mgmt*. Utilizando a tabela, o servidor de metadados procura a aplicação associada ao cliente que enviou uma requisição de E/S. Se não existe nenhuma entrada nesta tabela correspondente a este cliente, considera-se que ele não faz parte de nenhuma aplicação, e a requisição é transmitida a um dos IODs permanentes. No caso de existir uma entrada na tabela, o IOD escolhido é determinado a partir da lista de servidores temporários da aplicação.

As requisições a um mesmo bloco de dados podem ser tratadas por grupos de servidores de dados diferentes, dependendo da configuração do sistema, como é ilustrado na Figura 4.8. Esta multiplicidade de caminhos de dados pode evitar sobrecarga de servidores. Por serem acessados por apenas uma aplicação, os IODs temporários tendem a ser menos requisitados que os servidores permanentes e, portanto, podem oferecer maior desempenho.

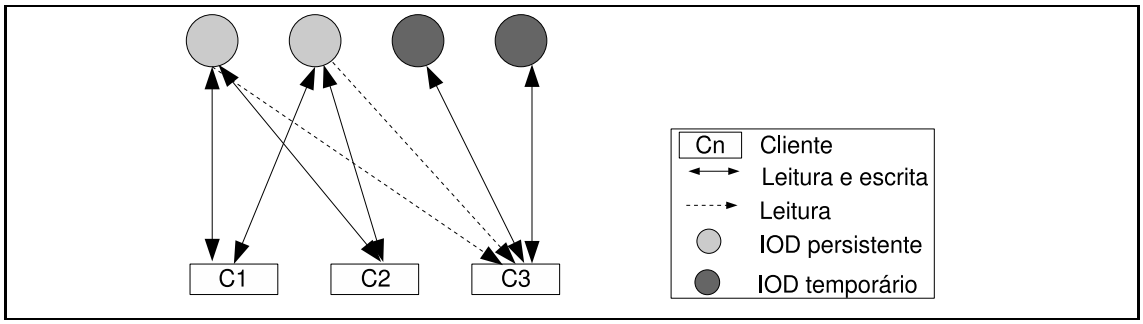

Figura 4.8: Associação entre os clientes (nós de processamento) e os servidores de dados de destino utilizando servidores temporários de dados

As requisições de leitura transmitidas aos servidores temporários de dados nem sempre podem ser respondidas diretamente por eles. Isso ocorre nos casos onde os dados não existem no servidor temporário. Neste caso, a requisição é repassada a um dos servidores permanentes que pertence ao mesmo VIOD. Como servidores de dados temporários e permanentes que pertencem a um mesmo VIOD são responsáveis pelo mesmo conjuntos de dados, se o dado existir ele será encontrado no servidor de dados permanente.

O uso de servidores temporários de dados pode trazer benefícios para aplica-

ções baseadas em iterações. Neste tipo de aplicação, os dados produzidos numa etapa são utilizados como entrada na etapa seguinte. No entanto, aplicações que necessitam ler uma grande quantidade inicial de dados não conseguem tirar proveito deste mecanismo durante a primeira leitura. Os dados criados antes da inserção de servidores temporários ficam armazenados nos servidores permanentes. Assim, o desempenho de leitura de dados que não se encontram nos servidores temporários fica limitado à capacidade dos servidores permanentes.

4.4.3 Sincronização entre Servidores de Dados

Quando a aplicação chega ao final, é preciso notificar o sistema de arquivos que se deseja remover os seus servidores temporários de dados. Neste instante, inicia-se a transferência de dados entre servidores temporários e permanentes. Uma vez que os servidores temporários de dados deixam de existir após a liberação dos recursos, as informações armazenadas por eles devem ser copiadas para os IODs permanentes para que possam ser acessados posteriormente.

O gerenciador de servidores de dados, ao receber a notificação de finalização, envia uma mensagem aos servidores temporários de dados (ver Figura 4.9). Na mensagem enviada está incluído o identificador do IOD persistente ao qual ele deve transferir seus dados. Após a transmissão é possível excluir os servidores temporários de dados e liberar os nós alocados.

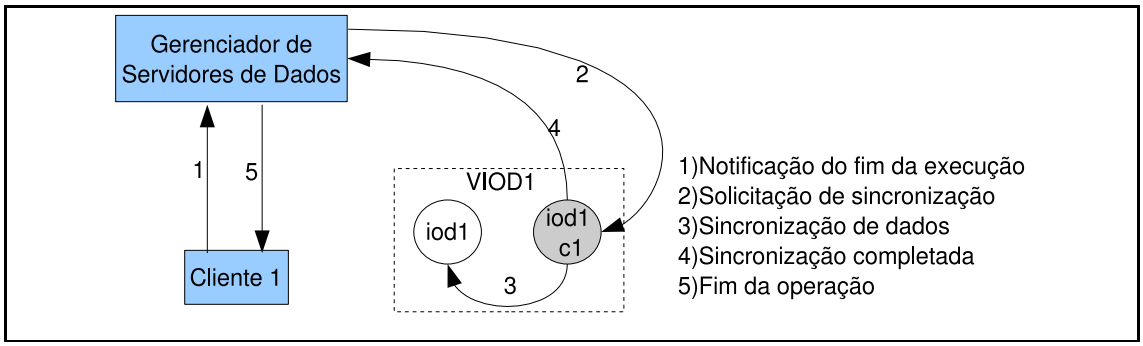

Figura 4.9: Fim da execução de uma aplicação contendo servidores temporários de dados, antes de liberar os nós é preciso transferir os dados dos servidores temporários para os permanentes

Como o envio de dados na etapa de sincronizações entre servidores e a recepção de requisições enviadas pelos clientes são tratados pelo mesmo fluxo de execução, é preciso tomar medidas para evitar que ocorra postergação indefinida (*starvation*) de uma das operações. Este tipo de problema pode ocorrer se uma operação tiver prioridade sobre a outra e sempre existirem requisições da operação prioritária a serem tratadas. Portanto, uma constante é definida para determinar a probabilidade de enviar um bloco de sincronização de dados, mesmo que existam requisições dos clientes. Além disso, a espera por requisições dos clientes tem tempo limitado. Assim, após passar um período pré determinado sem receber mensagens, o servidor de dados passa a tratar do envio de blocos de sincronização, verificando periodicamente se existem novas requisições de leitura e escritas feitas pelos clientes.

O mesmo mecanismo utilizado para sincronizar servidores temporários e permanentes é empregado na sincronização de um servidor de dados inserido pelo

administrador com os demais nós que pertencem ao mesmo VIOD. Em todos os casos, considera-se que existe espaço de armazenamento suficiente no servidor de dados de destino. A comunicação entre os dois servidores envolvidos na sincronização é feita através do protocolo TCP. O uso do TCP mostrou-se mais adequado que o do UDP devido à necessidade de um controle de fluxo para evitar o descarte de pacotes quando o emissor envia mensagens em uma freqüência maior do que a que o cliente pode tratar.

4.5 Tolerância a Falhas

Como apresentado no capítulo anterior, o modelo de falhas suportado pelo mecanismo de inserção dinâmica é o de falha silenciosa (*fail stop*). Com esta restrição, foi possível desenvolver um detector de falhas que executa verificações periódicas do funcionamento dos servidores de dados. Esta função é realizada pelo gerenciador de servidores de dados.

Um dos gerenciadores de servidores de dados é escolhido para verificar os servidores de dados. O gerenciador eleito é sempre aquele que possui o menor identificador, como ilustra a Figura 4.10. A cada intervalo de tempo, ele envia mensagens (*ping*) a todos os servidores de dados do sistema. Os possíveis estados de um servidor de dados em relação ao detector de falhas é ilustrado no autômato da Figura 4.11. Um servidor que não responde durante um ciclo de mensagens de *ping* é considerado suspeito. Ele somente deixará de ser suspeito quando enviar uma mensagem de resposta ao *ping*. Se um servidor de dados suspeito em um ciclo de verificação não responde ao ciclo subseqüente, considera-se que ele falhou.

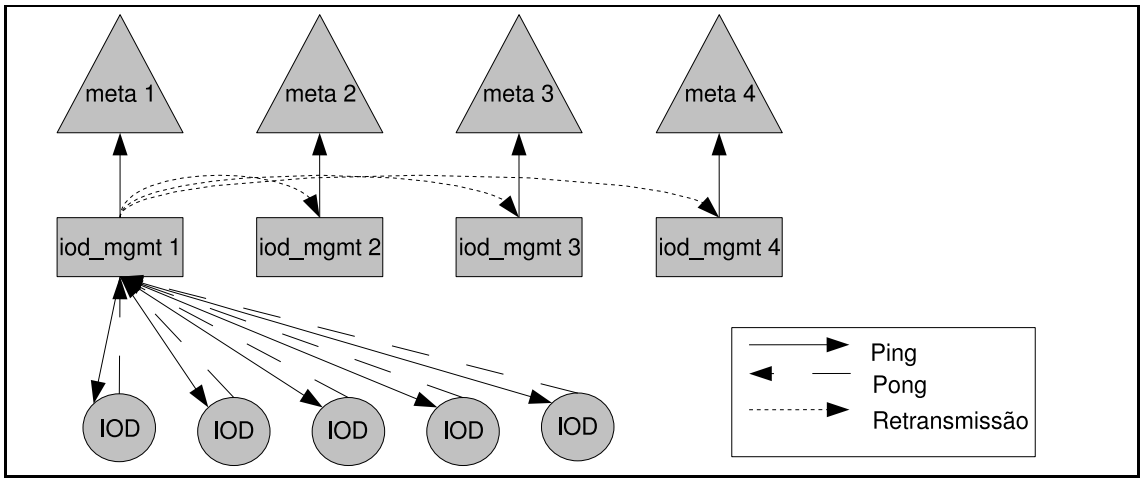

Figura 4.10: Mecanismo de detecção de falhas de servidores de dados permanentes baseado em *ping*

Este método de diagnóstico no nível do sistema tem como principal ponto frágil sua natureza centralizada. Se o detector de falhas parar de funcionar, o mecanismo de reconfiguração não será ativado após a falha de um servidor de dados. Porém, as falhas nos servidores de metadados não são consideradas neste trabalho. Isso porque ele está focado no dinamismo de servidores de dados e não no dinamismo de servidores de metadados. Desta forma, o uso de um algoritmo centralizado para detecção de falhas continua em conformidade com os objeti-

vos. A partir do momento que for levado em consideração a falha de servidores de metadados, devem ser utilizados mecanismos de diagnóstico distribuído. Um exemplo são os algoritmos baseados em DSD (*Distributed System-level Diagnosis*) (JR.; BUSKENS, 1991; DUARTE JUNIOR; NANYA, 1998) que decidem de forma distribuída o estado dos nós do sistema.

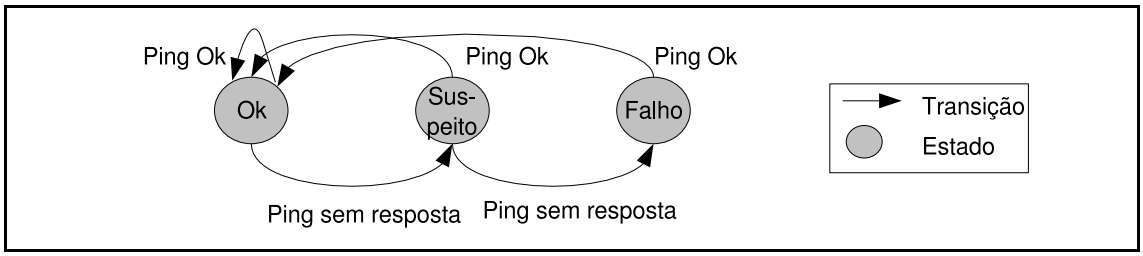

Figura 4.11: Autômato de estados de um servidor de dados de acordo com as respostas às mensagens de *ping*

Se o sistema for configurado corretamente, é possível garantir que o sistema continue suportando falhas após a perda de um servidor de dados. O número de réplicas existentes de uma determinada informação determina quantas falhas simultâneas o sistema de arquivos pode suportar sem que isso afete a disponibilidade dos dados. Neste trabalhou adicionou-se a possibilidade de determinar o número mínimo de réplicas de dados existentes no sistema. Isso é feito através da opção miniods, seguida do número mínimo de réplicas.

Quando é detectada a falha em um servidor de dados, o *iod_mgmt* verifica se o número de réplicas no sistema continua de acordo com o mínimo definido. Isso é feito verificando se todos os VIODs possuem pelo menos tantos VIODs quanto especificado pela opção miniods. Quem faz esta verificação é a mesma instância do *iod_mgmt* responsável pela detecção de falhas. Se a restrição for satisfeita, nada precisa ser feito. Caso contrário, é preciso eleger um outro servidor de dados para substituir o que falhou. Para isso, o gerenciador de servidores de dados verifica se algum VIOD possui um número de IODs maior que o mínimo obrigatório. Se este for o caso, basta mover um de seus IODs para o VIOD que perdeu o nó.

No entanto, se não existem nós disponíveis, é preciso satisfazer a restrição através do compartilhamento de IODs entre dois ou mais VIODs. Com este compartilhamento, alguns IODs passarão a fazer parte de mais de um VIOD e portanto, estarão sobrecarregados. Esta medida é necessária para permitir que o sistema possa suportar mais falhas no decorrer do tempo, apesar de degradar o desempenho do sistema em alguns casos.

Em ambos os casos, após a escolha do servidor de dados que irá substituir o que falhou, é preciso que este obtenha os dados de um IOD que já pertença ao seu novo grupo. Isso é feito utilizando o mecanismo de sincronização entre IODs. Após o final da transferência o sistema de arquivos esta apto a suportar mais uma falha.

A remoção de servidores de dados por parte do administrador, como por exemplo, a realocação de recursos para outro fim, é tratada da mesma maneira que a remoção por falhas. Porém, ao invés de obter os dados de uma réplica, os dados são obtidos diretamente do nó que será removido. Esta medida permite a remoção explícita de um nó mesmo quando não existem réplicas. Após a transmissão dos dados, o servidor de dados poderá ser removido do sistema.

A Figura 4.12 mostra um exemplo da perda de um servidor de dados por falha. Neste caso, o IOD 4 para de funcionar. Após detectar a perda do nó, o gerenciador de IODS solicita que o IOD 2 faça parte do VIOD 2. Assim o IOD 2 passa a ser compartilhado pelos VIODs 1 e 2.

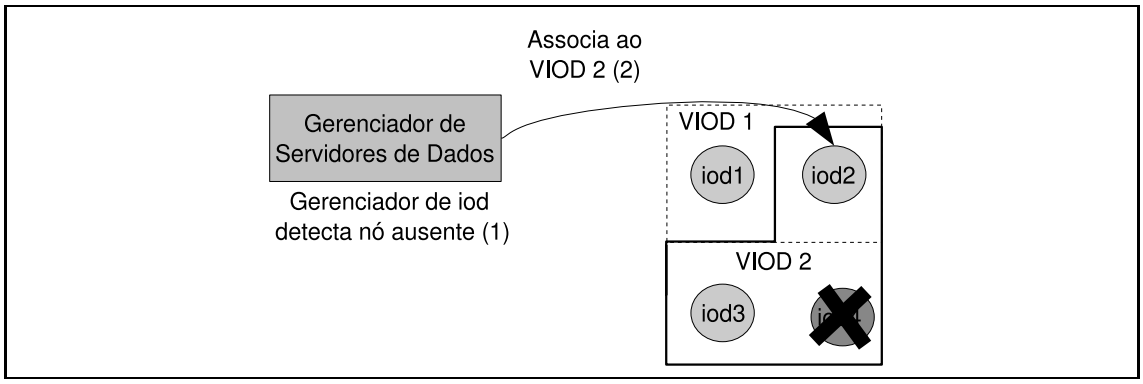

Figura 4.12: Notificação de reconfiguração do sistema de arquivos após à detecção de falha em um servidor de dados

Em alguns casos, é possível estimar o número de réplicas necessárias para que o sistema ofereça alta disponibilidade. Para isso devem existir meios de determinar o Tempo Médio entre Falhas de um nó (*Mean Time Between Failures*, MTBF). Além disso, é preciso estimar o tempo de sincronização entre dois IODs, que depende da quantidade de dados a ser transferida. Ao relacionar estas duas variáveis: o tempo médio entre falhas e o tempo médio de replicação é possível determinar a disponibilidade do sistema (LYU; MENDIRATTA, 1999). O aumento no número de réplicas faz com que a disponibilidade aumente. Na maioria dos casos, o tempo médio de replicação é menor que o tempo médio entre falhas. Nestas situações uma réplica é suficiente para que a perda de um nó não interfira na disponibilidade dos dados.

4.6 Arquitetura Resultante da Implementação

Na implementação apresentada neste capítulo foram inseridos novos módulos na arquitetura do sistema de arquivos dNFSp1. Além disso, a estrutura de comunicação entre os nós também foi alterada. Na Figura 4.13 temos a composição de serviços que fazem parte do dNFSp2.

Dentro de um nó servidor de metadados são executados 3 serviços: o gerenciador de *cache*, o gerenciador de servidores de dados, e o gerenciador de metadados. O primeiro foi inserido para possibilitar a implementação do novo mecanismo de sincronização entre servidores de metadados. É através dele que outros servidores de metadados obtêm o *token* do metasservidor.

Já o gerenciador de servidores de dados é responsável por enviar comandos de configuração aos servidores de dados e de metadados. Além disso, ele recebe comandos externos enviados por seus clientes. Este gerenciador também é o responsável por tratar as mensagens de associação enviadas em *broadcast* por um IOD que deseja participar do sistema de arquivos. O detector de falhas, que verifica periodicamente os nós, é um fluxo de execução dentro deste serviço que, em eventos de falha, inicia a reconfiguração do sistema.

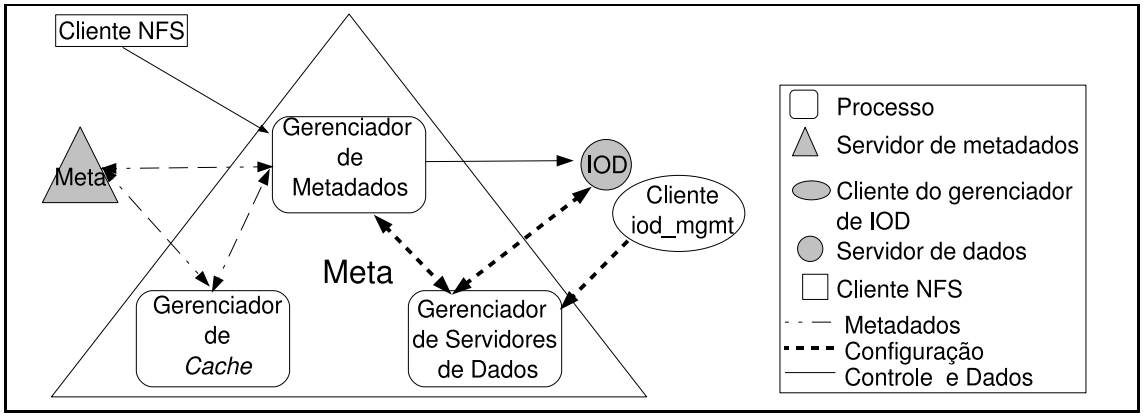

Figura 4.13: Arquitetura resultante da implementação do dNFSp2

O gerenciador de metadados é responsável por acessar os metarquivos armazenados localmente. Através deles se determina em qual VIOD os dados solicitados pelo usuário estão armazenados. Após determinar o VIOD, a requisição é transmitida a um IOD que pertence ao VIOD em questão. No gerenciador de metadados também é feita a associação entre clientes e servidores de dados de uma aplicação que contém servidores temporários de dados.

4.7 Resumo do Capítulo

Neste capítulo foi apresentada a implementação de um sistema de arquivos que é compatível com o modelo de dinamismo de servidores de dados descrito no capítulo anterior. Para este fim, foi utilizado como ponto de partida o sistema de arquivos dNFSp. Este sistema é resultado da tese de doutorado intitulada "Uma Proposta de Distribuição do Servidor de Arquivos em Clusters" (ÁVILA, 2005).

Partindo da versão original do dNFSp, chamada de dNFSp1, foram identificadas as adaptações necessárias para dar suporte às novas funcionalidades e à implementação da nova versão do dNFSp, chamada de dNFSp2. Na etapa inicial, foram feitas alterações no dNFSp1 para atender aos requisitos do dinamismo de servidores de dados. A primeira modificação concentrou-se no mecanismo de sincronização entre servidores de metadados. A versão original deste mecanismo, baseada em um algoritmo de consistência relaxada, não oferece o sincronismo necessário para desenvolver a inserção dinâmica.

Um novo protocolo entre os servidores de metadados foi desenvolvido. Este mecanismo garante que a versão dos metadados vista por um servidor de metadados é a mais recente do sistema. Para atingir este objetivo com eficiência, a localização dos metadados é feita através de uma função de *hash*. Também utilizou-se *cache* de metadados para evitar trocas desnecessárias de mensagens. A consistência da *cache* é feita através de um algoritmo baseado em *token*. Assim, quem possui o *token* tem acesso total sobre os metadados de um determinado arquivo, podendo alterá-lo sem contatar os demais.

Com o mecanismo de sincronização de metadados adaptado, foi necessário desenvolver uma versão distribuída do gerenciador de servidores de dados. A versão original do dNFSp faz uso de uma configuração estática, portanto, o me-

canismo de reconfiguração do sistema não foi portado no momento do desenvolvimento do dNFSp1. O desenvolvimento de uma versão distribuída permite que os comandos de configuração sejam executados em qualquer nó do *cluster* e a partir dele serem transmitidos automaticamente aos servidores responsáveis por tratar esta requisição.

A interface que isola o dinamismo de servidores de dados dentro do metasservidor foi adaptada para tornar-se compatível com a noção de grupos de armazenamento apresentado no capítulo anterior. A estrutura original de servidores de dados virtuais (VIOD) permite que um servidor de dados (IOD) possa fazer parte de apenas um grupo. Para contornar isso, foi necessário possibilitar que um servidor de dados faça parte de mais de um VIOD.

Algumas opções foram adicionadas aos serviços envolvidos no sistema de arquivos dNFSp para permitir a inserção e remoção de nós em tempo de execução. A relação destas opções é apresentada na Tabela 4.1.

Tabela 4.1: Lista de operações envolvidas no dinamismo de servidores de dados

Operação	**Parâmetros**	**Descrição**
iodng –notify	Sem parâmetros	IOD se anuncia em broadcast
iodng –applicationiod	ID da aplicação	Associa um IOD a uma aplicação
iod_mgmt –create-execution	Sem parâmetros	Cria um novo espaço para a aplicação
iod_mgmt –applicationnode	Endereço do nó e ID da aplicação	Associa um cliente a uma aplicação
iod_mgmt –delete-execution	ID da aplicação	Termina uma aplicação
iod_mgmt –addiod	Endereço do IOD e Identificador do VIOD	Insere um servidor de dados
iod_mgmt –deliod	Endereço do IOD e Identificador do VIOD	Remove um IOD de um VIOD
iod_mgmt –miniod	Número de IODs Identificador do VIOD	Número mínimo de IODs por VIOD

O primeiro caso tratado pela implementação é a inserção iniciada pelo administrador. O principal objetivo desta operação é expandir a capacidade do sistema. Para permitir a inserção de novos servidores de dados, o gerenciador de IODs oferece uma opção de linha de comando para notificar o sistema de arquivos. Isso é feito utilizando o comando iod_mgmt –addiod, seguido do endereço do servidor de dados e do identificador do VIOD. Além desta possibilidade é oferecida outra alternativa onde o servidor de dados se auto-anuncia. Para isso, basta inicializar o servidor de dados utilizando o comando iodng –notify. Esta opção faz com que o IOD envie uma mensagem em *broadcast*, anunciando sua presença. Neste caso, a decisão sobre VIOD ao qual o servidor vai ser associado é feita pelo gerenciador de servidores de dados. A primeira alternativa, permite ao administrador fazer um ajuste fino da disposição de nós, enquanto que a segunda opção deixa a associação entre VIOD e IOD por conta do sistema de arquivos.

Apenas a possibilidade do administrador expandir o sistema não garante que as necessidades de todas as aplicações sejam satisfeitas no que diz respeito ao desempenho do armazenamento. Por isso, o segundo caso tratado possibilita ao usuário expandir o sistema de acordo com o perfil de suas aplicações. Alguns dos nós alocados pelo usuário devem ser destinados ao armazenamento. Isso é feito dividindo os recursos em nós de processamento e nós de armazenamento.

Antes de iniciar a execução de sua aplicação, ele pode informar ao sistema de arquivos que uma nova aplicação será executada. Isso é feito utilizando o comando iod_mgmt –create-execution. Este comando retorna um número, que é o identificador da execução. Em seguida, o usuário pode especificar quais nós de processamento fazem parte da aplicação criada, através do comando iod_mgmt –applicationnode, seguido do endereço do nó e do identificador da aplicação. Os nós de armazenamento são adicionados à aplicação através do comando iodng –applicationiod, seguido do identificador da aplicação. Da mesma maneira que no caso anterior, o servidor de dados anuncia sua presença em *broadcast*. Os servidores de dados inseridos por uma aplicação são exclusivos a ela. Como outras aplicações não têm acesso a estes servidores, é possível que esta aplicação obtenha um maior desempenho nas operações de armazenamento de dados.

O último caso tratado ocorre quando um servidor de dados é removido do sistema devido a falhas. Utilizou-se um detector de falhas baseado em verificações periódicas dos servidores de dados para tratar este tipo de evento. Além disso, a possibilidade de ter mais de um IOD por VIOD permite a existência de réplicas no sistema de armazenamento. Assim, quando uma falha é detectada, um outro IOD que contém a réplica dos dados do IOD que falhou pode assumir suas funções. Após a perda de um IOD, o sistema pode reconfigurar-se utilizando os nós que restaram. Passado este período de reconfiguração, o sistema de arquivos está apto a suportar a perda de mais um servidor de dados. Para determinar a quantidade de réplicas de dados existentes no sistema, utiliza-se o comando iod_mgmt –miniod, seguido do número mínimo de IODs por VIOD. Assim, é possível especificar o número de falhas simultâneas de servidores de dados suportada pelo sistema.

Para a implementação do tratamento do dinamismo foi alterada a arquitetura do dNFSp1. No servidor de metadados do dNFSp2 passaram a existir 3 serviços em execução: o gerenciador de *cache*, o gerenciador de metadados e o gerenciador de IODs. Os metadados são transferidos entre os servidores através do gerenciador de *cache*. O servidor de metadados é responsável por tratar as requisições dos clientes. Já o gerenciador de IODs é responsável por receber as reconfigurações e repassá-las aos servidores de metadados.

O sistema resultante desta etapa de desenvolvimento foi validado e avaliado utilizando testes com operações simples e simulações de aplicações científicas. Os resultados da avaliação de desempenho e a validação da implementação são apresentados no capítulo seguinte. Além de testes sobre a implementação do dinamismo de servidores de dados, também são apresentados resultados relativos à etapa de adaptação da versão original.

5 VALIDAÇÃO E TESTES

Para verificar o funcionamento da implementação apresentada no capítulo anterior, foram efetuados testes envolvendo os diferentes cenários tratados neste trabalho. Juntamente com a validação, foram realizados testes de desempenho comparando o sistema desenvolvido com trabalhos semelhantes. Inicialmente, na Seção 5.2, são avaliadas as mudanças resultantes da etapa de adaptação do dNFSp1. Foram realizados testes comparando o dNFSp2 com a versão original, além de comparar com o NFS e o PVFS.

No restante do capítulo, cada um dos casos de dinamismo implementados pelo dNFSp2 foi avaliado. Primeiramente, foi testada a inserção de servidores de dados iniciada pelo administrador. Neste caso foi medido o ganho de desempenho resultante da inclusão de novos servidores de dados, e os resultados são apresentados na Seção 5.3. Após, na Seção 5.4, foi avaliada a inserção iniciada pelo usuário. Diferentes situações foram avaliadas, permitindo observar o desempenho deste mecanismo juntamente com o impacto que ele pode causar nas demais aplicações executando no *cluster*.

Finalmente, na Seção 5.5, foram feitos testes para avaliar o mecanismo de tratamento de falhas de servidores de dados. Neste caso, foi avaliado o desempenho dos sistemas de replicação tanto na escrita quanto na leitura. Também foram feitos testes para observar o impacto da perda de um servidor de dados sobre o desempenho do sistema. Isso porque o sistema deve se reconfigurar após perder um servidor de dados e esta reconfiguração pode resultar em desbalanceamento de carga nos servidores de dados.

5.1 Descrição do Ambiente de Testes

A plataforma de execução utilizada nos testes é o *i-cluster2* (I-CLUSTER 2, 2006) instalado em Montbonnot Saint Martin, França, na sede do *Institut National de Recherche en Informatique et en Automatique - Rhône-Alpes*. O *cluster* é composto por 100 nós. Cada nó está equipado com processadores Itanium2 900 MHz dual com 3 Gb de memória e armazenamento em disco com capacidade de 72Gb, 10000 rpm, SCSI. Todos os nós estão interconectados utilizando rede 1 Gigabit Ethernet, rede Fast Ethernet e rede Myrinet.

Os experimentos foram realizados utilizando a rede 1 Gigabit Ethernet. O sistema operacional instalado é a versão estável do Debian GNU/Linux, com a versão 2.6.15 do núcleo de sistema operacional Linux. A implementação do MPI utilizada com o *benchmark* BTIO é o MPICH 1.2.6. Quando não especificado de forma diferente, todos os resultados apresentados neste capítulo foram obtidos

da média das execuções de cada cenário de testes. Como os testes foram executados em um ambiente controlado, uma bateria de testes contendo 10 execuções foi suficiente para encontrar um valor com desvio padrão menor que 5% do tempo médio, na maioria dos casos. Em outros casos, onde o desvio padrão foi maior que 5% do tempo médio, os resultados foram repetidos até a obtenção de um valor dentro deste padrão.

O número de nós utilizados nos testes foi escolhido a partir do número de clientes da maior configuração possível nos testes feitos com o BTIO, que será apresentado na seção 5.2.3. Isso porque o BTIO possui uma restrição de que o número de clientes utilizados deve ser um quadrado perfeito (1, 4, 9, etc.). Com isso foi escolhido o número de 27 clientes para a maioria dos casos, o que permite a existência de três grupos de 9 clientes, no caso do BTIO. Já os números de servidores de metadados e de dados também foram escolhidos tendo como base múltiplos de 9, evitando assim um desequilibrio na distribuição de servidores entre os clientes.

5.2 Avaliação do Sincronismo entre Servidores de Metadados

Como discutido na Seção 4.1.1, o mecanismo de consistência de metadados utilizado pelo dNFSp1 não oferece o sincronismo necessário para a implementação do dinamismo de servidores de dados. Na primeira etapa da implementação deste trabalho, este mecanismo foi reformulado para aumentar o sincronismo entre servidores de metadados. Esta medida permite que outras funcionalidades, além da inserção dinâmica, possam ser implementadas com maior facilidade.

Nesta seção será avaliado o desempenho da nova versão do dNFSp. Nos casos testados não são executadas as operações relacionadas ao dinamismo de servidores. Apesar de não serem diretamente utilizados nestes testes, os mecanismos da inserção dinâmica já estão presentes durante estas execuções.

5.2.1 Testes de Desempenho de Base

Este teste tem como objetivo avaliar o dNFSp1 através de operações básicas de sistema de arquivos. Ele é baseado em operações de leitura e escrita de dados. Nenhuma operação de computação é efetuada sobre os dados, evitando assim que o processamento mascare o tempo de transferência. Além de observar o desempenho da transferência de dados, nestes testes é avaliada a influência das operações de metadados quando são executadas estas operações básicas.

Neste teste, cada cliente transfere uma quantidade fixa de dados. Em cada etapa dos testes, os dados são divididos em quantidades diferentes de arquivos. No caso ilustrado na Figura 5.1, cada cliente escreve um total de 50Mb em todos os passos, variando o número de arquivos entre 1 e 128. No primeiro passo, cada cliente escreve um arquivo contendo 50Mb. No segundo passo, foram transferidos 2 arquivos com 25Mb. A mesma regra foi seguida até que cada cliente transferisse 128 arquivos de 400Kb. Este procedimento permite avaliar a influência das operações de metadados na transferência de dados, visto que a quantidade de dados enviada é sempre a mesma. Como a quantidade de dados é dividida em um número variável de arquivos, o número de requisições de metadados entre uma etapa e outra também difere.

Os testes foram executados usando 8 nós como servidores de metadados, 8

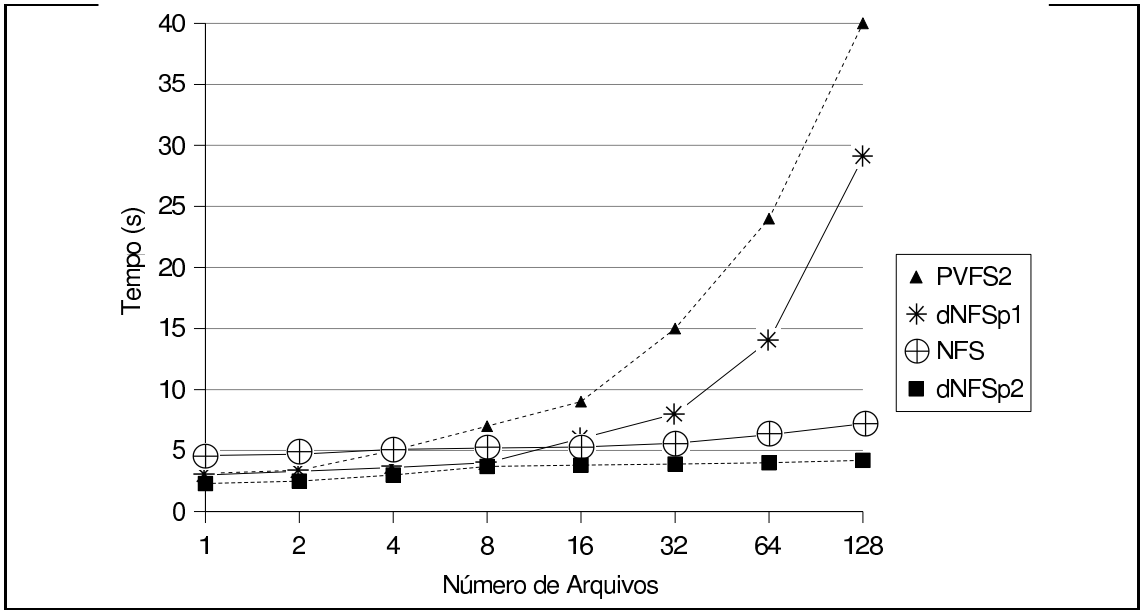

Figura 5.1: Transferência de 50Mb de dados divididos em tamanhos diferentes de arquivos

nós como servidores de dados, e 8 nós para clientes. Os resultados mostram que a nova versão do dNFSp, indicada por dNFSp2, mantém um tempo de transferência próximo do constante em todos os casos, enquanto que o dNFSp1 e o PVFSv2 sofrem uma degradação de desempenho à medida que os dados são divididos em um número maior de arquivos. O menor tamanho de arquivos atingido é de 400kb, que é maior que o tamanho de bloco utilizado pelo dNFSp, no caso, 8Kb. Em consequência disso, o ganho do dNFSp2 comparado com o dNFSp1 não tem sua origem no mecanismo de transferência e sim nas operações de metadados necessárias durante a escrita. A principal diferença no tratamento de metadados das duas versões está na complexidade dos algoritmos de sincronismo. O dNFSp1 possui uma complexidade $O(n)$, pois é preciso contatar todos os servidores até encontrar os metadados desejados, enquanto que o novo algoritmo possui uma complexidade $O(1)$, pois basta acessar o servidor *Hash* para encontrar a versão mais recente. Quando existem poucos arquivos, ambos possuem um desempenho semelhante, pois o número de arquivos criados é pequeno, resultando em menos operações de metadados.

O servidor NFS também mantém um comportamento bastante regular. As operações de metadados em um sistema de arquivos centralizados são todas executadas localmente. Conseqüentemente, o tempo de criação de um arquivo é menor que o gasto pelos sistemas que mantém metadados distribuídos. No entanto, o novo protocolo do dNFSp2 mostrou-se capaz de obter resultados próximos ao do NFS tradicional durante o acesso aos metadados, mesmo que ele os armazene de forma distribuída. Este resultado confirma o baixo *overhead* imposto pelo mecanismo de sincronização baseado em *hash*. O melhor desempenho obtido pelo NFS em alguns casos, como o da escrita de arquivos muito pequenos, não significa que ele possua maior desempenho de E/S que o PVFS e o dNFSp1. Com a diminuição do tamanho dos arquivos, as operações que predominam são as de metadados e são elas que fazem a diferença no tempo obtido.

O desempenho do PVFS2 é degradado principalmente pelo tamanho de escrita dos dados. O tamanho de bloco ótimo, segundo os desenvolvedores do PVFS2, é de 4Mb ou 8Mb. Por outro lado, o menor arquivo escrito nos testes é de 400kb, um valor bastante inferior ao caso ótimo. Outro fator que pode colaborar com a degradação do desempenho é o perfil do teste, que é baseado em escrita seqüencial. A cada nova escrita o tamanho do arquivo é modificado, resultando em modificações dos metadados.

5.2.2 Testes de Desempenho Utilizando DAB

Para dar continuidade aos testes de desempenho no mecanismo de sincronização do dNFSp1, foi utilizado o conjunto de testes chamado *Distributed Andrew Benchmark* (DAB). O DAB é uma implementação baseada no *Andrew Benchmark* (HOWARD et al., 1988). Este foi desenvolvido para testar o desempenho do *Andrew File System* (AFS) e tenta simular a carga de operações de vários usuários acessando um sistema de arquivos.

A versão original foi projetada para executar em uma única máquina. Já a variante desenvolvida para os testes tenta simular a carga de vários nós de computação acessando o sistema de arquivos distribuídos. Isso é feito executando simultaneamente várias instâncias do *benchmark* em diferentes máquinas.

O *benchmark* modificado é composto de cinco fases. Cada uma tem como objetivo sobrecarregar uma característica do sistema de arquivos. Na Fase 1, são criados vários diretórios, que são utilizados nas fases subseqüentes. A Fase 2 copia arquivos de uma árvore de diretórios do sistema de arquivos local para o sistema de arquivos distribuído. A Fase 3 executa operações de *stat*, obtendo os atributos de cada arquivo. Na Fase 4, cada cliente lê os arquivos de seu diretório. Na Fase 5, cada cliente compila um programa fonte. No caso deste teste, o programa compilado foi o Lynx [1]. As Fases 1 e 3 são focadas em avaliar o quanto o sistema de arquivos é eficiente e escalável no acesso aos metadados. As Fases 2 e 4 tentam avaliar o desempenho do acesso aos dados. Já a Fase 5 tenta medir o desempenho do sistema de arquivo em uma situação onde ambos, metadados e dados, são necessários.

A execução é feita de maneira coordenada. Todos os clientes devem completar uma fase antes de iniciar a seguinte, evitando a sobreposição de fases. Para evitar o mascaramento de resultados em consequência de *cache* ou *buffer*, o sistema de arquivo remoto é montado e desmontado no início e no final de cada fase.

Os testes foram executados utilizando 8 nós como cliente, 8 nós como servidores de metadados e 8 nós como servidores de dados. O código fonte utilizado na compilação totaliza 48Mb distribuídos em 1776 arquivos. Os resultados deste teste são apresentados na Figura 5.2 comparando dNFSp1, dNFSp2 e PVFS2.

Nas Fases 1, 2, 3 e 5 a versão utilizando *hash* (dNFSp2) obteve um desempenho melhor que a versão original (dNFSp1). Estas etapas envolvem a criação de arquivos e diretórios. Para criar uma nova entrada no sistema, o servidor de metadados do dNFSp1 deve contatar todos os outros através de rcp, esta comunicação é necessária para verificar a existência do arquivo ou diretório. No dNFSp2, esta verificação é feita contatando apenas o *Hash*. Na Fase 2, os arquivos são copiados para o sistema de arquivos distribuído. Nesta fase combina-se operações

[1] http://lynx.browser.org

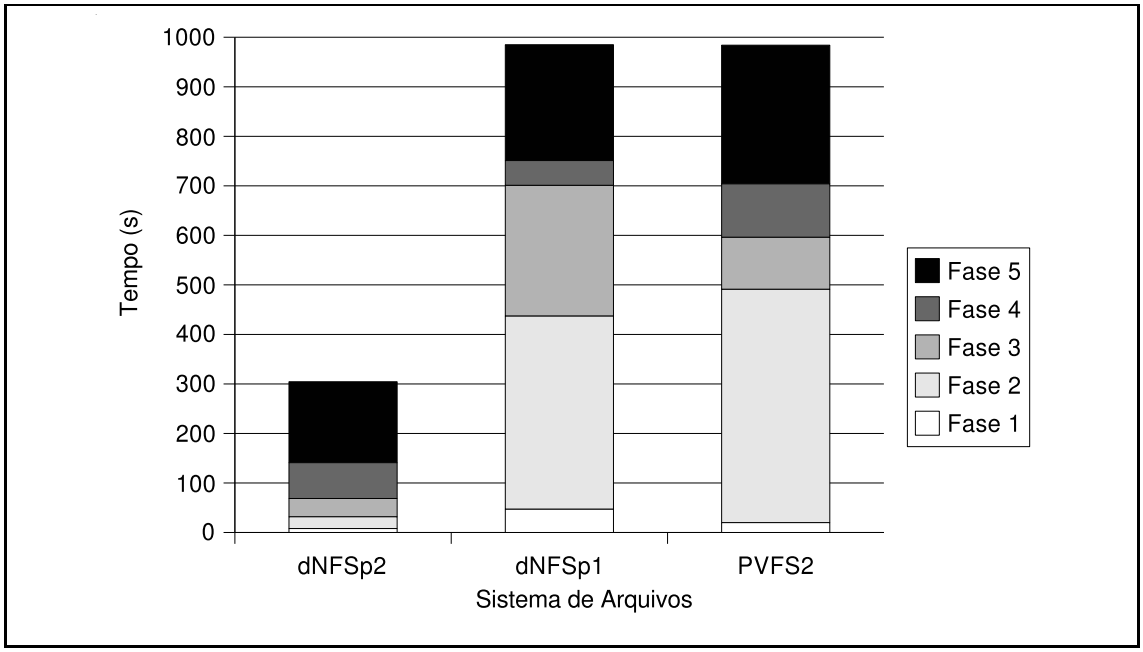

Figura 5.2: Resultados de desempenho das operações de dados e metadados do Distributed Andrew Benchmark

de dados para transferência do conteúdo e operação de metadados na criação dos arquivos. Novamente, a verificação da existência do arquivo solicitada pelo cliente antes da criação tem um peso importante no tempo de execução. O mecanismo de transferência de dados em ambas as versões é o mesmo, e portanto, a única diferença está no tratamento de operações de metadados.

Na Fase 3, os atributos de todos os arquivos no sistema são solicitados por cada cliente através de uma operação de *stat*. O dNFSp1 precisa contatar todos os outros servidores de metadados até encontrar os metadados dos arquivos ainda desconhecidos. Esta operação tem um custo significativo uma vez que é feita através de chamadas a comandos rsh. Já o dNFSp2, ao invés de contatar todos os servidores de metadados, precisa contatar apenas o servidor que contém o *token* para obter os metadados. Além disso esta operação é feita através de uma chamada remota de procedimentos, que possui um custo inferior ao rcp.

Na Fase 4, que é caracterizada pela leitura de dados, a versão original demonstrou um desempenho melhor que o dNFSp2. Os metadados necessários para localizar os dados dos arquivos foram obtidos na Fase 3, e portanto, podem ser acessados localmente. No dNFSp2, os servidores de metadados competem pela aquisição do *token*, exigindo uma grande troca de mensagens entre eles. Para completar esta fase, todos os servidores de metadados devem obter pelo menos uma vez o *token* de cada arquivo. Além disso apenas um servidor de metadados pode possuir o *token* em um determinado instante.

Na última fase, os arquivos fontes copiados nas fases anteriores são compilados. Cada cliente compila os arquivos em um diretório separado. Além das operações de leitura de arquivos fonte, é preciso criar os arquivos binários e escrever o seu conteúdo. Novamente esta operação de criação teve um impacto importante, fazendo com que o dNFSp2 obtivesse um melhor desempenho melhor que o dNFSp1.

Por outro lado, o PVFS2 obteve um desempenho pior que os demais sistemas

em quase todas as fases. Na etapa de compilação o resultado é semelhante para todos, pois o tempo de computação é mais significativo que as operações de entrada e saída. Na fase 1 e 4, o PVFS foi mais rápido que o dNFSp1, visto que estas operações envolvem apenas metadados. No PVFS, o cliente contata diretamente o servidor de metadados que contém a informação desejada. Já no dNFSp1, se as informações não são encontradas pelo servidor de metadados, ele deve contatar todos os demais até encontrar os metadados solicitados. Nas demais fases, o pior desempenho do PVFS2 é causado pelo tamanho médio dos arquivos fonte utilizados no teste. Na seção anterior, foram apresentados testes de escritas de dados contendo arquivos de no mínimo 400k, já nos arquivos fonte utilizados no DAB a média fica em torno de 30Kb. Assim, o baixo desempenho na escrita de arquivos pequenos novamente é o principal delimitador do PVFS2.

5.2.3 Testes de Desempenho Utilizando BTIO

O objetivo deste teste é avaliar os sistemas em uma situação próxima da realidade das aplicações científicas. O sistema de avaliação de desempenho escolhido foi o BTIO. O BTIO é uma extensão do BT (BAILEY, 1991). Este é baseado em algoritmos de dinâmica de fluídos (*Computational Fluid Dynamics*, CFD) para resolver equações tridimensionais de Navier-Stokes. O BTIO utiliza o mesmo método computacional empregado pelo BT. Porém, operações de entrada e saída foram inseridas, forçando a escrita dos resultados para o disco.

O número de processos executando o BTIO deve ser um quadrado perfeito (1, 4, 9, 16, etc.). O tamanho do problema é escolhido através da especificação de uma classe. Cada classe corresponde às dimensões de uma matriz cúbica a ser resolvida pela aplicação: Classe A (64^3), Classe B (102^3), Classe C (162^3). A classe escolhida foi a Classe A pois mostrou uma boa relação entre processamento e operações de sistema de arquivos.

Uma outra opção do BTIO é o método de escrita de dados no sistema de arquivos. Para os testes, foi escolhido a variante *epio*. Neste tipo de escrita, cada nó de processamento escreve seus resultados em um arquivo separado. Assim, não existe entrelaçamento de escritas efetuadas por clientes diferentes sobre um mesmo arquivo. Esta situação permite obter o melhor desempenho do dNFSp1, sem a desvantagem da incoerência de metadados, uma vez que os arquivos não são compartilhados pelos clientes.

Os resultados da Figura 5.3 mostram o tempo de execução do BTIO para cada sistema de arquivos. Para cada caso os testes foram repetidos no mínimo 10 vezes e calculado a média dos valores. Cada um dos sistemas de arquivos foi configurado com 12 servidores de dados e 12 servidores de metadados. O número de clientes variou entre o mínimo de 4 nós e o máximo de 49 nós.

Tanto o dNFSp1 quanto o dNFSp2 mantêm um perfil semelhante de resultados. Uma pequena diferença de desempenho a favor do dNFSp2 ocorre quando o número de clientes aumenta. A maioria das operações realizadas pelo BTIO são de escrita de dados, enquanto que operações de metadados são feitas na criação dos arquivos. É neste momento, quando os arquivos são criados, que o dNFSp2 tira vantagem sobre a versão original. Após a criação do arquivo no dNFSp1, o servidor de metadados que possui uma cópia local do metarquivo não necessita contatar os outros durante as operações de escrita. O dNFSp2 diminui a comunicação entre servidores de metadados através do uso de *cache*. Após o primeiro

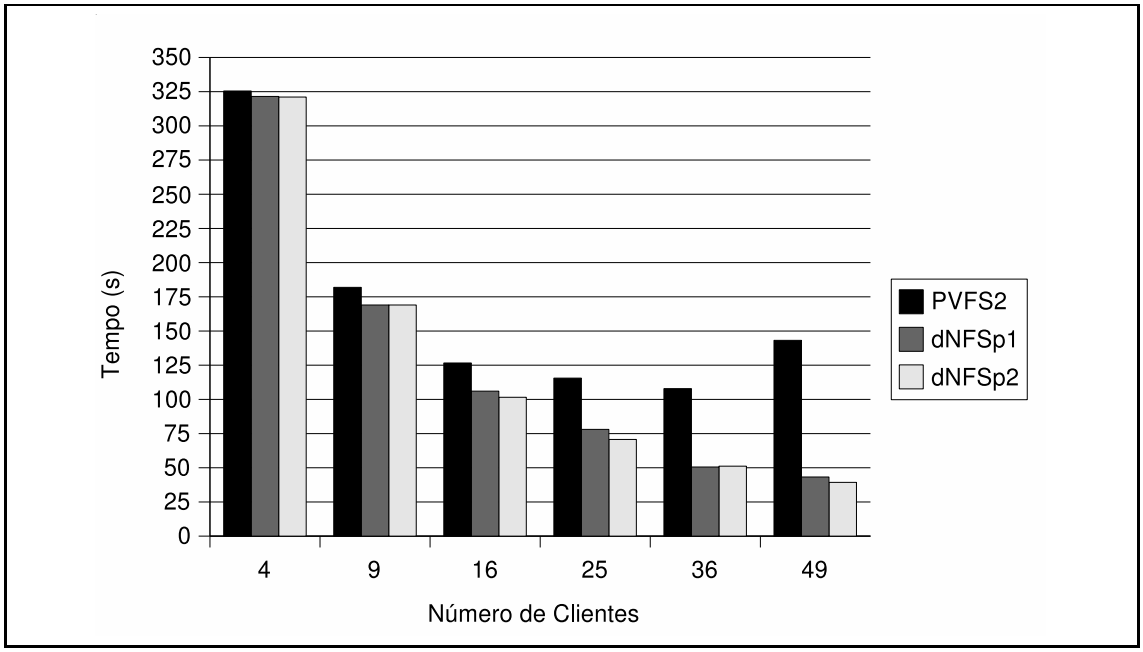

Figura 5.3: Teste de desempenho utilizando o *benchmark* BTIO-epio, que simula operações de uma aplicação científica

acesso aos metadados, o servidor de metadados armazena as informações na *cache*. Como cada cliente escreve em um arquivo diferente, o servidor de metadados não tem sua *cache* invalidada por outro servidor. Isso garante que, a partir do primeiro acesso, as operações de escrita sejam processadas sem a necessidade de comunicação entre metasservidores.

Independentemente do número de clientes, o dNFSp2 superou a versão original em todos os casos. Isso mostra que os mecanismos de *Hash* e *cache* conseguem conciliar o maior sincronismo entre os servidores de metadados com um desempenho semelhante ao do dNFSp1. Isto pode ser observado inclusive em situações onde o dNFSp1 tem o seu melhor caso. O BTIO é um exemplo onde o dNFSp1 pode obter o melhor de seu desempenho sem a desvantagem da incoerência de metadados, pois os arquivos não são acessados simultaneamente por clientes diferentes.

O PVFS2, mesmo possuindo comunicação direta entre cliente e servidores de dados, tem seu desempenho degradado com o aumento do número de clientes. Novamente, o tamanho das escritas foi importante no desempenho deste sistema. Com o aumento do número de clientes, a quantidade de dados escrita por cada um diminui. Por ter um pior desempenho em escritas pequenas, o PVFS2 acaba limitando o tempo de execução do BTIO. Observa-se também que no caso contendo 49 clientes o tempo de escrita não compensa a distribuição da computação em um número maior de nós, pois o tempo de execução com 49 clientes é maior que o obtido com 25 clientes. Esta limitação mostra que o PVFS2 exige que uma porção maior do problema seja dedicada a um nó para que o tempo de escrita seja compensado pelo ganho em processamento.

A cada escrita de dados o cliente PVFS2 deve contatar o servidor de metadados para atualizar o tamanho do arquivo. Isso ocorre porque o BTIO acrescenta conteúdo ao final do arquivo, alterando o seu tamanho. Como o tamanho do arquivo faz parte dos metadados, este deve ser atualizado, resultando em maior

comunicação. No dNFSp, é o próprio servidor de metadados que trata da escrita. Assim, a atualização dos metadados pode ser feita localmente, sem a necessidade de comunicação extra.

5.3 Inserção de Servidores de Dados Iniciada pelo Administrador

A capacidade de expandir o sistema de arquivos dNFSp está diretamente ligada ao número de servidores de dados virtuais (VIODs) configurados na sua instalação. Um arquivo é distribuído entre os servidores de dados de acordo com o número de VIODs. O conteúdo é fracionado em blocos de tamanhos iguais, que são distribuídos de forma circular entre os VIODs. Por exemplo, quando o sistema de arquivos possui 5 VIODs, seus arquivos são divididos em 5 partes, e cada uma é armazenada em um VIOD diferente. Com isso, o número de frações de um arquivo é independente do número de servidores de dados, pois ele está associado ao número de servidores de dados virtuais. A adição de novos servidores de dados não aumenta a capacidade de armazenamento do sistema quando o número de IODs atinge um número igual ao de VIODs. Em consequência desta restrição, é adequado configurar o sistema de arquivos com um número extra de servidores de dados virtuais. Isso permite que novos IODs sejam inseridos e que estes possam assumir a responsabilidades sobre alguns VIODs.

Um exemplo de progressão do número de servidores de dados em um sistema contendo 5 VIODs é ilustrado na Figura 5.4. Inicialmente o sistema possui apenas um IOD, e este é responsável por armazenar os dados de todos os VIODs. Na segunda linha desta figura, é inserido um novo servidor de dados. Para tratar esta inserção, aplica-se o algoritmo descrito na Seção 4.3. Neste algoritmo, o novo servidor de dados passa a tomar conta de um dos VIODs do servidor de dados que possui o número maior de VIODs. A transferência de VIODs é repetida até que a diferença entre o número máximo e mínimo de VIODs por IODs seja menor que 2. De acordo com este algoritmo, o novo IOD passa a tomar conta dos VIODs 4 e 5, enquanto que o IOD 1 continua responsável pelos VIODs 1, 2 e 3.

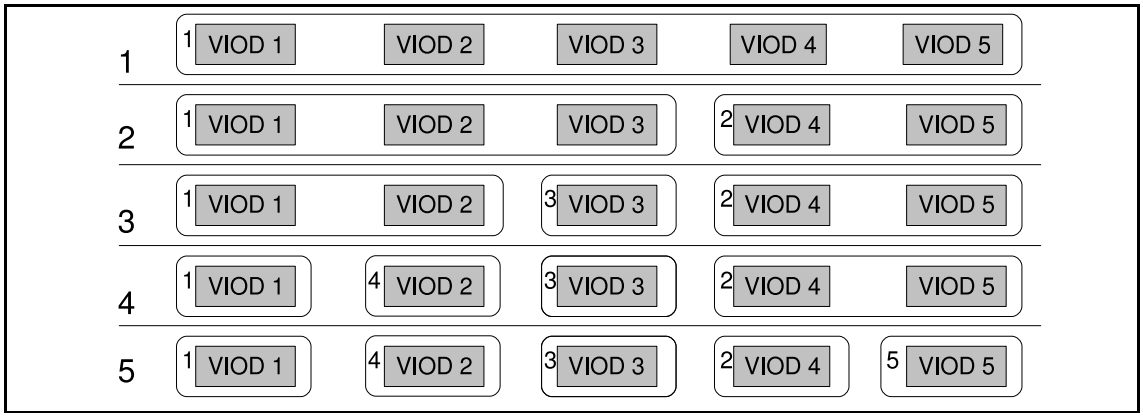

Figura 5.4: Exemplo de progressão da associação entre VIODs e IODs de acordo com a inserção de novos servidores de dados

Na representação da Linha 3 da Figura 5.4, é inserido um terceiro IOD. Este passa a ser responsável pelo VIOD 3, deixando apenas os VIODs 1 e 2 para o

IOD 1. O mesmo procedimento é repetido nas Linhas 4 e 5, até que o número de IODs seja igual ao de VIODs. O caso onde o número de IODs é igual ao de VIODs representa o ponto limite de expansão do sistema. A partir deste instante a inserção de novos servidores de dados não resulta em aumento da capacidade de armazenamento.

Os resultados de desempenho de uma situação semelhante à da Figura 5.4 são apresentados nas Figuras 5.5 e 5.6. O cenário testado contém 9 VIODs. Para cada novo IOD inserido é obtida a vazão de escrita e a vazão de leitura. Em todos os casos foram utilizados 27 clientes, cada um deles escrevendo 1 Gb de dados e, em seguida, lendo os mesmos. A configuração inicial do sistema de arquivos contém apenas um IOD. Em todos os casos são utilizados 9 servidores de metadados para diminuir o gargalo provocado por um número pequeno de servidores de metadados.

A Figura 5.5 mostra os resultados da escrita juntamente com a curva teórica resultante da inserção de novos IODs. Para obter a curva teórica utilizou-se como base a taxa de transferência do sistema de arquivos com um IOD. Ao serem adicionados mais servidores de dados este valor é multiplicado pela quantidade de IODs no sistema, resultando assim em um crescimento linear da taxa de transferência de dados.

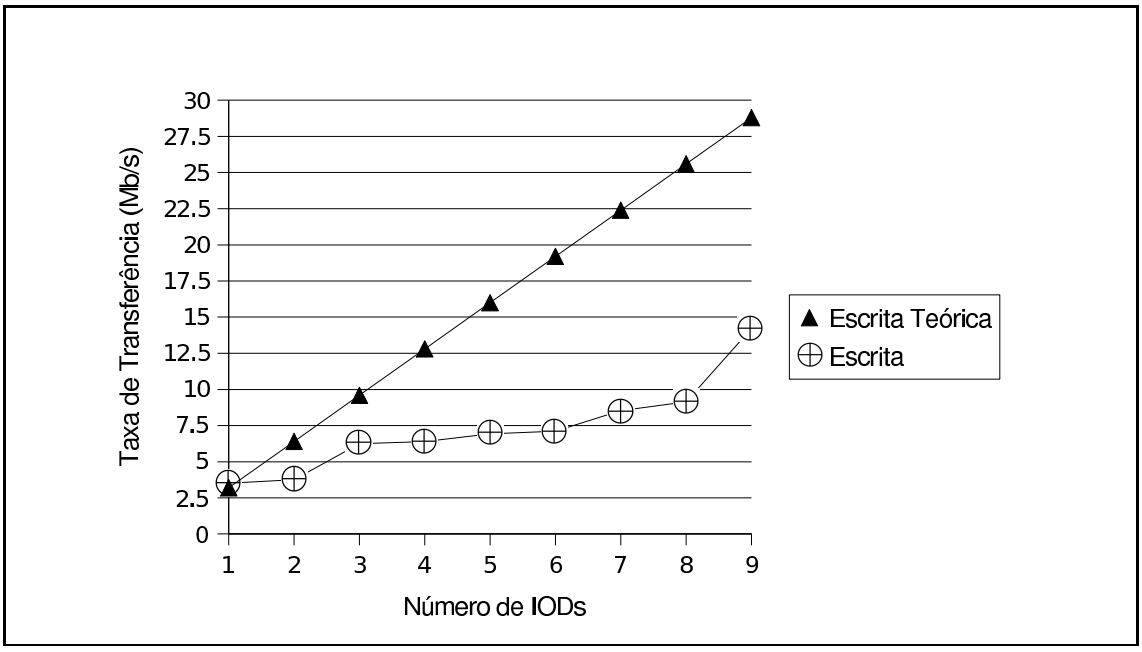

Figura 5.5: Resultado da escrita de 1Gb de dados variando-se o número de IODs associados ao sistema

O ganho de desempenho na escrita obtido pelo dNFSp com a inserção de novos servidores de dados mantém-se distante da curva teórica. Na maioria dos casos, atingindo menos de 50% do desempenho teórico. Um dos principais motivos do distanciamento entre os resultados obtidos e a curva teórica, é a indireção na transferência de dados na escrita. Todas as escritas passam pelos servidores de metadados e após são transmitidas aos servidores de dados. Esta retransmissão faz com que a interface de comunicação dos servidores de metadados fique sobrecarregada, visto que ela deve receber os dados e após retransmitir aos servidores de dados.

Além da necessidade de retransmissão, o desequilíbrio entre o número de VIODs por IOD faz com que o desempenho do sistema seja determinado pelo IOD mais lento. Como nem sempre o número de IODs é um divisor do número de VIODs, pode ocorrer que um IOD acabe sendo responsável por um número maior de VIODs que os demais. Um exemplo é o caso onde um IOD possui 2 VIODs enquanto que os demais possuem apenas 1. Nesta situação, o IOD que faz parte de 2 VIODs recebe o dobro de requisições de escrita que os demais, fazendo com que o seu desempenho seja o limitador de todo o sistema. As consequências do desequilíbrio é visível nos saltos de desempenho com 3 e 9 IODs. Estes são casos onde todos os IODs possuem o mesmo número de VIODs. Já nos demais casos, o desempenho permanece próximo do constante, pois existem alguns IODs mais sobrecarregados que os outros, limitando o desempenho geral do sistema.

Na Figura 5.6, são apresentados os resultados da leitura de dados com diferentes números de IODs. Como neste caso os dados são transferidos diretamente entre servidores de dados e clientes, é possível obter resultados que se aproximam da curva teórica. Enquanto que a diferença de carga entre os IODs é pequena, o dNFSp se mantém bastante próximo dos resultados teóricos. Este é o caso do teste com 2 IODs, onde um deles possui 4 VIODs enquanto que o outro possui 5. Nesta configuração foi possível atingir resultados acima de 90% do valor teórico. Já com 4, 6, 7 e 8 servidores de dados a diferença de carga entre os servidores é maior, resultando em um desempenho quase que constante, até que todos os IODs atinjam um número igual de VIODs. Nos casos onde o número de IODs é divisor do número de VIODs, como 3 e 9, a curva obtida chega a tocar a curva teórica, pois neste caso existe uma distribuição uniforme entre os servidores de dados.

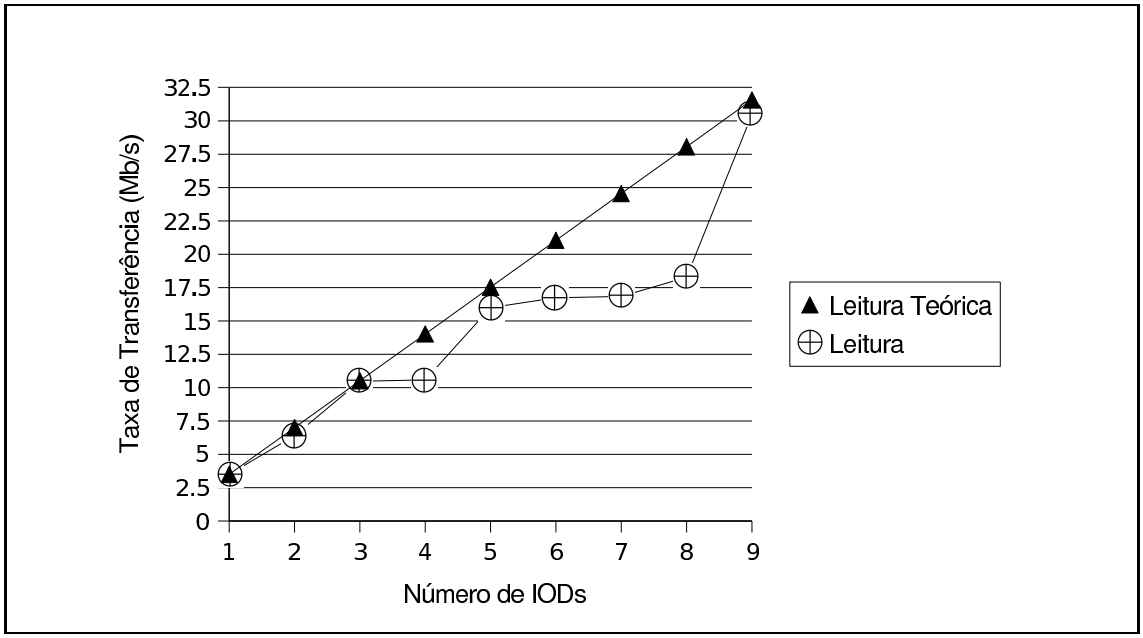

Figura 5.6: Resultado da leitura de 1Gb de dados variando-se o número de IODs associados ao sistema

5.4 Inserção de Servidores de Dados Iniciada pelo Usuário

Para avaliar o mecanismo de inserção de servidores de dados iniciada pelo usuário, foi desenvolvido um conjunto de testes englobando diversas situações que fazem parte deste cenário. O primeiro caso tem como objetivo avaliar o ganho de desempenho ao executarmos uma aplicação com servidores de dados exclusivos em uma situação onde existem outras aplicações executando no *cluster*. No segundo caso, a analise é voltada à descrição do impacto das diferentes etapas do ciclo de vida da aplicação nas demais aplicações que estão executando no *cluster* no mesmo instante. Neste caso, o objetivo é observar o quanto o desempenho da aplicação é afetado pelas aplicações executando no *cluster* ao mesmo tempo que ela.

5.4.1 Desempenho no Uso de Nós Temporários

Neste teste, foi feito uma instalação do dNFSp contendo 27 clientes, 9 metas-servidores e 9 IODs. Os 27 clientes foram divididos em 3 grupos, representando aplicações iguais, compostas de 9 clientes cada uma. Sobre esta configuração, foram executados testes de leitura e escrita seqüencial de dados, onde cada cliente escreveu e leu 1GB de dados. Em seguida, foram inseridos servidores de dados temporários a uma das aplicações, em um número igual ao de servidores de dados permanentes. Para completar o teste, a mesma quantidade de servidores de dados do caso anterior, somando-se permanentes e temporários, foi distribuída na forma de servidores de dados permanentes. As três configurações testadas são ilustradas na figura 5.7. O valor exibido no gráfico é a média dos tempos de cada no de processamento pertencente a cada aplicação.

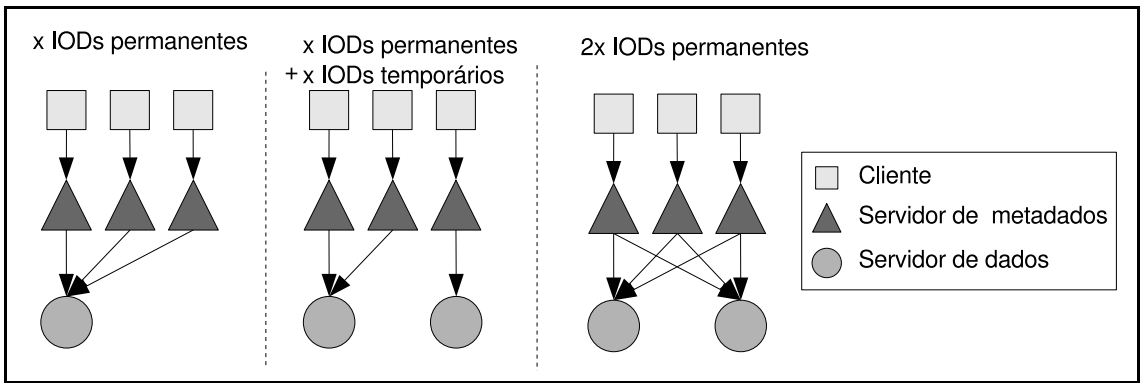

Figura 5.7: Descrição do cenário utilizado para avaliar o desempenho da inserção dinâmica de servidores temporários de dados iniciada pelo usuário

Os resultados do teste de escrita podem ser observados na Figura 5.8. Como esperado, a execução utilizando servidores de dados exclusivos trouxe os melhores resultados para a aplicação que possui servidores temporários. As demais aplicações tiveram resultados iguais entre si, visto que compartilham o mesmo recurso. No caso onde existem 18 servidores permanentes, houve um ganho de desempenho das aplicações sem IODs temporários, mas para a aplicação 3 este não foi o melhor resultado entre as três configurações testadas. Isso mostra que o uso de servidores de dados exclusivos pode oferecer vantagens de desempenho que não seriam obtidas se os recursos fossem compartilhados com todas as

aplicações. O ganho da aplicação 3 utilizando servidores temporários ao ser comparado com a primeira é de 45%.

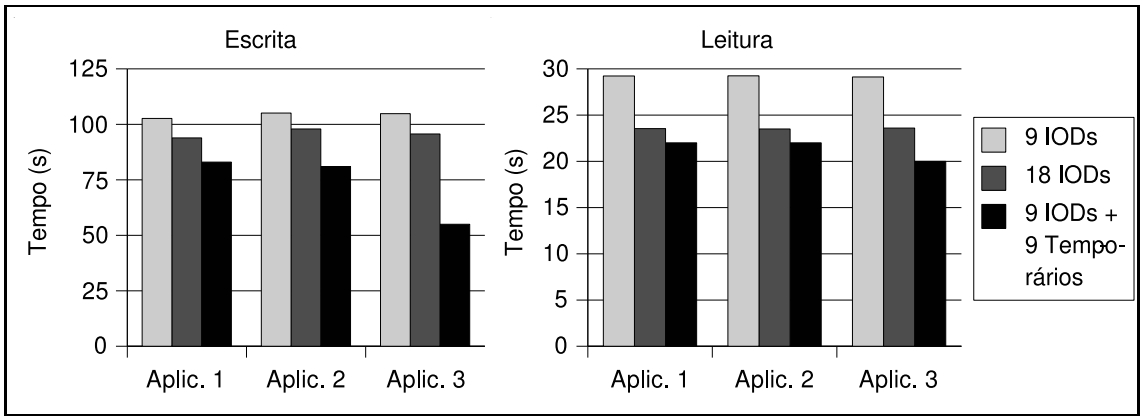

Figura 5.8: Escrita e leitura de dados utilizando servidores temporários de dados

Já no gráfico à direita da Figura 5.8, temos os resultados da leitura de dados. Neste caso não existe a participação do servidor de metadados na transferência de dados, visto que os dados são transferidos diretamente entre o servidor de dados e o cliente. Como o tempo de transferência é menor, o impacto da existência de servidores de dados temporários é menos visível, mas mesmo assim foi possível obter um ganho de desempenho de 31% utilizando servidores temporários.

5.4.2 Perfil da Taxa de Transferência Disponível

Este teste tem como objetivo simular o comportamento da taxa de transferência de dados das aplicações durante a utilização do mecanismo de servidores de dados temporários. O cenário de testes é o mesmo da seção anterior. No entanto, a transferência de dados é feita continuamente, obtendo-se a vazão de dados em determinados instantes da execução. A Figura 5.9 mostra os resultados obtidos neste teste.

Inicialmente existem apenas 9 servidores de dados, que são compartilhados pelas três aplicações (t1). Portanto, a taxa de transferência disponível a cada uma é igual. Após 100 segundos de teste (t1), a Aplicação 1 é re-executada com a inserção de 9 servidores de dados temporários. Em conseqüência, disso ela tem sua taxa de transferência quase dobrada, passando de 9,75 Mb/s para 19Mb/s, pois estes 9 IODs temporários são acessados por ela com exclusividade. As outras aplicações também tiveram um leve ganho de desempenho, pois a Aplicação 1, ao inserir servidores temporários, deixa de utilizar os servidores permanentes. Este evento fez com que as aplicações 2 e 3 tivessem um aumento de 20% na taxa de transferência.

Nos 100 segundos subseqüentes, as aplicações continuam a transferir dados. Até que no instante t2 a Aplicação 1 finaliza sua execução. Neste momento, os dados armazenados nos servidores temporários são transferidos para os permanentes. Como a operação de replicação não envolve os servidores de metadados, tanto a Aplicação 2 quanto a Aplicação 3 passam a ter uma maior taxa de transferência, ganhando cerca de 15%. Após 100 segundos, a transferência entre servidores temporários e permanentes foi interrompida. Com isso, no instante t3, as aplicações 3 e 2 passam a obter taxas de transferência ainda maiores, com um ga-

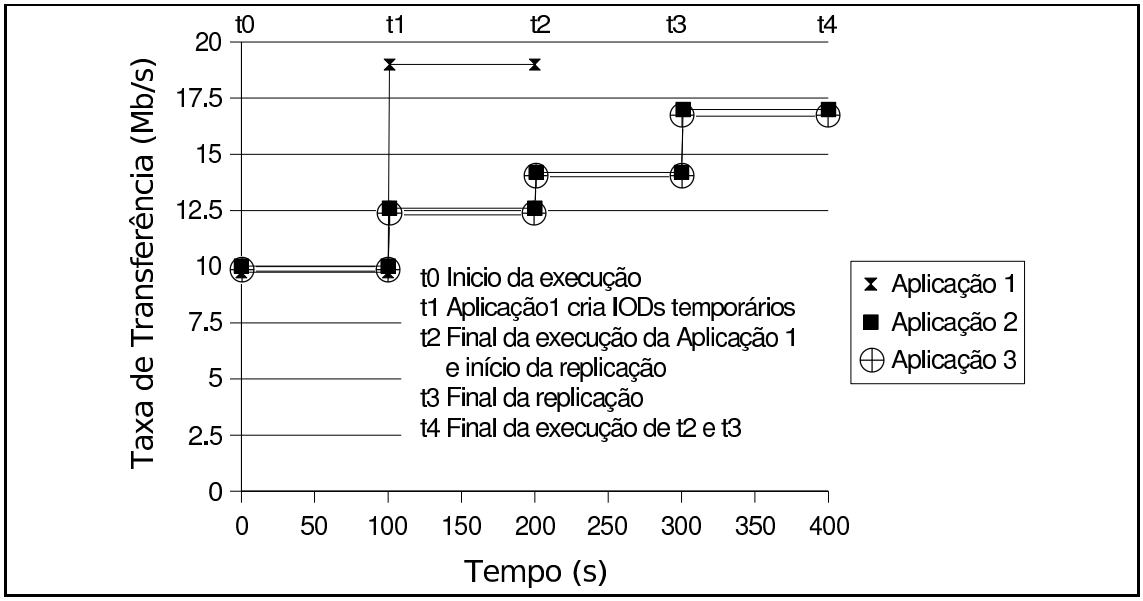

Figura 5.9: Impacto do mecanismo de IODs temporários nas demais aplicações executando no *cluster*

nho de aproximadamente 20%. Isso porque os servidores de dados permanentes não estão mais envolvidos na replicação e, portanto, podem ser dedicados plenamente às operações destas duas aplicações, até a finalização das Aplicações 2 e 3 em t4.

Estes resultados mostram que, ao inserirmos servidores temporários, foi possível obter um ganho de desempenho significativo. A taxa de transferência foi quase dobrada no período em que existiam servidores temporários de dados. A influência dos servidores de metadados continua a existir, pois o teste realizado é um teste de escrita. Mesmo assim, foi possível obter uma melhoria no desempenho com a inserção de um grupo de servidores de dados temporários.

5.4.3 Testes de Desempenho Utilizando BTIO

Os testes realizados anteriormente utilizam clientes que efetuam apenas operações de entrada e saída. A exemplo do que foi feito com o mecanismo de sincronização de servidores de metadados, nesta seção é avaliado o comportamento do mecanismo de inserção de servidores temporários utilizando uma aplicação científica. Novamente, o analisador de desempenho escolhido foi o BTIO, como descrito na Seção 5.2.3.

Durante estes testes, tentou-se simular a utilização de um *cluster*, onde três usuários executam simultaneamente instâncias do BTIO. A Figura 5.10 apresenta os resultados obtidos. No primeiro caso, cada usuário possui 4 clientes. Já o sistema de arquivos é composto por 4 servidores de metadados e 4 servidores de dados. A aplicação 1 possui 4 servidores de dados temporários acessados com exclusividade por ela. Este padrão de configuração, contendo mesmo número de clientes por usuário, mesmo número de servidores de dados e metadados, é repetido para os dois casos restantes.

O BTIO é caracterizado pela escrita de dados após cada etapa de processamento. Assim, os servidores de dados não são os únicos delimitadores de desempenho do sistema de arquivos. Isso porque os servidores de metadados são

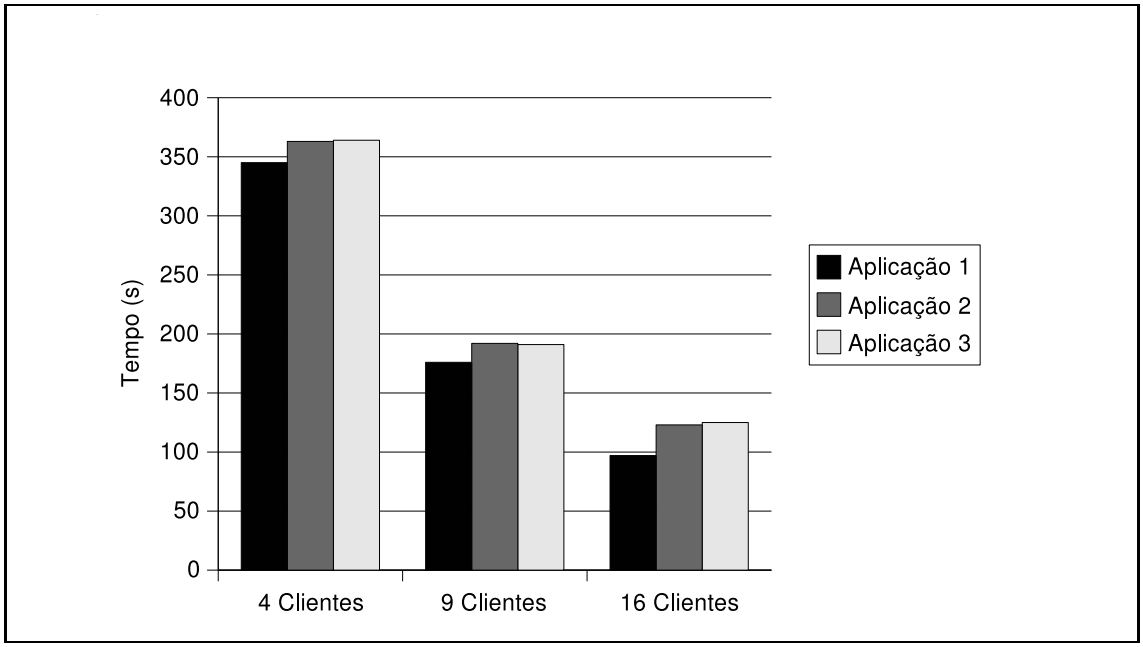

Figura 5.10: Reultado do teste utilizando o BTIO mesclando aplicações com e sem IODs temporários

compartilhados por todos os clientes e são responsáveis por repassar os dados escritos aos servidores de dados. Mesmo no caso onde existem servidores de dados temporários, os servidores de metadados continuam sendo compartilhados por todos os clientes. O fato do BTIO mesclar operações de processamento com operações de E/S permite que um cliente escreva no sistema de arquivos enquanto outro está na fase de processamento. Isso diminui a interferência de um usuário sobre os demais, pois nem todos efetuam operações de E/S ao mesmo tempo.

Mesmo que algumas características do BTIO dificultem a obtenção de um melhor desempenho com o uso de servidores temporários, em todos os casos testados a aplicação com servidores temporários executou em um tempo menor. A porcentagem de ganho desta aplicação aumenta com o número de nós envolvidos no teste. No caso de 4 clientes, o ganho foi de apenas 5%. Já com 9 clientes, foi possível obter uma melhoria de 10%. No terceiro caso, com 16 clientes, a Aplicação 1 foi 20% mais rápida que as demais.

Outro teste foi efetuado para avaliar uma configuração onde o sistema de arquivos possui um número reduzido de servidores permanentes. Para isso, foi configurada uma instalação do dNFSp contendo apenas 2 nós. Em ambos os nós foram instalados tanto servidores de dados quanto de metadados. Sobre este sistema de arquivos, executou-se o BTIO com 16 nós de processamento. A média de tempo de execução obtido foi de 172 segundos. Em um segundo momento, foram inseridos 4 IODs temporários e executado novamente o BTIO com 16 clientes. Neste caso a média do tempo de execução foi de 114 segundos. Assim, o uso de servidores temporários permitiu um ganho de 33% no tempo de execução ao compararmos com a execução sem servidores temporários. Além disso, o uso de servidores temporários permite que alguns nós sejam destinados ao armazenamento apenas quando necessário. Neste caso, um sistema de arquivos contendo apenas 2 nós pode ser expandido para suportar 4 servidores de dados durante o tempo de execução do BTIO.

5.5 Falha de Servidores de Dados

Na existência de réplicas de dados o dNFSp2 pode suportar a falha de servidores de dados, sem que estas falhas afetem a disponibilidade do sistema. Com perda de um servidor de dados decorrente de uma falha, o sistema pode reconfigurar-se para manter a restrição do número mínimo de réplicas de dados. Passada a etapa de replicação, ele está apto a suportar a perda de mais um servidor de dados.

Para avaliar este cenário, são apresentados, nesta seção, alguns testes do desempenho do dNFSp2 utilizando replicação de dados. Primeiramente é analisado o impacto dos mecanismos de replicação no desempenho do sistema, comparando o seu desempenho com uma instalação sem réplicas. Num segundo teste, são simuladas falhas sucessivas de servidores de dados e para cada falha é mensurado o desempenho do sistema após a reconfiguração.

5.5.1 Desempenho da Replicação

Estes testes têm como objetivo avaliar o desempenho do dNFSp2 utilizando replicação de dados. Para isso, os resultados foram comparados com os obtidos pelo sistema de arquivos Lustre. Este sistema permite que nós funcionem em pares espelhados, ativo e *failover*. Porém, ele não oferece mecanismos de replicação. Portanto, é preciso existir um armazenamento que seja comum entre os dois nós que fazem parte do espelhamento. Para obter isso, é possível utilizar discos compartilhados ou então soluções que fazem replicação em *software*. Um exemplo de sistema que oferece replicação em software através da rede é o Dispositivo de Blocos Replicados Distribuídos (*Distributed Replicated Block Device*, DRBD) (REISNER, 2000).

O DRBD é um módulo para o núcleo de sistema operacional Linux, que proporciona replicação de dados entre dois nós de um *cluster*. As leituras são tratadas pelo servidor principal, enquanto que as escritas são espelhadas no outro nó. Todas as requisições de armazenamento são interceptadas por este sistema antes de serem repassadas aos dispositivos de armazenamento físico. Assim, no caso de escritas, além de serem transferidos aos dispositivos de armazenamento, os dados são transmitidos ao nó espelho. Cada um dos dois nós tem uma função específica, um é designado como principal, enquanto que o outro é o secundário. Por razões de consistência, a escrita de dados é apenas oferecida ao nó principal.

Outra funcionalidade do DRBD é a possibilidade de escolher entre diferentes mecanismos de espelhamento entre os nós. Existem três protocolos, que influenciam tanto no desempenho quanto na disponibilidade dos dados. Eles diferem no momento em que a confirmação da escrita é feita ao sistema. O **Protocolo A** confirma a escrita assim que os dados são escritos no disco local. Esta opção tem a vantagem de ter um alto desempenho, no entanto não existe garantia de que os dados foram escritos ou enviados. O **Protocolo B** confirma a escrita assim que o nó secundário recebe a resposta, e os dados são escritos no disco local. O *Protocolo C* é completamente síncrono, a confirmação de escrita apenas é feita quando os dados são escritos em ambos os discos.

Os testes apresentados nas Figuras 5.11 e 5.12 foram efetuados utilizando 9 servidores de dados sendo que cada um possui uma réplica, totalizando 18 servidores de dados. No caso do dNFSp2 foi utilizado o repasse de mensagens, e

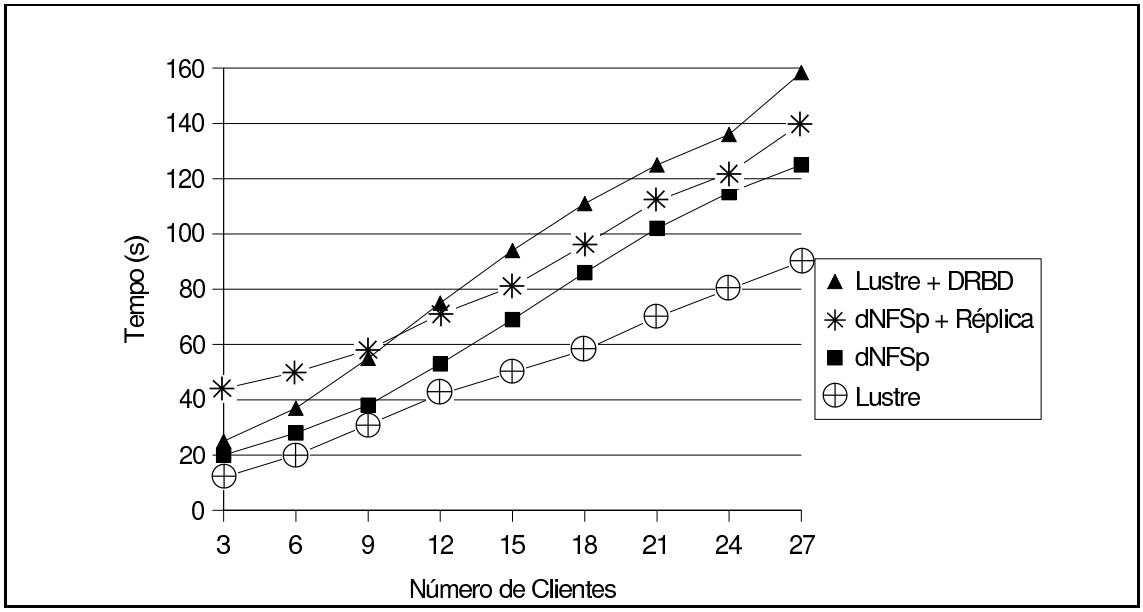

Figura 5.11: Desempenho de escrita de dados comparando configurações com e sem replicação

no Lustre foi utilizado o DRBD com Protocolo C para manter réplicas entre os servidores. Os testes iniciaram com 3 clientes, adicionando-se mais 3 clientes a cada passo até atingir um total de 27. Cada cliente escreveu e em seguida leu 1Gb de dados. Também foram realizados testes sem a existência de réplicas, para que com isso fosse possível avaliar a sobrecarga inserida pela replicação. Nestes casos foram utilizados apenas 9 servidores de dados.

Os resultados da escrita de dados são ilustrados na Figura 5.11. Tanto com o Lustre quanto com o dNFSp2, a escrita em servidores replicados teve desempenho pior que quando feita sem o uso de réplicas. Isso ocorre porque, em ambos os sistemas de arquivos, é preciso transmitir os dados do servidor que recebeu a requisição para o outro servidor de dados que faz parte do mesmo grupo de réplicas. Ao compararmos os testes com réplica do dNFSp2 e do Lustre, percebe-se que, para um número maior de clientes, o dNFSp2 possui um desempenho melhor, enquanto que o Lustre foi melhor quando utilizada uma quantidade pequena de clientes.

O melhor desempenho do mecanismo de replicação do dNFSp2 é resultante do menor número de mensagens necessárias para o espelhamento de dados. No DRBD, o servidor que recebe a escrita transmite os dados para o outro servidor, em seguida, aguarda a confirmação e só então confirma para o cliente. Já no dNFSp2, o servidor que recebe os dados o transmite para o outro servidor, e este último envia a confirmação diretamente para o cliente, resultando em uma mensagem a menos comparado ao DRBD. Este enfileiramento de servidores na escrita de dados no dNFSp2, formado pelo metasservidor e os servidores de dados replicados, faz com que o comportamento da escrita se assemelhe a um *pipeline* de processamento. Enquanto existem requisições suficientes, o sistema possui um bom desempenho. Quando o número de requisições é pequeno, o tempo levado para percorrer todos os servidores acaba tornando-se um limitador no desempenho do sistema. Em consequência disso, tem-se um mecanismo que possui uma vazão de dados maior que o DRBD porém com uma latência mínima mais

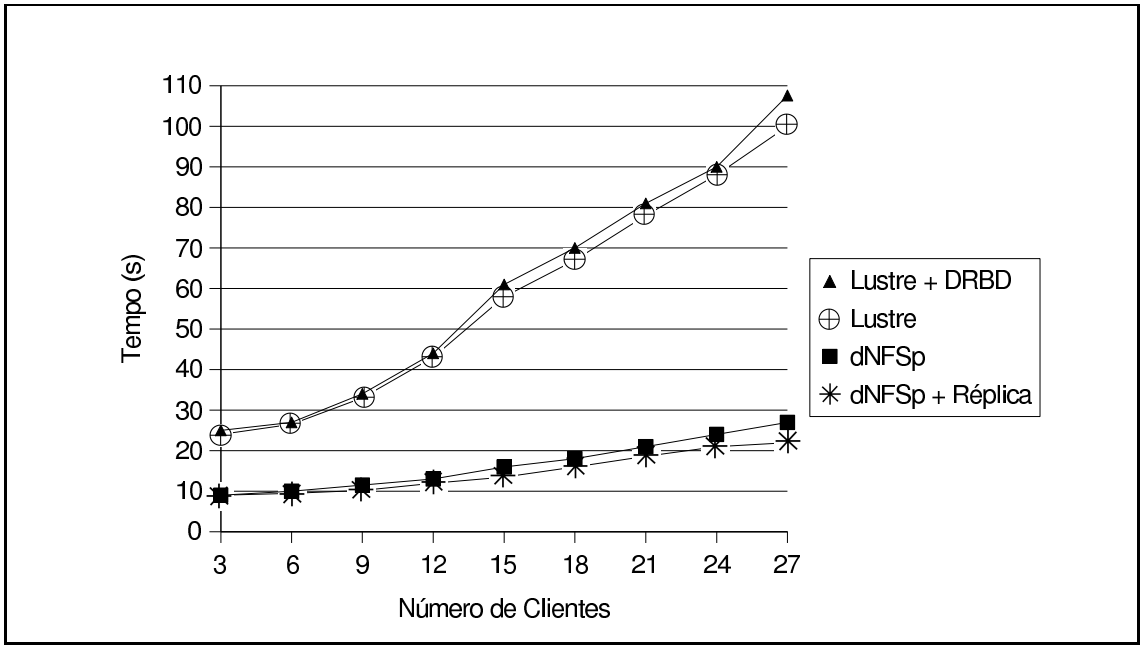

Figura 5.12: Desempenho de leitura de dados comparando configurações com e sem replicação

elevada.

Nos casos onde não há replicação, o Lustre sempre manteve um desempenho melhor que o dNFSp2. Novamente, o principal motivo é a necessidade do repasse de dados entre o servidor de metadados e o servidor de dados no dNFSp2. O Lustre sem replicação tem um tempo de transferência próximo da metade em relação à configuração com replicação. Já o dNFSp2 possui uma diferença menor entre a versão com réplicas e a sem réplica principalmente quando o número de clientes é maior. Isso ocorre novamente porque as requisições passam a ser tratadas de forma encadeada fazendo com que o período de escrita de um novo dado se sobreponha com a replicação do dado anterior.

Na Figura 5.12, são apresentados os testes de leitura. Estes testes foram feitos na seqüência dos testes apresentados anteriormente. Portanto, os dados lidos nesta etapa são os que foram escritos durante os testes de escrita. Os dois sistemas testados demonstraram comportamentos diferentes na leitura com e sem réplicas. Como o Lustre é baseado em réplica ativa e *failover*, a leitura é feita apenas de um dos servidores, enquanto que o outro não participa no tratamento de requisições de leitura. Assim, o desempenho utilizando réplicas teve resultados bastante semelhantes aos obtidos sem réplica, com uma pequena desvantagem causada pela camada responsável pelo tratamento da replicação. Por outro lado, no dNFSp2 as leituras são distribuídas entre as réplicas. Em consequência disso, a leitura de dados utilizando réplicas obteve melhores resultados do que no caso sem réplicas. Também é possível observar que o dNFSp2 possui um desempenho geral de leitura melhor que o Lustre, pois neste caso ele não possui a mesma restrição que tinha nas escritas, onde os dados devem ser repassados entre servidores de metadados e servidores de dados. Na leitura os dados são transferidos diretamente dos servidores de dados para os clientes.

Além deste teste, foram avaliados casos onde um sistema de replicação contendo poucos nós é acessado por um grande número de clientes. Isso permite

avaliar o desempenho em situações de grande sobrecarga. Cada um dos sistemas foi instalado com 2 servidores de dados e cada um dos 27 clientes escreveu e leu 256Mb de dados. Os tempos em segundos são apresentados na Tabela 5.1.

Tabela 5.1: Tempos de leitura e escrita de dados com e sem replicação utilizando uma instalação de sistema de arquivos formada por dois nós

	Escrita	**Leitura**
dNFSp + replicação	167	27
Lustre + DRBD	172	66
dNFSp	136	40
Lustre	64	65

O perfil dos resultados obtidos na escrita condiz com os apresentados na Figura 5.11. Os sistemas com réplicas obtiveram um tempo de execução mais elevado, causado pela necessidade de transmitir os dados para os dois servidores de dados. O dNFSp2 sem réplicas teve um desempenho pior que o Lustre, devido à necessidade de repasse de dados entre o servidor de metadados e o servidor de dados.

A diferença principal deste teste com maior sobrecarga está na leitura de dados. Este caso salientou a vantagem de utilizar duas cópias ativas no lugar de uma copia ativa e outra *failover*, que é o caso do Lustre. A leitura de dados sem réplicas obteve um resultado 30% inferior que o com réplicas. Isso porque na leitura as requisições são distribuídas entre os IODs que fazem parte de um mesmo VIOD. Já no Lustre isso não acontece, pois todas as requisições são tratadas apenas pelo servidor de dados ativo.

5.5.2 Desempenho após Reconfiguração

A opção miniods oferece a possibilidade de especificar o número mínimo de réplicas de dados existentes no sistema. Assim, esta condição deve ser verificada quando um nó é retirado do sistema, seja por motivo de falha ou por remoção explícita do administrador. Uma vez detectado que o sistema não possui o número mínimo de réplicas exigidas, inicia-se o processo de reconfiguração. Esta reconfiguração pode levar à sobrecarga de alguns nós.

Quando a opção de número mínimo de IODs por VIOD é definida, o sistema de arquivos passa a priorizar a disponibilidade antes do desempenho. Portanto, em alguns casos a reconfiguração após a perda pode não ser a mais eficiente, mas garante que após um determinado período o sistema possa perder outros servidores de dados. A Figura 5.13 ilustra os resultados dos testes de desempenho do sistema de arquivos com o objetivo de determinar o comportamento do sistema após a perda de servidores de dados.

Para isso, os testes partiram de uma configuração inicial onde o número de IODs por VIOD foi ajustado para 2. O sistema de arquivos foi instalado com 9 servidores de metadados e existem 27 clientes lendo e escrevendo 1Gb de dados. No início o sistema possui 18 IODs e cada cliente escreve e lê os seus dados. Após, um dos servidores de dados é interrompido, simulando uma falha. Ao notificar esta perda o sistema se reconfigura. Após a reconfiguração o cliente repete a escrita e leitura dos dados. Estes passos são efetuados sucessivamente até que o

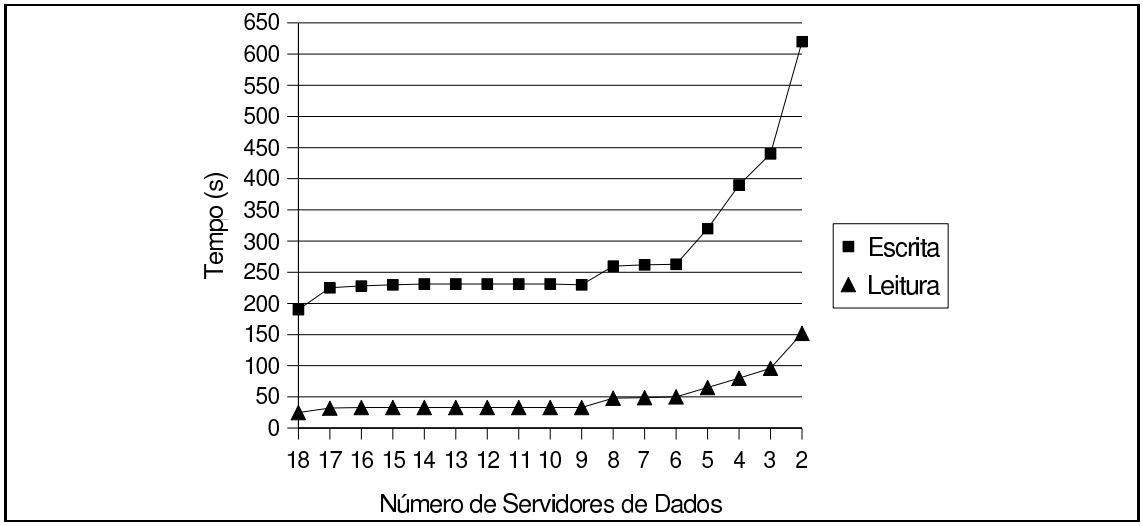

Figura 5.13: Impacto da reconfiguração após a falha de servidores de dados no desempenho do sistema de arquivos

número de servidores de dados seja insuficiente para satisfazer a restrição de 2 IODs por VIOD. A cada etapa é registrado o tempo médio de leitura e de escrita.

O impacto da reconfiguração no desempenho tem o mesmo perfil tanto nas operações de leitura quanto nas de escrita. Porém, a degradação é mais significativa nas escritas. De forma semelhante ao que ocorre na inserção de servidores de dados, o desempenho resultante esta diretamente ligado à distribuição dos VIODs entre os IODs. Também observou-se que as mensagens enviadas pelo mecanismo de detecção de falhas através de *ping* não modificaram o desempenho do sistema de arquivos.

Após a perda do primeiro IOD, um dos IODs restantes passa a participar de 2 VIODs, para assim compensar esta perda. Este desequilíbrio acaba limitando o desempenho de todo o sistema. A perda de mais alguns nós na seqüência, não degrada o desempenho, pois a eficiência do sistema está limitada ao desempenho de um IOD participando de dois VIODs. Quando a perda de um IOD exige que um deles passe a fazer parte de 3 VIODs, existe uma nova queda de desempenho. Este é o caso da passagem de 9 para 8 IODS. Esta situação se repete para as perdas seguintes. Sempre com um relacionamento direto entre desempenho e a distribuição de VIODs entre os IODs em funcionamento.

5.6 Resumo do capítulo

A avaliação de desempenho e validação da implementação foram feita utilizando os nós instalados no *i-cluster2* (I-CLUSTER 2, 2006). Os primeiros testes realizados visaram avaliar os resultados da etapa de adaptação do dNFSp1. Foi testado o mecanismo de sincronização de metadados, comparando-o com outras implementações. Inicialmente foram feitos testes com operações básicas sobre o sistema de arquivos. No caso, foram realizadas leituras e escritas seqüenciais. O tamanho dos arquivos foi variado para enfatizar o impacto que o número de operações de metadados provoca no desempenho final. Os resultados mostram que o dNFSp2 e o NFS sustentaram uma taxa de transferência mais próxima de uma constante, independente do tamanho dos arquivos. Já o dNFSp1 e o PVFS2 ti-

veram uma degradação de desempenho significativo com a redução do tamanho dos arquivos.

Partindo para um teste mais complexo, foi utilizado o *benchmark* DAB, que é baseado no *Andrew Benchmark*. Ele é dividido em 5 fases, que incluem a criação de arquivos, escrita de dados, leitura de atributos, leitura de conteúdo e compilação de códigos fonte. Na grande parte dos casos, o dNFSp2 obteve um desempenho melhor que os demais sistemas. A única etapa que o dNFSp1 foi superior ao dNFSp2 foi a fase de leitura de dados. Isso porque o mecanismo de consistência do dNFSp1 é mais relaxado que o do dNFSp2 o que diminui a quantidade de comunicação entre os servidores de metadados. Em todas as demais fases, as operações de metadados do dNFSp2 foram mais rápidas que o dNFSp1 e o PVFS2.

O terceiro teste utilizou um *benchmark* que simula uma aplicação científica. Neste caso, operações de escritas foram combinadas com etapas de processamento. Como cada cliente escreve em um arquivo separado, o dNFSp1 não tem a desvantagem da inconsistência de dados. Mesmo nesta situação, onde o dNFSp1 tem seu melhor caso, o dNFSp2 conseguiu obter desempenho semelhante. Nos casos onde o número de nós de processamento é maior, e por conseqüência a quantidade de dados escrito por cada um é menor, o dNFSp2 consegue superar o dNFSp1. Isso mostra que o novo protocolo consegue conciliar maior sincronismo com alto desempenho.

Terminada a etapa de avaliação da adaptação do sistema, foram feitos testes para analisar o comportamento da implementação do dinamismo de servidores de dados em seus diferentes cenários. O primeiro deles é a inserção iniciada pelo administrador para expansão do sistema. Neste caso, o teste foi feito com a escrita de arquivos seguida de leitura. O sistema foi configurado inicialmente com apenas um servidor de dados (IOD), e a cada etapa foi inserido um novo IOD e medido o tempo de transferência dos dados. Tanto a leitura quanto a escrita apresentaram perfis de desempenho semelhantes. Isso porque o desempenho está diretamente ligado ao equilíbrio da distribuição dos servidores de dados virtuais (VIODs) entre os IODs. Os melhores resultados são obtidos quando o número de VIODs é o mesmo para todos os IODs. Nestes casos, as operações de leitura atingem a curva teórica de desempenho. Nos outros casos, o desempenho está limitado pela capacidade dos servidores de dados mais sobrecarregados.

Outra situação avaliada é a execução de aplicações com servidores de dados temporários. No primeiro teste, três aplicações foram executadas. Todos os clientes efetuaram operações de leitura e de escrita, com a diferença de que uma das aplicações foi configurada com IODs temporários. Em todos os casos, a aplicação com IODs temporários obteve resultados melhores. Mesmo na escrita, onde a influência dos servidores de metadados é significativa, foi possível obter melhoria de desempenho.

Em outro teste, foi avaliado o impacto da fase de transferência dos dados dos servidores temporários para os permanentes. Com a finalização da aplicação com servidores temporários, as outras tiveram um aumento na taxa de transferência de dados disponível. Isso porque, ao finalizar, ela deixa de enviar requisições aos servidores de metadados. Terminada a transferência dos dados para os servidores permanentes, as aplicações sem servidores temporários têm um ganho de desempenho. Este ganho ocorre porque os servidores de dados permanentes dei-

xam de receber os dados enviados pelos servidores temporários. O impacto desta etapa de transferência depende da quantidade dos dados criados pela aplicação.

O último teste com IODs temporários foi feito utilizando o BTIO. Os clientes foram distribuídos em 3 Aplicações, cada uma rodando uma instância do BTIO. Em todos os casos, a aplicação com IODs temporários teve um desempenho melhor. Porém, este foi pouco significativo devido a combinação entre processamento e E/S realizada pelo BTIO, o que em alguns casos mascara o tempo gasto pelas operações de escrita. O ganho de desempenho esteve relacionado ao número de nós de processamentos. A melhoria no desempenho utilizando servidores de dados temporários variou entre 5% para o caso com 4 clientes e 20% para o caso com 16 clientes.

Também foi testado um caso onde o sistema de arquivos contém apenas 2 nós. Nos clientes foi executado o BTIO, e em seguida foram adicionados 4 IODs temporários ao sistema de arquivos. Neste caso, foi possível obter um ganho de até 33% de desempenho comprando-se o uso de servidores temporários com a configuração inicial.

O último cenário do dinamismo testado é a perda de nós por falha. Neste caso, foi avaliado o desempenho de dois sistemas de arquivos: o dNFSp2 e o Lustre. Os resultados mostraram que na escrita de dados tanto o dNFSp2 com réplica quanto o Lustre obtiveram resultados piores que as versões sem réplica. Isso porque é preciso transmitir os dados para os dois servidores de dados para efetuar a replicação. Na leitura, o dNFSp2 com réplicas obteve resultados um pouco melhor que o caso sem réplicas. No dNFSp2, ambos os servidores da replicação podem tratar as operações de leituras. Já no Lustre, apenas o servidor de dados ativo pode tratar as leituras, fazendo com que a versão com réplicas não tire vantagem da existência de duas cópias dos dados.

Por fim, foi avaliado o perfil do desempenho com a perda sucessiva de nós. A cada etapa do teste foi excluído um servidor de dados, e testado o desempenho de escrita e leitura de dados. O desempenho das operações após a perda de nós está limitada pelo desempenho do servidor de dados mais sobrecarregado. Existem momentos em que a perda de nós não resulta em perda direta de desempenho. Isso ocorre quando já existem outros servidores de dados sobrecarregados restringindo o desempenho do sistema.

6 CONCLUSÃO

Com a popularização do uso de *clusters* de computadores, é cada vez mais comum a implantação de sistemas contendo centenas ou milhares de nós. Para estes ambientes, o desempenho do sistema de arquivos torna-se um elemento crucial. Porém, a diversidade de aplicações executando nestes ambientes torna o dimensionamento do sistema de arquivos uma tarefa complexa. A maioria dos sistemas de arquivos para *cluster*, ao utilizarem uma configuração estática, acaba limitando as ações do administrador. Isso força o administrador a ter que prever as necessidades de armazenamento dos usuários no momento da instalação do sistema de arquivos, pois na maioria dos casos o seu redimensionamento exige uma reinstalação.

Para que o redimensionamento do sistema de arquivos possa ser facilitado, este trabalho apresentou um modelo de dinamismo de servidores de dados para o sistema de arquivos dNFSp. O dNFSp oferece uma implementação do protocolo NFS, distribuindo suas funcionalidades entre os nós de um *cluster*. Esta distribuição é feita classificando os nós em dois tipos: servidores de metadados, responsáveis pelo armazenamento dos atributos do arquivos e servidores de dados, responsáveis por armazenar o conteúdo dos arquivos. O dinamismo tratado neste trabalho se restringe aos servidores de dados. Portanto, a inserção e remoção dinâmica de servidores de metadados não foi levada em consideração. O sistema desenvolvido neste trabalho foi chamada de dNFSp2 e os resultados obtidos no desenvolvimento deste trabalho também podem ser vistos em publicações do grupo de pesquisa (HERMANN et al., 2006; KASSICK et al., 2005; HERMANN et al., 2006).

O restante desta conclusão está dividido em duas seções. Na Seção 6.1 são apresentadas as principais contribuições resultantes deste trabalho. Em seguida, na Seção 6.2 são descritas as principais atividades que podem ser desenvolvidas para dar continuidade a este trabalho, acompanhadas de algumas sugestões de como desenvolvê-las.

6.1 Contribuições

Dentre os resultados obtidos deste trabalho, a sua principal contribuição está no desenvolvimento de um modelo de inserção de servidores temporários de dados por parte do usuário. Este tipo de tratamento não foi encontrado entre os sistemas de arquivos estudados. A sua implantação permite dar maior autonomia ao usuário no que diz respeito à capacidade e desempenho de armazenamento. Em todos os testes realizados foi possível obter um desempenho melhor com o

uso de servidores temporários de dados, chegando a atingir um desempenho 45% melhor se comparado com o obtido sem o uso deste mecanismo. Isso porque os servidores temporários de dados são acessados com exclusividade pela aplicação que os adicionou.

Além desse cenário de inserção de servidores temporários, algumas idéias implementadas em outros trabalhos foram adaptadas para o dNFSp2. Estas adaptações resultaram em outras duas contribuições: a inserção feita pelo administrador e o suporte à tolerância a falhas. A inserção feita pelo administrador possibilita adaptar o sistema de arquivos em tempo de execução, e assim fazer um melhor uso dos recursos. Assim, é possível oferecer maior desempenho e capacidade de armazenamento de acordo com o conhecimento que o administrador tem sobre a utilização do *cluster*. Nos testes realizados abordando esta questão, foi possível verificar uma eficiência muito próxima da curva teórica da taxa máxima de transferência de dados, o que mostra o bom desempenho deste mecanismo.

O cenário envolvendo tolerância a falhas é resultante da combinação do dinamismo de servidores de dados com a replicação de dados. Com isso, foi possível desenvolver um sistema que se reconfigura após a falha de um servidor de dados. Esta reconfiguração permite que, após um certo período, o sistema suporte a falha de um outro servidor de dados. O mecanismo de replicação do dNFSp mostrou-se mais eficiente que os demais na maioria dos casos, chegando a atingir um tempo de transferência três vezes menor que o obtido pelo sistema de replicação DRBD.

Também é importante salientar que na etapa inicial deste trabalho foi realizada a adaptação do algoritmo de *hashing* para o contexto do dNFSp. Este novo algoritmo implementado conseguiu conciliar maior sincronismo com a capacidade de alto desempenho. Foi possível passar de uma complexidade $O(n)$ para $O(1)$ durante a etapa de busca de metadados. Com isso, em testes com o *benchmark* DAB, obteve-se um tempo de execução que representa aproximadamente 30% do tempo levado pelo mecanismo implementado no dNFSP1.

Por fim, é importante ressaltar que para que se possa tirar o máximo de proveito dos *clusters* de computadores é fundamental o emprego de sistema de arquivos capazes de prover alto desempenho. Este trabalho contribuiu diretamente neste escopo, uma vez que possibilita que os usuários do dNFSp2 possam inserir servidores temporários de dados para melhorar o desempenho de suas aplicações. O administrador também pode, dinamicamente, alterar a configuração do sistema de arquivos a fim de melhor atender aos objetivos dos usuários. Para efetivar a implementação de tais melhorias foram necessárias adaptações na implementação original do dNFSp, além de permitir o suporte à tolerância a falhas.

6.2 Trabalhos Futuros

Os resultados deste trabalho apresentaram melhorias em diversos aspectos do sistema de arquivos. Novas funcionalidades foram implementadas enquanto que outras foram aperfeiçoadas. Porém, ainda restam alguns pontos importantes a serem desenvolvidos. No restante desta seção serão apresentados algumas atividades que podem ser realizadas para dar seqüência ao desenvolvimento deste trabalho.

Fracionamento flexível de dados: A implementação corrente do dNFSp utiliza o algoritmo *round-robin* para implementar o fracionamento de dados. Esta maneira de distribuir os dados é bastante restrita e pode não ser adequada a todos os tipos de aplicações. Por isso, uma solução mais flexível que permita modificar o algoritmo de acesso aos dados pode trazer ganho de desempenho para algumas aplicações. Outro ponto importante, é a possibilidade de distribuir o conteúdo dos arquivos em um subconjunto de servidores de dados. Esta medida possibilita tirar proveito de arquiteturas heterogêneas.

Integração com a implementação do *kernel*, e NFSv3: Com a existência de duas implementações do NFSp, está em andamento a integração do dNFSp2 com a versão *kernel* do NFSp. Outros grupos de pesquisa que fazem parte do projeto também estão portando o NFSp para o NFS versão 3. A principal vantagem do protocolo NFSv3 é a possibilidade de executar operações assíncronas.

Falha de servidores de metadados: Muitas das técnicas apresentadas neste trabalho podem ser aplicadas na implementação de tolerância a falhas em servidores de metadados. Técnicas como a de servidores atuando em pares, ativo e *failover* também podem ser utilizadas. A principal dificuldade deste caso é fazer com que a substituição dos servidores de metadados seja transparente aos clientes.

Redistribuição dos dados: Como mostrado nos resultados, o desempenho da inserção de servidores de dados está diretamente ligada à distribuição dos VIODs entre os IODS. Para que seja possível sempre obter o melhor desempenho possível independente do número de VIODs e IODS, é preciso implementar mecanismos de redistribuição de dados. A eficiência deste mecanismo é muito importante para que o período de reconfiguração não seja muito elevado.

Otimização da sincronização de dados: A implementação do mecanismo de sincronização entre servidores de metadados apresentada neste trabalho não leva em consideração o fato de que apenas uma parte dos dados necessita ser atualizada. Um protótipo baseado em versão de bloco foi desenvolvido porém não encontra-se plenamente funcional. Resta estudar maneiras de efetuar esta atualização de forma eficiente, inclusive buscando alternativas à versão de bloco.

Teste com aplicação do projeto GBRAMS: O Grupo de Processamento Paralelo e Distribuído da UFRGS faz parte do projeto GBRAMS, que pesquisa a utilização de uma arquitetura GRID para a execução de simulações de climatologia. Estas aplicações realizam operações que envolvem grandes quantidades de dados. Portanto, a utilização de um sistema de arquivos de alto desempenho, como o dNFSp, pode trazer grandes melhorias no tempo de execução destas aplicações.

Implentação do dNFSP como armazenamento de testes no GRID5000: O ambiente GRID5000 agrega vários *clusters* distribuídos pela França, como uma plataforma de testes para desenvolvimento de soluções de alto desempenho. Está em andamento a instalação do NFSp em um cluster dedicado em Grenoble, que será oferecido como sistema de armazenamento alternativo aos usuários que desejarem testar suas aplicações utilizando um sistema de arquivos distribuído.

REFERÊNCIAS

ANDERSON, T. et al. Serverless network file systems. In: SYMPOSIUM ON OPERATING SYSTEM PRINCIPLES. ACM, 15., 1995, Copper Mountain Resort, Colorado. **Proceedings...** [S.l.: s.n.], 1995. p.109–126.

ÁVILA, R. B. **Uma Proposta de Distribuição do Servidor de Arquivos em Clusters**. 2005. Tese (Doutorado em Ciência da Computação) — Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

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. **Proceedings...** Washington: IEEE, 2004. p.100–105.

BAILEY, D. H. The NAS Parallel Benchmarks. **International Journal of Supercomputer Applications**, [S.l.], v.5, n.3, p.63–73, 1991.

BAKER, M.; BUYYA, R. Cluster Computing at a Glance. In: BUYYA, R. (Ed.). **High Performance Cluster Computing**. Upper Saddle River, NJ: Prentice Hall PTR, 1999. v.1, p.3–47.

BOLOSKY, W. J. et al. Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs. In: ACM SIGMETRICS INTERNATIONAL CONFERENCE ON MEASUREMENT AND MODELING OF COMPUTER SYSTEMS, 2000, Santa Clara, California, United States. **Proceedings...** New York: ACM Press, 2000. p.34–43.

BRAAM, P. J. The Coda Distributed File System. **Linux J.**, Seattle, WA, USA, v.1998, n.50es, p.6, 1998.

BRANDT, S. A. et al. Efficient Metadata Management in Large Distributed Storage Systems. In: IEEE/NASA GODDARD CONFERENCE ON MASS STORAGE SYSTEMS AND TECHNOLOGIES, MSS, 20., 2003, San Diego, CA, USA. **Proceedings...** Washington: IEEE Computer Society, 2003. p.290.

CALLAGHAN, B.; PAWLOWSKI, B.; STAUBACH, P. **NFS Version 3 Protocol Specification**: RFC 1813. [S.l.]: IETF, 1995.

CARNS, P. H. et al. PVFS: A parallel file system for linux clusters. In: ANNUAL LINUX SHOWCASE AND CONFERENCE, 4., 2000, Atlanta, GA. **Proceedings...** [S.l.]: USENIX Association, 2000. p.317–327.

CLUSTER FILE SYSTEMS, INC. **Lustre**: a scalable, high-performance file system. Disponível em <http://www.lustre.org/docs/whitepaper.pdf>. Acesso em: ago. 2006.

CORBETT, P. F.; BAYLOR, S. J.; FEITELSON, D. G. Overview of the Vesta parallel file system. **SIGARCH Comput. Archit. News**, New York, NY, USA, v.21, n.5, p.7–14, 1993.

CORBETT, P. F.; FEITELSON, D. G. The Vesta parallel file system. **ACM Trans. Comput. Syst.**, New York, NY, USA, v.14, n.3, p.225–264, 1996.

DEERING, S. **Host extensions for IP multicasting**: RFC 1112. [S.l.]: IETF, 1989.

DUARTE JUNIOR, E. P.; NANYA, T. A Hierarchical Adaptive Distributed System-Level Diagnosis Algorithm. **IEEE Trans. Comput.**, Washington, DC, USA, v.47, n.1, p.34–45, 1998.

EISLER, M. **XDR:** External Data Representation Standard. [S.l.]: IETF, 2006.

GALLMEISTER, B. O. **POSIX.4**: programming for the real world. Sebastopol, CA, USA: O'Reilly & Associates, 1995.

GARCIA-CARBALLEIRA, F. et al. The design of the expand parallel file system. **International Journal of High Performance Computing Applications**, [S.l.], v.17, n.1, p.21–37, 2003.

HAHNE, E. L. Round-Robin Scheduling for Max-Min Fairness in Data Networks. **IEEE Journal of Selected Areas in Communications**, [S.l.], v.9, n.7, p.1024–1039, 1991.

HARTMAN, J. H.; OUSTERHOUT, J. K. The Zebra Striped Network File System. In: JIN, H.; CORTES, T.; BUYYA, R. (Ed.). **High Performance Mass Storage and Parallel I/O**: technologies and applications. New York, NY: IEEE Computer Society Press: Wiley, 2001. p.309–329.

HERMANN, E.; ÁVILA, R.; NAVAUX, P.; DENNEULIN, Y. Metaserver Locality and Scalability in a Distributed NFS. In: VECPAR, 2006, Rio de Janeiro. **Proceedings...** [S.l.: s.n.], 2006.

HERMANN, E.; CONRAD, D. F.; BOITO, F. Z.; KASSICK, R. V.; AVILA, R. B.; NAVAUX, P. O. A. Utilização de Recursos Alocados pelo Usuário para Armazenamento de Dados no Sistema de Arquivos dNFSp. In: WORKSHOP EM SISTEMAS COMPUTACIONAIS DE ALTO DESEMPENHO, WSCAD, 7., 2006, Ouro Preto - MG. **Anais...** [S.l.: s.n.], 2006.

HILDEBRAND, D.; HONEYMAN, P. Exporting Storage Systems in a Scalable Manner with pNFS. In: IEEE / 13TH NASA GODDARD CONFERENCE ON MASS STORAGE SYSTEMS AND TECHNOLOGIES, 22., 2005. **Proceedings...** [S.l.]: IEEE Computer Society, 2005. p.18–27.

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 Trans. Comput. Syst.**, New York, NY, USA, v.6, n.1, p.51–81, 1988.

I-CLUSTER 2. Disponível em: <http://i-cluster2.inrialpes.fr>. Acesso em: jul. 2006.

IAMNITCHI, A.; FOSTER, I.; NURMI, D.; FOSTER. A Peer-to-Peer Approach to Resource Location in Grid Environments. In: SYMP. ON HIGH PERFORMANCE DISTRIBUTED COMPUTING, 2002. **Proceedings...** [S.l.: s.n.], 2002.

JALOTE, P. **Fault tolerance in distributed systems**. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1994.

JI, M. et al. Archipelago: an Island-Based file system for highly available and scalable internet services. In: USENIX WINDOWS SYSTEMS SYMPOSIUM, 4., 2000. **Proceedings...** [S.l.: s.n.], 2000.

JR., R. P. B.; BUSKENS, R. W. An Adaptive Distributed System-Level Diagnosis Algorithm and Its Implementation. In: INTERNATIONAL SYMPOSIUM ON FAULT-TOLERANT COMPUTING, FTCS, 21., 1991. **Proceedings...** [S.l.: s.n.], 1991. p.222–229.

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

KELEHER, P. J. **Lazy release consistency for distributed shared memory**. 1995. Tese (Doutorado em Ciência da Computação) — Rice University, Houston, TX, USA.

LAMPORT, L.; SHOSTAK, R.; PEASE, M. The Byzantine Generals Problem. **ACM Transactions on Programming Languages and Systems, TOPLAS**, New York, NY, USA, v.4, n.3, p.382–401, 1982.

LATHAM, R. et al. A Next-Generation Parallel File System for Linux Clusters. **LinuxWorld**, [S.l.], v.2, n.1, Jan. 2004.

LEVY, E.; SILBERSCHATZ, A. Distributed file systems: concepts and examples. **ACM Comput. Surv.**, New York, NY, USA, v.22, n.4, p.321–374, 1990.

LOMBARD, P. **NFSP** : Une solution de stockage distribué pour architectures grande échelle. 2003. Thèse de doctorat en informatique — Institut National Polytechnique de Grenoble, France.

LOMBARD, P.; DENNEULIN, Y. NFSp: a distributed nfs server for clusters of workstations. In: INTERNATIONAL PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM, 16., 2002, Washington, DC, USA. **Proceedings...** [S.l.]: IEEE Computer Society, 2002. p.352.

LOMBARD, P.; DENNEULIN, Y.; VALENTIN, O.; LEBRE, A. Improving the Performances of a Distributed NFS Implementation. In: INTERNATIONAL CONFERENCE PARALLEL PROCESSING AND APPLIED MATHEMATICS, 5., 2003. **Proceedings...** [S.l.]: Springer, 2003. p.405–412. (Lecture Notes in Computer Science, v.3019).

LYU, M. R.; MENDIRATTA, V. B. Software fault tolerance in a clustered architecture: techniques and reliability modeling. In: AEROSPACE CONFERENCE, 1999. **Proceedings...** [S.l.]: IEEE, 1999. p.141–150.

MORRIS, J. H. et al. Andrew: a distributed personal computing environment. **Commun. ACM**, New York, NY, USA, v.29, n.3, p.184–201, 1986.

PATTERSON, D. A.; GIBSON, G.; KATZ, R. H. A case for redundant arrays of inexpensive disks (RAID). In: ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, 1988, New York, NY, USA. **Proceedings...** New York: ACM Press, 1988. p.109–116.

PAWLOWSKI, B. et al. NFS Version 3: design and implementation. In: USENIX SUMMER, 1994. **Proceedings...** [S.l.: s.n.], 1994. p.137–152.

PAWLOWSKI, B. et al. The NFS Version 4 Protocol. In: INTERNATIONAL SYSTEM ADMINISTRATION AND NETWORKING CONFERENCE, 2., 2000. **Proceedings...** [S.l.: s.n.], 2000. p.94.

POPEK, G. J. et al. Replication in Ficus Distributed File Systems. In: WORKSHOP ON MANAGEMENT OF REPLICATED DATA, 1990. **Proceedings...** [S.l.]: IEEE Computer Society, 1990. p.20–25.

REISNER, P. **Festplattenspiegelung übers Netzwerk für die Realisierung hochverfügbarer Server unter Linux**. 2000. Dissertação (Mestrado em Ciência da Computação) — Vienna university of technology.

ROSARIO, J. M. del; CHOUDHARY, A. N. High-performance I/O for massively parallel computers: problems and prospects. **Computer**, Los Alamitos, CA, USA, v.27, n.3, p.59–68, 1994.

ROSENBLUM, M.; OUSTERHOUT, J. K. The Design and Implementation of a Log-Structured File System. **ACM Transactions on Computer Systems**, [S.l.], v.10, n.1, p.26–52, 1992.

SANDBERG, R. et al. Design and Implementation of the Sun Network Filesystem. In: SUMMER USENIX CONF., 1985, Portland OR (USA). **Proceedings...** [S.l.: s.n.], 1985. p.119–130.

SCHLICHTING, R. D.; SCHNEIDER, F. B. Fail-Stop Processors: an approach to designing fault-tolerant computing systems. **Computer Systems**, [S.l.], v.1, n.3, p.222–238, 1983.

SCHWAN, P. Lustre: building a file system for 1000-node clusters. In: LINUX SYMPOSIUM, 2003. **Proceedings...** [S.l.: s.n.], 2003.

SHEPLER, S. et al. **Network File System (NFS) version 4 Protocol**: RFC 3530. [S.l.]: IETF, 2003.

SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE, G. **Operating System Concepts**. New York, NY, USA: John Wiley & Sons, 2001.

SRINIVASAN, R. **RPC**: remote procedure call protocol specification version 2: RFC 1831. [S.l.]: IETF, 1995.

SUN MICROSYSTEMS. **NFS**: network file system protocol specifications: RFC 1094. [S.l.]: IETF, 1989.

ZHU, Y.; JIANG, H.; QIN, X.; FENG, D.; SWANSON, D. Improved Read Performance in a Cost-Effective, Fault-Tolerant Parallel Virtual File System (CEFT-PVFS). In: IEEE/ACM INTERNATIONAL SYMPOSIUM ON CLUSTER COMPUTING AND THE GRID, CCGRID, 2003, Tokyo, Japan. **Proceedings...** [S.l.: s.n.], 2003.