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

RODRIGO VIROTE KASSICK

Reconfiguração Automática de I/O para Aplicações Paralelas no Sistema de Arquivos dNFSp2

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, julho de 2010

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Kassick, Rodrigo Virote

Reconfiguração Automática de I/O para Aplicações Paralelas no Sistema de Arquivos dNFSp2 / Rodrigo Virote Kassick. – Porto Alegre: PPGC da UFRGS, 2010.

108 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2010. Orientador: Philippe O. A. Navaux.

1. Armazenamento, Sistema de Arquivos Paralelo, Sistema de Arquivos Dinâmico. I. Navaux, Philippe O. A.. II. Título.

“Einstein argued that there must be simplified explanations of nature, because God is not capricious or arbitrary. No such faith comforts the software engineer.”

— FRED BROOKS, "NO SILVER BULLET", 1987

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

AppIOD	IOD Exclusivo de Aplicação
DLM	Distributed Lock Manager
dNFSp	Distributed NFSp
HPC	High Performance Computing
I/O	Input/Output
IOD	Input/Output Daemon
MGS	Management Service
MPI	Message Passing Interface
MPI-IO	MPI I/O Library
NFS	Network File System
OBD	Object Based Disks
OST	Object Storage Targets
PFM	Performance Monitoring Tool
PFS	Parallel File Systems
POSIX	Portable Operating System Interface for UniX
PVFS	Parallel Virtual File System
RAID	Redundant Array of Independent Disk
RPC	Remote Procedure Call
TCP	Transport Control Protocol
UDP	User Datagram Protocol
VFS	Virtual File System Layer
VIOD	Virtual IOD
VSS	Very Simple Scheduler
XDR	eXternal Data Representation

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Diversas aplicações executadas em ambientes de cluster necessitam de uma área de armazenamento permanente com alta capacidade e que forneça uma visão homogênea dos dados entre todos os nós. Esta área compartilhada é comumente implementada através de um sistema de arquivos distribuído, permitindo o acesso através da abstração mais comum para gerenciamento de dados. A disparidade entre poder de processamento e desempenho de dispositivos de armazenamento atuais, no entanto, torna tais sistemas um ponto crítico ao desempenho de aplicações paralelas que lidam com grandes volumes de dados.

Ambientes de cluster podem apresentar execução concorrente de aplicações em conjuntos independentes de máquinas. Desta forma, uma grande quantidade de clientes com características distintas farão acessos ao sistema de arquivos compartilhado. Em tais casos, o dimensionamento do sistema de armazenamento distribuído nem sempre poderá prover o desempenho necessário à execução das aplicações com os recursos inicialmente a ele destinados.

O presente trabalho propõe uma estratégia de reconfiguração dinâmica para o sistema de arquivos dNFSp. Esta estratégia leva em consideração o comportamento temporal presente em aplicações paralelas para inserir servidores de dados exclusivos a aplicações com alta demanda de I/O. Com a utilização de servidores exclusivos, torna-se possível isolar aplicações com comportamentos que causam grande perda de desempenho no sistema como um todo.

Foi desenvolvida uma ferramenta de monitoramento do desempenho junto aos servidores do dNFSp, de forma a solicitar novos servidores apenas quando a interação das fases de I/O das aplicações em execução interferirem no desempenho do sistema. Esses novos servidores são solicitados para um sistema escalonador de recursos para cluster, de forma a utilizar nós livres para o armazenamento de dados.

Os resultados mostram que a estratégia proposta é capaz de detectar a saturação do sistema de armazenamento e iniciar os servidores exclusivos, levando a um ganho de desempenho para as aplicações em execução. A reconfiguração dinâmica também mostrou-se capaz de evitar baixos desempenhos causados pela interação dos períodos de inatividade de aplicações temporais e pela utilização de diversos processos de I/O em um único nó.

Palavras-chave: Armazenamento, Sistema de Arquivos Paralelo, Sistema de Arquivos Dinâmico.

ABSTRACT

Several applications executed in cluster environments need a shared storage area with high capacity and a homogeneous view of the stored data to all processing nodes. This area is commonly implemented as a distributed file system, allowing the access to data through the well-known file abstraction. The great gap in performance of processors and storage devices, on the other hand, makes such system a critical point to the performance of parallel applications.

A common trait of large cluster environments is the concurrent execution of applications. In this scenario, many clients with distinct behaviors will compete to access the shared storage system. The number of I/O resources originally dedicated to this shared storage may provide unsatisfactory performance to the applications in this case.

This work proposes a dynamic reconfiguration strategy for the dNFSp file system. This strategy takes into consideration the temporal behavior of distributed applications to launch dedicated I/O resources to the more I/O-demanding applications. The exclusiveness of data servers allows for the isolation of access patterns that cause contention on the system, resulting in improved performance for all executing applications.

We developed a tool to monitor the performance of the storage servers in order to launch new servers only when the interaction of I/O phases from running applications cause each other's performance to drop. These resources are required to a batch scheduler system present on the cluster, allowing the use of computing nodes for temporary data storage.

The results show that the proposed metrics lead to the detection of performance saturation on the file system and the start of the dedicated resources, resulting in increased I/O performance. The reconfiguration has also been able to avoid some low performance situations caused by interactions of inactivity periods from temporal applications and by utilization of several I/O processes on the same cluster node.

Keywords: Storage, Paralle File System, Dynamic File System.

1 INTRODUÇÃO

A popularização de clusters de computadores (BAKER; BUYYA, 1999) nos últimos anos tornou a programação paralela via troca de mensagens extremamente popular para o desenvolvimento de aplicações de alto desempenho. A arquitetura de cluster permite agregar poder de processamento, memória, recursos de rede e armazenamento de diversas máquinas independentes e utilizá-las de forma coordenada para a execução de uma aplicação distribuída.

Com tal agregação de recursos, aplicações paralelas distribuem suas tarefas entre os nós disponíveis no sistema, efetuando o processamento dos dados necessários e trocando mensagens entre si, quando necessário. Diversos tipos de aplicações científicas utilizam esta estratégia com o intuito de melhorar seu desempenho e processar mais informações, aumentando a qualidade de seus resultados. Estas aplicações incluem simulações de modelos climáticos, de reações químicas, de comportamento de corpos celestes, de comportamento de partículas subatômicas em diversos cenários, etc.

As limitações que aplicações paralelas devem contornar dizem respeito, principalmente, à independência dos nós do cluster. Sem uma memória compartilhada, a comunicação entre as instâncias da aplicação deve ser feita através de troca de mensagens via rede. Aplicações distribuídas que possuem grande interdependência de dados processados por suas instâncias podem sofrer problemas para escalar seu desempenho com o aumento do número de processadores envolvidos devido ao maior uso dos recursos de rede.

A operação de grandes quantidades de dados gera, também, um problema com relação a seu armazenamento. Os dados iniciais devem, de alguma forma, ser transportados até as instâncias da aplicação e os dados de saída devem estar disponíveis para posterior análise. Os limites físicos de memória podem exigir, também, o armazenamento temporário de parte destes dados para serem posteriormente recuperados, seja pela mesma ou por outra instância da aplicação distribuída.

Surge, então, a necessidade de um espaço de armazenamento compartilhado por todas os nós de um cluster, permitindo às aplicações compartilharem dados através da noção de um sistema de arquivos. Sistemas de arquivos de rede como NFS provêm este tipo de funcionalidade, permitindo uma visão consistente sobre os dados a todos os nós de um cluster. Sua natureza centralizada, no entanto, não é capaz de lidar com o grande número de acessos a dados utilizados por diversas instâncias das aplicações paralelas, tornando o armazenamento um grave gargalo para aplicações de alto desempenho.

Para contornar este problema, foram desenvolvidos sistemas de arquivos distribuídos. Nestes sistemas, os dados são armazenados não em um único servidor, mas divididos entre um conjunto de máquinas independentes, permitindo o tratamento de requisições em paralelo, bem como a agregação das larguras de rede destes servidores. O transporte dos dados entre os servidores e os nós de processamento é feito usando a infra-estrutura

de rede já existente para a troca de mensagens.

Diversos sistemas de arquivos paralelos foram desenvolvidos com o intuito de explorar as vantagens do armazenamento distribuído de dados. Lustre e PVFS são exemplos bastante conhecidos na área e utilizados em ambientes de produção. Sistemas como pNFS e dNFSp são implementações que visam prover alto desempenho de I/O com a utilização do protocolo NFS, suportado nativamente por diversas implementações de sistemas Unix.

Um fator determinante para o desempenho do sistema de arquivos distribuído é a quantidade de servidores utilizados para o gerenciamento e armazenamento dos dados. Esta quantidade é, tradicionalmente, definida conforme a disponibilidade de hardware no sistema. Cabe às políticas do sistema de armazenamento ou de cada aplicação distribuir seus dados entre estes servidores ou um subconjunto deles.

Clusters de produção podem apresentar execução concorrente de aplicações em conjuntos independentes de nós. Desta forma, uma grande quantidade de nós com característica de acesso distintas irão acessar o sistema de arquivos compartilhado. O conjunto permanente de servidores destinados ao armazenamento pode, mesmo com políticas de acesso e distribuição de dados adequadas a cada aplicação, tornar-se inapto às necessidades de desempenho das aplicações em execução devido à falta de recursos disponíveis.

O dimensionamento do sistema de arquivos, portanto, torna-se uma tarefa crucial para seu desempenho. Dimensionar a quantidade de servidores destinados para I/O, no entanto, não é tarefa trivial. Deve-se levar em conta as características das aplicações que são executadas no cluster, quantidade de nós envolvidos, espaço de armazenamento necessário, etc. Uma estimativa que exija poucos recursos pode levar a um sistema que se torna rapidamente saturado. A utilização de muitos servidores pode ser eficiente quanto sobrecarregada, mas pode levar a uma subutilização dos recursos quando o cluster foi utilizado por aplicações com menores exigências de I/O.

Surge a necessidade de explorar a adaptação do sistema de arquivos em termos de *quantidade de recursos utilizados para I/O*. O objetivo desta adaptação é corrigir situações onde o sistema de armazenamento não é capaz de prover desempenho suficiente às aplicações em execução, utilizando nós de processamento livres para o armazenamento de dados.

Esta estratégia levanta algumas questões que devem ser estudadas. Inicialmente, deve-se avaliar em quais situações esta reconfiguração pode trazer benefícios ao desempenho das aplicações. Deve-se, também, definir métodos para avaliar o custo que esta reconfiguração irá causar devido à utilização de recursos de processamento.

Este trabalho propõe uma estratégia de reconfiguração dinâmica para o sistema de arquivos dNFSp através da utilização de *servidores exclusivos de dados para aplicações*. Nesta reconfiguração, são levadas em conta características de I/O das aplicações em execução em um determinado momento para definir as possibilidades de reconfiguração e os possíveis ganhos com ela obtidos. É apresentada a avaliação desta estratégia para o caso de aplicações com comportamento temporal de escrita, avaliando o ganho obtido com a reconfiguração automática. Também é avaliada a eficiência da reconfiguração quando os clientes utilizam diversos processos por cliente e quando estas escritas são efetuadas apenas por uma das instâncias.

O restante deste trabalho está dividido da seguinte maneira: os *Trabalhos Relacionados* são apresentados na Capítulo 2. Esse capítulo aborda sistemas de arquivos paralelos e estratégias de adaptação do sistema de I/O às necessidades das aplicações durante sua execução.

No Capítulo 3 é apresentado o sistema de arquivos dNFSp. São apresentadas as ca-

racterísticas que definem o desempenho deste sistema, bem como o funcionamento dos servidores exclusivos de dados que são utilizados para a reconfiguração dinâmica.

Os *Padrões de Acesso de Aplicações Paralelas* que definem as classes de aplicações utilizadas na avaliação das estratégias de reconfiguração são apresentadas no Capítulo 4. Esse capítulo apresenta as características de I/O de cada uma e ressalta as que são mais adequadas para a utilização com servidores exclusivos.

O *Modelo de Reconfiguração* é detalhado no Capítulo 5. São apresentados os pré-requisitos necessários para a reconfiguração do I/O de uma aplicação e os critérios utilizados para medir o custo associado.

O Capítulo 6 descreve a implementação do modelo junto ao dNFSp. Esse capítulo expõe as decisões de projeto tomadas durante o trabalho, as ferramentas desenvolvidas para a avaliação do modelo e o funcionamento do sistema.

A *Avaliação do Modelo* é apresentada no Capítulo 7. Nesse capítulo são avaliados o comportamento do sistema com a utilização de servidores exclusivos, a detecção de saturação de I/O das aplicações e o ganho em desempenho obtido com a reconfiguração dinâmica com a presença de aplicações com comportamento temporal.

Por fim, no Capítulo 8, são apresentadas as conclusões obtidas com este trabalho e os trabalhos futuros.

2 TRABALHOS RELACIONADOS

O acesso a arquivos com grande volumes de dados é uma situação freqüentemente observada em ambientes de cluster. Aplicações paralelas fazem uso desta abstração para o armazenamento de grandes conjuntos de dados e permitir seu acesso às várias instâncias da aplicação (SMIRNI; REED, 1997; CARNS et al., 2000; AMIRI; GIBSON; GOLDING, 2000), seja para acesso compartilhado aos dados por todas as instâncias de uma aplicação ou para o armazenamento de dados para posterior análise. Para prover tal acesso, faz-se necessário um sistema de armazenamento que forneça um espaço de armazenamento que possa ser acessado por todas as máquinas onde as instâncias da aplicação são executadas e que mantenha estes dados consistentes quando acessados de diferentes origens.

Um exemplo de sistema de arquivos que provê este tipo de abstração é o NFS (Network File System) (SUN MICROSYSTEMS, 1989). Neste tipo de sistema, um único servidor permite acesso remoto a um diretório de seu sistema de arquivos local. Todos os clientes que acessam este servidor têm uma visão consistente dos arquivos criados por outros clientes e das alterações feitas nos arquivos compartilhados.

O NFS, no entanto, utiliza apenas um servidor para prover acesso aos arquivos compartilhados por todas as máquinas de um *cluster*. Com o aumento do número de clientes, essa infra-estrutura centralizada torna-se ineficiente devido à saturação das capacidades de disco e rede desta máquina servidora. (MARTIN; CULLER, 1999).

Devido ao lento aumento no desempenho de dispositivos de armazenamento quando comparado ao de processadores (NIEUWEJAAR et al., 1996), fazem-se necessárias técnicas que explorem o paralelismo destes dispositivos para aumentar a performance do sistema de arquivos. Para tanto, utiliza-se algum critério para distribuição dos dados das aplicações entre um conjunto de discos independentes, permitindo agregar sua vazão de leitura e escrita.

Soluções de mercado como RAID (CHEN et al., 1994) incorporam, no nível de dispositivo, o tratamento para a distribuição de dados entre vários discos físicos aos quais o hardware tem acesso. Esta distribuição pode incluir informações de redundância que permitem ao sistema continuar em funcionamento mesmo na presença de falhas em alguns dispositivos. O acesso aos dados, no entanto, continua sendo feito através de um recurso centralizado – o controlador de discos – que reside em apenas um servidor.

Para aumentar o desempenho do sistema de armazenamento na presença de um grande número de clientes, podem ser utilizados não apenas vários dispositivos de armazenamento, mas também vários servidores. Desta forma, é possível dividir entre eles não só a carga associada aos dispositivos de armazenamento, mas também a referente à transmissão dos dados lidos ou gravados, bem como a de seu processamento, explorando o paralelismo das operações de entrada e saída. Este tipo de solução é chamada de **sis-**

tema de arquivo paralelo (Parallel File System, PFS), uma vez que as operações sobre os arquivos são atendidas paralelamente nos servidores.

Para manter a visão de um único sistema de arquivos compartilhado, devem ser empregados mecanismos de sincronização que permitam aos vários clientes e servidores saberem onde encontram-se os dados gravados por diferentes clientes. Esta sincronização, no entanto, deve ser feita de forma escalável, permitindo a inclusão de mais recursos ao sistema de armazenamento sem que ele sofra grande degradação de performance.

2.1 Conceitos Básicos de Sistemas de Arquivos Paralelos

Com a utilização de recursos distribuídos para o armazenamento de dados, faz-se necessária uma divisão clara de tarefas entre estes recursos. Esta distribuição de tarefas também exige a aplicação de mecanismos de coerência para garantir que os acessos concorrentes dos clientes gerem dados consistentes.

A distribuição das tarefas também gera questões a respeito da escalabilidade de um sistema de arquivos distribuído. A quantidade de servidores aos quais os clientes podem fazer requisições, a necessidade ou não de mecanismos de consistência e sincronização entre servidores e clientes, dentre outros aspectos, influenciam o quanto é possível escalar um PFS.

2.1.1 Dados e Meta-dados

Para permitir uma sincronização escalável, os sistemas de arquivos paralelos optam por uma divisão clara entre os dados dos arquivos e seus meta-dados, com diferentes garantias de consistência para cada um.

As operações sobre *meta-dados* dizem respeito à gerência de informações sobre os arquivos, como nomes, permissões e datas de acesso. Além destas, fazem parte dos meta-dados informações sobre a localização dos blocos de dados do arquivo. Esta localização pode ser determinada por um esquema fixo, como uma distribuição cíclica de blocos entre os servidores de dados, incorporar esquemas de dinamismo, utilizando servidores de dados diferenciados para prover melhor desempenho para algumas aplicações, ou distribuir os dados de acordo com um esquema sugerido pela aplicação. Em qualquer um destes casos, é função do serviço de meta-dados manter este mapeamento consistente entre os vários clientes que possam estar acessando o mesmo arquivo.

O acesso aos dados gravados no sistema de arquivos paralelo se dá primeiramente pelo serviço de meta-dados. Através de uma chamada de abertura de arquivo o cliente passa a ter uma referência ao arquivo ou ao mapeamento de seus dados nos servidores de dados. O acesso aos dados propriamente dito é, então, feito com base neste mapeamento.

2.1.2 Acesso Direto *vs* Acesso Indireto

Os arquivos podem ser acessados de forma direta ou indireta. Quando feito diretamente, o cliente, após obter a referência ao arquivo e seu mapeamento, passa a enviar e receber mensagens diretamente dos servidores de dados, requisitando a leitura ou gravação dos blocos de dados. A Figura 2.1 ilustra esta situação. O cliente, primeiramente, obtém os meta-dados do arquivo para saber como localizá-lo nos servidores de dados. Com esta informação, o cliente pode solicitar diretamente aos servidores que lhe enviem os dados desejados ou enviar operações de escrita. Desta forma, as operações de I/O são feitas em paralelo nos vários servidores. Esta estratégia é adotada por soluções como o PVFS (CARNS et al., 2000), o Lustre (Cluster File Systems, Inc., 2002) e o

Expand (GARCIA et al., 2003).

Quando o acesso é feito indiretamente, o cliente envia mensagens a um conjunto de servidores intermediários. Todas as operações de leitura e escrita devem passar primeiro por estes servidores e estes as encaminham para os servidores de dados. Com este tipo de solução, é dispensado o conhecimento da distribuição de dados pelo cliente.

Enquanto o acesso indireto adiciona um custo fixo em cada acesso devido à duplicação de mensagens, ele permite a utilização de esquemas dinâmicos de distribuição dos dados sem a necessidade de atualizar o mapeamento de dados em todos os cliente interessados, aplicar políticas de escalonamento de acesso (LEBRE et al., 2006; ZHOU; WEI, 2003; JAIN et al., 1997) ou efetuar balanceamento de carga entre réplicas. PFS com acesso indireto permitem, também, fornecer um mecanismo de armazenamento paralelo através de protocolos legados, como NFS. É o caso de sistemas de arquivos como NFSp (LOMBARD; DENNEULIN, 2002), do dNFSp (ÁVILA, 2005), do Split-Server NFS (HILDEBRAND; HONEYMAN, 2005a) e do pNFS (HILDEBRAND; HONEYMAN, 2005b).

O caso do acesso indireto é ilustrado na Figura 2.2. Após abrir o arquivo, o cliente envia requisições de leitura ou escrita para um servidor intermediário. Este, por sua vez, localiza as informações do arquivo em questão em um dos repositórios de meta-dados do sistema e, em posse desta informação, envia as instruções para os servidores de dados. Os servidores de dados podem, neste caso, enviar os dados diretamente aos clientes, sem necessidade de passar pelos servidores intermediários.

2.1.3 Escalabilidade de Sistemas de Arquivos Paralelos

Uma das questões que devem ser tratadas em sistemas de armazenamento para clusters é a escalabilidade: um PFS utilizado em ambientes de cluster tem que ser capaz de tratar requisições de um grande número de clientes. Tanto os dispositivos de armazenamento quanto o serviço de meta-dados devem estar preparados para esta carga.

O acesso aos dados tradicionalmente é o ponto de maior atenção para garantir uma boa escalabilidade para o sistema. O número de servidores que irão atender às requisições dos clientes deve ser dimensionado de forma que seja possível manter um mínimo de desempenho em casos de carga excessiva e que eventuais mecanismos de consistência entre os servidores não se tornem muito pesados.

No armazenamento de dados, é possível utilizar técnicas para diminuir o efeito da concorrência entre os vários clientes. É possível, por exemplo, replicar os dados mais acessados e utilizar mecanismos de balanceamento de carga para evitar a sobrecarga nos servidores. As operações de escrita, no entanto, podem sofrer degradação de performance devido a eventuais mecanismos de sincronização das réplicas. A complexidade destes algoritmos pode, portanto, ser um dos principais fatores na escalabilidade dos nós de armazenamento.

O serviço de meta-dados também necessita garantir escalabilidade quando o número de clientes efetuando acessos se torna grande ou quando o número de servidores e clientes com réplicas destas informações aumenta (FAN; XIONG; MA, 2004; BRANDT et al., 2003; DEVULAPALLI; OHIO, 2007) . Apesar de informações de meta-dados serem pequenas quando comparadas com o tamanho de arquivos que um PFS necessita gerenciar, a escalabilidade desse subsistema é vital, uma vez que é ele que provê as primeiras informações necessárias para o acesso aos arquivos. Assim como no caso dos servidores de dados, os servidores de meta-dados devem ser capazes de manter a visão consistente de um sistema de arquivos compartilhado e, ainda assim, evitar que tal sincronização acarrete grande carga na rede ou aumente significativamente o tempo necessário para acessar

um arquivo.

Caso o acesso seja feito indiretamente, é importante que os servidores intermediários estejam adequadamente dimensionados para arcar com a carga gerada pelos clientes e repassar as requisições aos servidores de dados de forma eficiente. Soluções que utilizam o acesso indireto geralmente optam por distribuir este serviço, evitando o gargalo de um único servidor. Este é o caso do dNFSp e do Split-Server NFS.

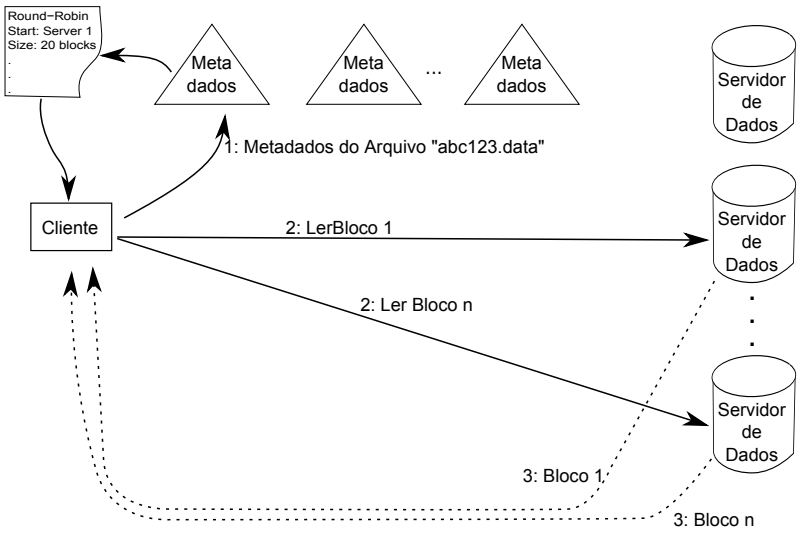


Figura 2.1: Acesso direto aos dados

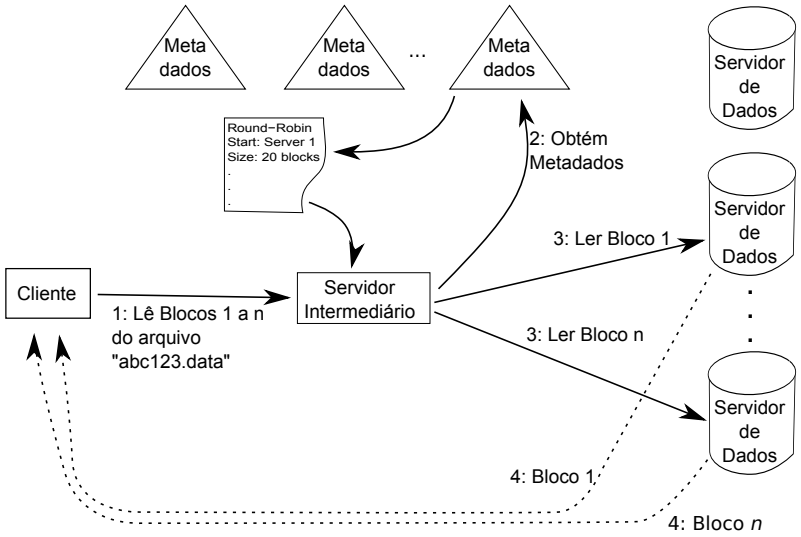


Figura 2.2: Acesso Indireto aos dados

2.2 Sistemas de Arquivos Paralelos

As seguintes seções apresentam alguns sistemas de arquivos encontrados na bibliografia. Para cada sistema, são ressaltadas algumas características consideradas relevantes para a elaboração deste trabalho.

2.2.1 Harp

O sistema de arquivos Harp (Highly Available, Reliable, Persistent File System) (LISKOV et al., 1991) é um sistema de arquivos cujo principal objetivo era prover armazenamento com alta disponibilidade. Sua arquitetura consistia de um servidor primário e diversos servidores de *backup*. O acesso ao sistema era feito através de um servidor NFS, de forma que as estratégias de replicação eram transparentes aos clientes.

O Harp foi um dos primeiros sistemas de arquivos distribuídos que empregou replicação total de dados sem necessidade de hardware específico. A utilização de uma camada NFS para o acesso também deve ser destacada, uma vez que garante a disponibilidade dos dados de forma transparente aos clientes.

2.2.2 Bigfoot-NFS

O Bigfoot NFS (KIM et al., 1994) é um sistema de arquivos distribuído que permite a utilização transparente de vários servidores NFS, agregando seu espaço de armazenamento. Estes servidores são apresentados como um único sistema de arquivos com um mesmo espaço de nomes.

O Bigfoot utiliza um *daemon* que é executado junto aos clientes e é acessado através da camada VFS. Este *daemon* é responsável por contatar os servidores distribuídos e encaminhar adequadamente as requisições dos clientes.

A unidade de distribuição é o próprio arquivo, ou seja, todos os dados de um arquivo devem estar armazenados em um único servidor. O *daemon* executado nos clientes é responsável por descobrir a localização de um arquivo nos servidores de dados.

Esta localização é feita contatando todos os servidores de dados. Mensagens de *lookup* são enviadas a todos os servidores em paralelo. Apenas um deverá responder confirmando a presença do arquivo e o *daemon* registra um mapeamento do arquivo para este servidor em sua memória. Requisições subseqüentes são encaminhadas diretamente ao servidor correto.

As estratégias adotada pelo Bigfoot têm duas características importantes. A primeira é a utilização de servidores NFS padrão para o armazenamento de dados. A segunda é a inexistência de um serviço de meta-dados. O Bigfoot apenas necessita armazenar em memória um mapeamento dos arquivos abertos pelas aplicações para um dos servidores de dados.

2.2.3 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. O Vesta permite definir a distribuição dos dados de forma diferente para cada arquivo, permitindo ao usuário determinar o número de nós e a granularidade da distribuição no momento da criação do arquivo.

O Vesta permite dividir os dados de um arquivo em particionamentos lógicos disjuntos. A disposição do particionamento lógico é especificada no momento da abertura do arquivo. Desta forma, é possível que cada cliente acesse um mesmo arquivo com uma visão lógica diferente.

O acesso aos meta-dados é feito de forma direta através do cálculo de uma função de *hash* sobre o nome do arquivo para se determinar o processador que contém os metadados. Nas operações de escrita e leitura não existe necessidade de se acessar os metadados, pois o sistema de arquivos garante que o cliente possui localmente toda a infor-

mação necessária para se localizar 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.

2.2.4 Expand

O sistema de arquivos Expand (GARCIA et al., 2003) é um sistema de arquivos paralelo cujo objetivo é agregar vazão e capacidade de armazenamento de diversos servidores NFS padrão. O acesso ao sistema é feito através de uma biblioteca que provê a lógica de distribuição de arquivos nos servidores disponíveis.

Um arquivo no sistema Expand é composto de vários sub-arquivos distribuídos entre os servidores NFS. Cada arquivo pode ser dividido e distribuído em diferentes quantidades de servidores e políticas – como por exemplo, arquivos particionados de forma cíclica ou arquivos redundantes utilizando RAID 4 ou RAID 5.

Cada sub-arquivo possui uma cópia dos meta-dados do arquivo em um pequeno cabeçalho em seu início. Apenas um destes sub-arquivos, no entanto, possui a cópia mais atualizada dos meta-dados. Este nó, chamado de nó mestre, é determinado por uma função de *hash*.

2.2.5 AdExpand

O AdExpand – Adaptative Expand – (PEREZ et al., 2003) é um sistema de arquivos baseado no Expand. Seu principal objetivo é construir um sistema de arquivos paralelo para ambientes heterogêneos de propósito geral – como clusters heterogêneos ou grades de computadores.

O AdExpand permite a inclusão de novos servidores de dados para efetuar o redimensionamento do sistema compartilhado. Estes servidores podem ter espaços de armazenamento desiguais. Para tentar maximizar o espaço de armazenamento total, o AdExpand provê diferentes políticas de seleção de servidores para o armazenamento de dados. Desta forma, evita-se a interrupção do serviço devido à falta de espaço em apenas um dos servidores. A carga nos servidores que possuem mais espaço, no entanto, tende a ser maior, uma vez que este irá armazenar frações de mais arquivos do que os servidores com menos espaço.

2.2.6 PVFS – Parallel Virtual File System

O PVFS (CARNS et al., 2000) é um sistema de arquivos voltado para o processamento de alto desempenho que permite a utilização de servidores com hardware comumente encontrado no mercado. Suas funcionalidades são divididas entre servidores de meta-dados (meta-servidores) e servidores de dados. O alto desempenho deste sistema é obtido através da distribuição dos dados dos arquivos em vários blocos armazenados de forma distribuída entre os servidores, permitindo leituras e escritas paralelas. Os dados são armazenados nos servidores em arquivos comuns, chamados *datafiles*. O número de *datafiles* nos quais um arquivo é dividido pode ser definido no momento de sua criação, bem como a quantidade de servidores que os irão armazenar.

O PVFS utiliza acesso direto aos servidores de dados. Os clientes utilizam os meta-servidores para consultar as informações necessárias ao primeiro acesso e as mantêm em uma *cache* local. Eventuais alterações efetuadas aos meta-dados são enviadas aos servidores de meta-dados e são percebidas pelos clientes após a invalidação de suas caches por time-out. O acesso aos dados no PVFS pode ser feito tanto através de uma biblioteca, com

rotinas específicas para este sistema de arquivos, ou com uma interface POSIX padrão, implementada como um módulo do *kernel* do sistema operacional.

A versão 2 do PVFS conta com servidores de meta-dados distribuídos. Para aumentar o desempenho, o protocolo foi feito de forma *stateless*, inclusive com a eliminação de *locks*. Esta estratégia visa permitir ao sistema escalar para uma grande quantidade de clientes com pouco *overhead* de gerenciamento, delegando às aplicações eventuais políticas de consistência de dados necessárias devido ao acesso concorrente. Esta estratégia é mantida em trabalhos na literatura (LANG et al., 2009; KULKARNI; GABRIEL, 2009) e pode ser provida por API's de I/O de alto nível como MPI-I/O.

O PVFS possui uma configuração estática – uma vez que são definidos os servidores de dados e meta-dados, a configuração do sistema não pode ser alterada sem interrupção do serviço. Não é possível, portanto, a inclusão ou remoção de servidores para adequar o sistema para novas situações. Também não são oferecidos recursos via software para tolerância a falhas, como replicação e substituição de servidores. A tolerância a falhas em servidor provida pelo PVFS requer dispositivos de armazenamento compartilhados por vários servidores.

Alguns trabalhos visam aumentar o grau de dinamismo do PVFS para melhorar seu suporte a tolerância a falhas e solucionar problemas de gargalos. O CEFT-PVFS (ZHU et al., 2003) (Cost Effective Fault Tolerant PVFS) é um sistema de arquivos baseado no PVFS que utiliza replicação de dados para fornecer tolerância a falhas, permitindo a substituição de servidores falhos por servidores funcionais em um grupo de *backup*. A configuração destes servidores, no entanto, também é estática, não sendo possível aumentar o número de servidores.

Em (KUNKEL, 2007), é estudado o efeito de um mal balanceamento de carga nos servidores do sistema de arquivos e proposto um mecanismo de redistribuição de dados para melhorar seu desempenho. Esta redistribuição é feita de forma automática pelo sistema de arquivos através da transferência de um *datafile* de um servidor considerado sobrecarregado para um considerado subutilizado. A detecção de gargalos é revista em (KUNKEL; LUDWIG, 2008). Neste último trabalho, o monitor de desempenho é estendido para permitir análise *off-line* do comportamento do sistema, permitindo a um administrador identificar servidores com problemas de desempenho.

O gerenciamento de meta-dados no PVFS também é objeto de estudo para melhorar seu desempenho para cenários observados com algumas aplicações. O desempenho destas operações é essencial para o desempenho de algumas aplicações, dado que alguns workloads apresentam muitas operações de meta-dados (ROSELLI; LORCH; ANDERSON, 2000). Em (DEVULAPALLI; OHIO, 2007), são estudadas estratégias para aumentar o desempenho de operações de criação de arquivos no ambiente de meta-servidores distribuído. Em (KUHN; KUNKEL; LUDWIG, 2009), a consistência de meta-dados é relaxada para isentar o sistema de armazenar algumas informações sobre os arquivos (como permissões e datas de criação) visando aumentar o desempenho do PVFS para arquivos pequenos. Esse gerenciamento relaxado pode ser ativado para cada diretório através da criação de um arquivo especial.

2.2.7 Lustre

O objetivo do sistema de arquivos Lustre (Cluster File Systems, Inc., 2002; MICROSYSTEMS, 2008) é remover os gargalos comumente encontrados nestes sistemas. Sua arquitetura é composta por um serviço de meta-dados (MDS, Meta-Data Server) centralizado e diversos servidores de dados, chamados OSS (Object Storage Servers, Servi-

dores de Armazenamento de Objetos). Um OSS gerencia dispositivos de dados chamados de OST's (Object Storage Targets). Juntamente com estes serviços, operam um serviço de gerenciamento – o MGS (Management Service, Serviço de Gerenciamento) – e um mecanismo de controle de *locks*, chamado DLM (Distributed Lock Manager, Gerenciador Distribuído de *Locks*). A arquitetura do Lustre é ilustrada na Figura 2.3.

O serviço de meta-dados é responsável por manter as informações de localização do arquivo nos OST's, bem como as informações referentes a permissões e tamanho. Uma vez que os desenvolvedores do Lustre consideram que as operações sobre os meta-dados são em pequena quantidade quando comparadas àquelas sobre dados, o serviço de meta-dados é centralizado, possuindo apenas servidores de *fail-over* para caso de falhas. Esta estratégia, no entanto, está sendo revista para a próxima versão do sistema.

Os clientes podem manter também os meta-dados em *cache*, evitando repetidas chamadas ao MDS. Fica a cargo dos MDS's manter a consistência entre as caches dos clientes, enviando mensagens de invalidação quando necessário.

Um OST representa um dispositivo capaz de armazenar os objetos de um arquivo e é hospedado em um servidor do sistema de arquivos. Um mesmo servidor pode gerenciar diversos OST's.

Os OST's armazenam seus dados em dispositivos chamados OBD's (Object-Based Disks). Um OBD representa uma interface genérica que permite ao Lustre utilizar indistintamente partições de discos com sistemas de arquivos locais (como *ext3*, *reiserfs* e *XFS*) ou dispositivos remotos conectados via FiberChannel ou iSCSI. Essa abstração permite cenários com utilização de dispositivos simples ou com soluções de alta disponibilidade para o armazenamento.

O armazenamento no Lustre é orientado a *objetos*. Um objeto representa uma unidade de armazenamento e pode descrever um arquivo inteiro ou apenas uma fração dele. Quando armazenado em vários objetos, os dados de um arquivo são divididos em *stripes* e distribuídos de forma circular entre os objetos. Cada objeto é gerenciado por um único OST. A configuração padrão do Lustre divide os arquivos em dois objetos[1] e os posiciona aleatoreamente entre os OST's disponíveis, resultando em um armazenamento semelhante a outros sistemas de arquivos paralelos.

Cada OST executa uma instância do gerenciador de *locks* distribuído. O DLM é responsável por manter a consistência dos dados gravados pelos clientes quando há operações conflitantes.

A tolerância a falhas é provida por servidores *fail-over*. Todas as instâncias dos serviços replicados devem compartilhar um mesmo OBD. Este compartilhamento pode ser feito através de dispositivos de armazenamento multi-porta ou através de mecanismos de sincronização de dados entre servidores. No caso desta última alternativa, é deixada a cargo do administrador a utilização de mecanismos externos ao Lustre para esta tarefa, como o DRDB (DRBD, 2006). Um mecanismo nativo de replicação está sendo estudado para a versão 2.0 do sistema.

A ativação de um servidor *fail-over* no Lustre é feita quando um cliente falha por *time-out* ao acessar um OST. Cabe ao serviço MGS receber notificações destes *time-outs* e orientar os clientes quanto às localizações alternativas dos objetos.

A comunicação entre clientes e servidores é feita através da biblioteca LNET (Lustre Networking). Em sua implementação, a LNET provê aos clientes um mecanismo de RPC com transporte otimizado para diversas tecnologias de interconexão, como Ethernet, Infiniband, Myrinet e Quadrics. A utilização de mecanismos de transporte específicos

[1] O número de objetos padrão em um sistema Lustre pode ser configurado pelo administrador.

para essas redes garante uma vantagem no desempenho do Lustre em comparação com a utilização de transportes genéricos como TCP/IP (YU et al., 2006).

No Lustre, novos OST's podem ser criados e disponibilizados aos clientes sem a interrupção dos serviços. A distribuição dos dados em objetos e o posicionamento dos objetos nos OST's, no entanto, são permanentes, a menos que o usuário explicitamente solicite tal modificação.

Algum grau de adaptação às características das aplicações e à eventual heterogeneidade do sistema de arquivos pode ser obtida com a técnica de *OST Pools* (MICROSYSTEMS, 2009). Um *Grupo de OST* (*Pool*) consiste em servidores que compartilham determinada característica definida pelo administrador. Estas características podem ser referentes, por exemplo, ao tipo de interconexão com os nós, à velocidade de acesso aos dispositivos de I/O, à capacidade de armazenamento destes dispositivos, etc. Isto permite agrupar servidores conectados a dispositivos RAID e utilizá-los apenas para aplicações que demandem alta vazão de dados, enquanto OST's conectados a dispositivos de I/O de menor custo, como discos comuns, são utilizados por aplicações que apenas necessitam de um grande espaço de armazenamento.

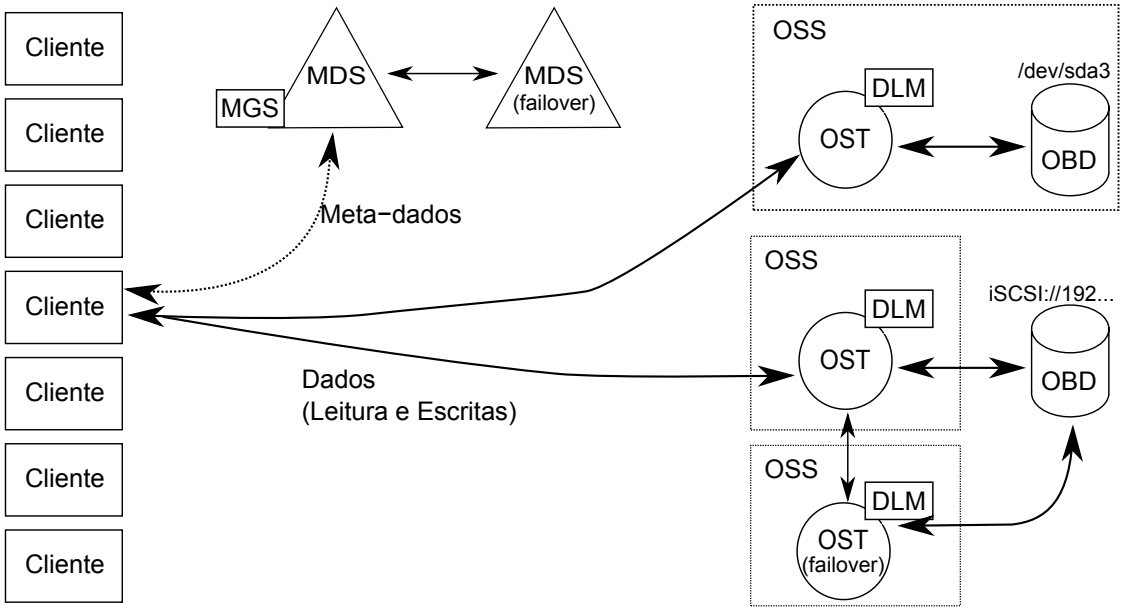


Figura 2.3: Arquitetura do Sistema de Arquivos Lustre

2.2.8 ClusterFile

O ClusterFile (ISAILA; TICHY, 2001) é um sistema de arquivos que utiliza as informações fornecidas pelas aplicações sobre a forma como os dados são dispostos nos arquivos para definir seu particionamento entre os servidores de dados. Este particionamento é feito através da divisão dos arquivos em sub-arquivos e sua distribuição entre os servidores de dados.

O ClusterFile efetua a distribuição dos dados em dois níveis. Primeiramente, a descrição dos dados de um arquivo é utilizada para definir a sua divisão em sub-arquivos. Esta divisão é feita de forma que uma instância da aplicação paralela tenha acesso, em um único arquivo, aos dados que serão por ela tratados. Isto permite efetuar leituras e escritas de um grande número de blocos de dados de forma contínua nos dispositivos de

armazenamento. A Figura 2.4 mostra esta divisão para dois esquemas de divisão de dados: a distribuição cíclica de um conjunto de blocos de dados entre os clientes (2.4a) e a divisão cíclica bloco a bloco bidimensional do arquivo entre 4 clientes (2.4b).

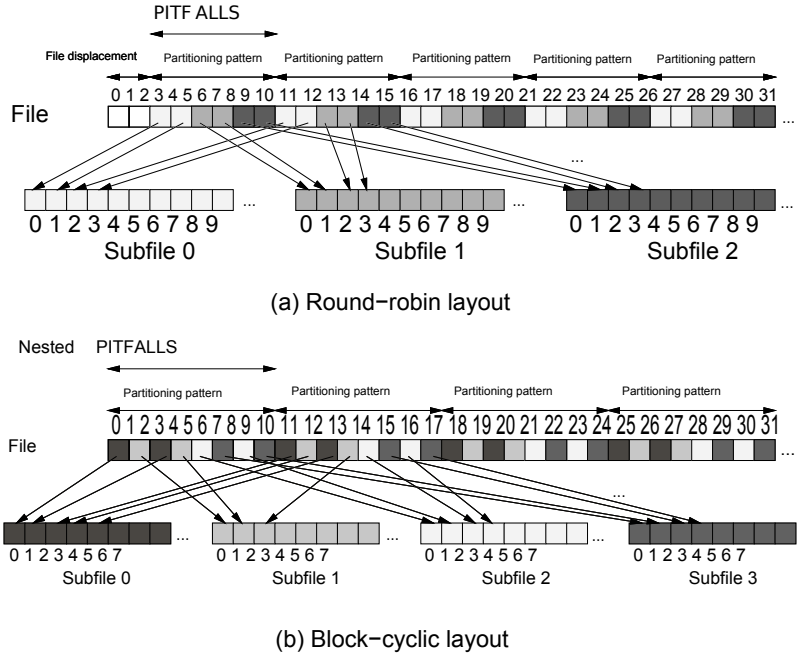


Figura 2.4: Divisão dos dados em sub-arquivos de acordo com a descrição

Em outro nível, o ClusterFile também divide os sub-arquivos entre os nós de armazenamento disponíveis. Quando o número de servidores de dados é maior que o de sub-arquivos, estes são divididos entre um conjunto disjunto de servidores, permitindo a agregação de bandas de disco e rede, explorando o paralelismo das operação de leitura e escrita. A Figura 2.5b ilustra esta situação.

Quando o número de sub-arquivos é maior que o número de servidores de dados disponíveis, cada sub-arquivo é inteiramente gravado em um único servidor. Esta abordagem visa tirar vantagem da seqüencialidade dos sub-arquivos. A Figura 2.5a ilustra esta situação.

Uma descrição de arquivo do ClusterFile pode ser derivada através dos *datatypes* da biblioteca MPI-I/O (ISAILA et al., 2006). Desta forma, aplicações podem utilizar uma biblioteca de uso genérico para descrever os dados gerenciados e permitir ao sistema de arquivos criar um mapeamento que melhor explore as características de acesso derivadas de tais informações.

2.2.9 XtreemFS

XtreemFS (HUPFELD et al., 2008) é um sistema de arquivos distribuído desenvolvido junto ao projeto XtreemOS, o qual visa atender às necessidades de aplicações de Grid. Sua arquitetura foi projetada para a utilização em redes WAN, levando em consideração aspectos de segurança, necessidade de replicação de dados e questões de sincronização.

O armazenamento dos dados dos arquivos é feito, como no Lustre, por objetos. Cada arquivo tem um mapeamento próprio de seus dados para um diferente número destes objetos. Estes últimos são gerenciados por servidores de dados – também chamados

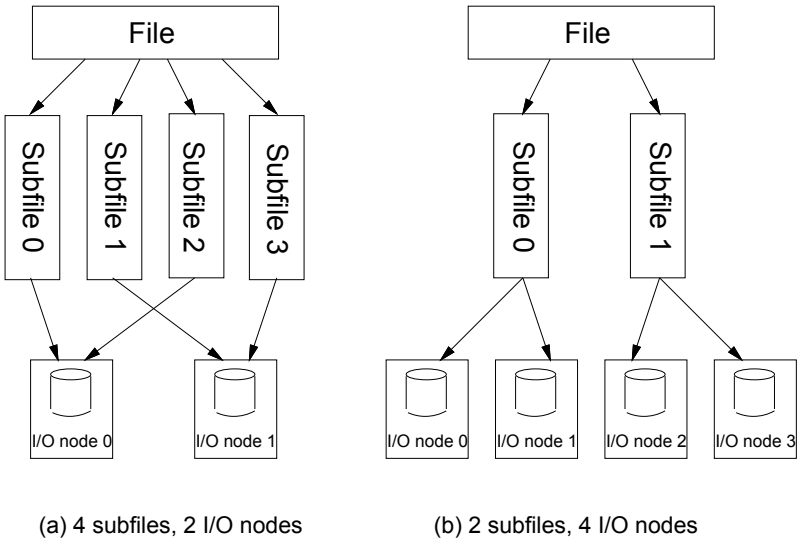


Figura 2.5: Divisão dos sub-arquivos entre os servidores de dados

OSD's – que são responsáveis apenas por atender requisições de leitura e escrita dos clientes. Um objeto pode estar replicado em vários OSD's.

Um serviço de meta-dados chamado *Metadata Replica Catalogs* (MRC) é executado em um conjunto de servidores distribuídos no grid. Clientes podem acessar qualquer um dos MRC's e solicitar informações a respeito dos arquivos uma vez que apresentem credenciais válidas – e.g. certificados X509 previamente concedidos.

É responsabilidade dos MRC controlar a localização dos objetos nos OSD's. Ao abrir um arquivo, o MRC contatado informa aos clientes dos OSD's disponíveis e suas respectivas distâncias em uma topologia de rede. O cliente, então, passa a fazer acessos aos OSD's mais próximos que possuam réplicas dos objetos.

Para cada objeto, existe uma réplica chamada *master*. Operações de escrita são sempre enviadas a esta réplica. Uma vez que a réplica master é alterada, operações de leitura passam a ser feitas apenas a esta réplica. O XtreemFS permite a sincronização automática das réplicas uma vez que o arquivo referente aos objetos for fechado.

Uma funcionalidade proposta no XtreemFS é a criação de réplicas *on-demand*. Esta replicação automática pode acontecer tanto durante inicialização de uma aplicação quanto durante sua execução (KOLBECK et al., 2008).

No primeiro caso, são utilizadas informações do sistema de distribuição de tarefas do XtreemOS sobre a distribuição de processos para detectar a necessidade de réplicas dos dados em OSD's mais próximos. No segundo caso, o comportamento da aplicação é estudado *on-line* e é tomada uma decisão quanto à criação de novas cópias, permitindo ao DFS adaptar-se conforme o comportamento da aplicação muda.

Adicionalmente, é previsto que estas réplicas possam ser apenas parciais. Mecanismos de mineração de dados são utilizados para prever os próximos acessos dos clientes e determinar que partes de um objeto serão necessárias no futuro próximo, permitindo iniciar a replicação antes de os clientes necessitarem dos dados.

Enquanto a versão 1.1 do XtreemFS[2] já possui suporte a replicação automática após criação e fechamento de um arquivo, a criação de réplicas *on-demand* ainda é considerado funcionalidade futura (STENDER; KOLBECK, 2009). Também não são encontrados tra-

[2]XtreemFS 1.1, lançado em 15 de Setembro de 2009

balhos científicos que avaliem a eficiência dos mecanismos propostos para esta replicação.

2.2.10 Ceph File System

Ceph (WEIL et al., 2006; WEIL, 2007) é um sistema de arquivos distribuído baseado em objetos, com servidores de dados e de meta-dados independentes e com suporte a dinamismo de servidores. Um dos princípios que levaram ao desenvolvimento do Ceph foi o de que sistemas na escala de *petabytes* são inerentemente dinâmicos: o sistema de armazenamento é construído de forma incremental, adicionando mais recursos conforme necessário devido aos distintos *workloads* aos quais o sistema é submetido ao longo de sua execução. Da mesma forma, falhas ocorrem como regra e mecanismos de replicação e recuperação de dados são essenciais.

O gerenciamento de meta-dados é feito por um conjunto de servidores independentes dos servidores de dados, podendo-se realizar inclusão e remoção dinâmica destes servidores. A divisão dos meta-dados utilizada a técnica de *sub-tree partitioning*, onde cada servidor é responsável por uma árvore de diretórios. Com a adição de novos servidores, ramos da árvore são divididos e passam a ser gerenciados por estes. Com esta estratégia, surge uma localidade de acesso de grupos de clientes em poucos (ou um único) meta-servidores de acordo com os arquivos utilizados pelos clientes de uma mesma aplicação. Isto é visto como vantagem pelos desenvolvedores, uma vez que *workloads* de uma aplicação com alta taxa de acesso a meta-dados não irão denegrir o desempenho de acesso aos meta-dados de outras aplicações.

O gerenciamento de dados é feito por *objetos*, sendo os servidores responsáveis por esta tarefa também chamados OSD's. O conjunto de OSD's é visto pelos clientes como um único volume lógico. Os dados são distribuídos nestes servidores através do algoritmo *CRUSH* (WEIL et al., 2006). O CRUSH mapeia novos objetos para servidores de forma aleatórea, enquanto redistribui dados cujas réplicas foram perdidas de forma uniforme. Este método estocástico visa evitar a criação de *hot-spots* e ao mesmo tempo evitar a ociosidade de novos recursos.

Uma das características distintas do Ceph é a forma como a localização dos objetos é feita: ao invés de manter uma mapeamento de objetos para OSD's junto aos meta-servidores, a localização é feita de forma algorítmica a partir de uma lista hierárquica de todos os OSD's que compõem o sistema. Isto permite que qualquer parte do sistema – cliente, meta-servidor ou OSD – localize qualquer objeto de um arquivo de forma independente.

No lado do cliente, o Ceph pode ser acessado tanto através de um módulo de *kernel* quanto através de uma biblioteca utilizada diretamente pela aplicação. Em qualquer um dos casos, a semântica oferecida é próxima à oferecida pela semântica POSIX. Devido a considerações de desempenho, no entanto, o programa cliente pode solicitar o relaxamento de alguns requisitos (como visibilidade imediata de escritas) através de *flags* quando da abertura dos arquivos.

2.2.11 Google File System

O Google File System (GFS) (GHEMAWAT; GOBIOFF; LEUNG, 2003) é um sistema de arquivos desenvolvido pela empresa Google para suas aplicações internas. Este sistema foi projetado levando em conta diversas características próprias aos programas por ela utilizados e à infra-estrutura de hardware utilizada.

As aplicações alvo do GFS são aquelas que efetuam acessos predominantemente seqüencias sobre um grande conjunto de dados. Os arquivos possuem tamanhos na ordem

dos terabytes e são "imutáveis" – seu conteúdo raramente necessita ser modificado, mas pode-se inserir novos dados ao seu final.

No aspecto de infra-estrutura de hardware, o sistema de armazenamento é constituído de uma imensa quantidade de componentes de baixo custo. Neste cenário, considera-se falhas como regra. Com estas características em mente, o GFS foi desenvolvido com otimizações para acessos seqüenciais, imutabilidade de dados já gravados e alta taxa de replicação para garantir tolerância a falhas.

A arquitetura do GFS, ilustrada na Figura 2.6 é constituída por um servidor de meta-dados centralizado e diversos servidores de dados. O servidor de meta-dados, chamado *GFS Master*, coordena a distribuição e eventual modificação dos arquivos. Os dados são armazenados em servidores chamados *Chunk Servers*. Tanto o master quando os chunk-servers são processos em modo de usuário que executam em sistemas Linux e armazenam suas informações no sistema de arquivos local.

Os arquivos gravados no GFS são divididos em blocos (chunks) de 64MB. A escolha deste tamanho de bloco reflete o objetivo de armazenar arquivos muito grandes. Um mesmo chunk é gravado em diversos chunk-servers visando tolerância a falhas.

O serviço de meta-dados é responsável por gerenciar o espaço de nomes e o seu mapeamento nos blocos. A informação a respeito de localização dos blocos, no entanto, não é mantida de forma persistente: durante a inicialização do sistema, o nó master solicita a todos os chunk-servers a lista de chunks neles armazenados e então armazena esta informação em memória. Esta decisão se deve ao dinamismo e à quantidade de servidores presentes no sistema. Sem a necessidade de manter um mapeamento consistente de chunks para chunk-servers, tanto o código dos chunk-servers quando o do master se torna mais simples.

Os arquivos armazenados no GFS raramente necessitam ser alterados – afora a inclusão de novos dados ao seu final. Como esta atualização é feita por processos distribuídos, o GFS fornece um mecanismo de *append atômico* para garantir a consistência dos dados. Este append atômico permite aos clientes gravar novos registros ao final dos arquivos de dados sem correr o risco de que um processo sobrescreva dados de outros.

A consistência binária dos dados, no entanto, não é garantida. Quando uma operação de append falha em um dos chunk-servers, o cliente que efetuou a operação é instruído a repetí-la. Isto implica em gravar dados duplicados em alguns chunk-servers e, em alguns casos, deixar regiões do arquivo com dados indefinidos. Isso não é considerado um problema para o GFS, obrigando as aplicações que o utilizam a aplicar mecanismos de consistência dos dados lidos.

O GFS fornece mecanismos de replicação automática dos chunks. Um número mínimo de réplicas pode ser configurado pelo usuário, com um número padrão de 3 réplicas. Conforme servidores de armazenamento sofrem falhas, o servidor master solicita a criação de novas réplicas a partir dos chunk-servers ainda funcionais. Esta replicação leva em consideração o posicionamento dos servidores dentro dos racks, visando obter tolerância a falhas e agregar a vazão de rede de vários racks.

O dinamismo do GFS é limitado aos seus mecanismos de replicação automática, não oferecendo mecanismos de adaptação dinâmica às características das aplicações. As API's que o utilizam, no entanto, podem prover tais funcionalidades.

A biblioteca MapReduce (DEAN; GHEMAWAT, 2004), também desenvolvida pela Google, permite dividir o espaço de dados de entrada entre diversas tarefas independentes que são executadas em um cluster. Sua implementação faz uso do GFS para armazenar dados de entrada e de saída e executa no mesmo cluster onde os dados estão armazenados.

Assim, o MapReduce procura executar processos nos chunk-servers que armazenam cópia dos dados de entrada da tarefa.

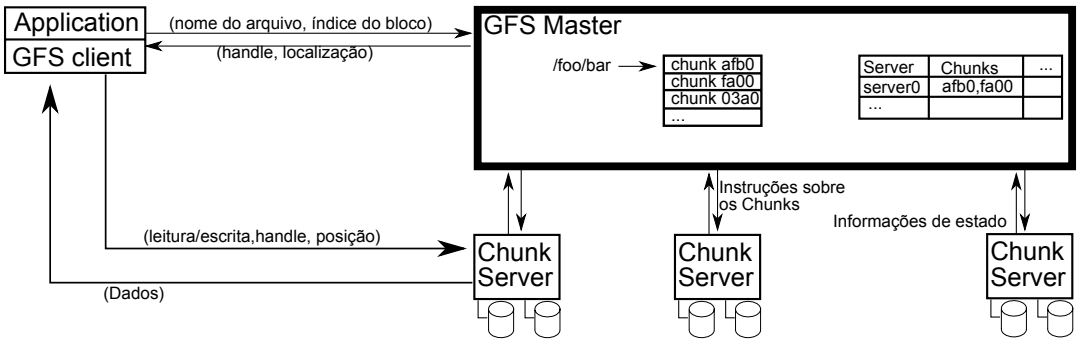


Figura 2.6: Arquitetura do Google File System

2.2.12 HDFS

O projeto Hadoop, organizado pela Apache Foundation, tem por objetivo fornecer ferramentas para desenvolvimento de sistemas distribuídos confiáveis e escaláveis. O Hadoop File System (HDFS) (Apache Foundation, 2009) foi desenvolvido dentro deste projeto para suprir as necessidades de um sistema de armazenamento largamente escalável e de grande vazão.

Assim como o GFS, o HDFS considera que o sistema de armazenamento é inerentemente instável. Desta forma, os dados do HDFS são sempre replicados em diversos servidores para garantir tolerância a falhas.

O HDFS é desenhado para aplicações que gerenciam grandes quantidades de dados. Seu projeto prioriza alta vazão de dados a baixa latência: as aplicações devem, portanto, fazer acessos a grandes quantidades de dados em uma única vez para tirar proveito do sistema.

Em diversos aspectos, o HDFS é semelhante ao GFS. Sua arquitetura consiste em um servidor de meta-dados centralizado chamado NameNode e diversos servidores de dados, os DataNodes. Os dados são divididos em blocos que são distribuídos entre os DataNodes.

Os dados são replicados automaticamente em diversos DataNodes. Esta replicação leva em consideração a distribuição dos DataNodes em diferentes racks. As políticas empregadas visam maximizar a vazão e ao mesmo tempo evitar perda de dados por falhas tanto em discos ou DataNodes individuais quanto indisponibilidades de racks inteiros.

Diferentemente do GFS, o HDFS não permite a escrita de um arquivo por mais de um cliente. Assim, os mecanismos de consistência do NameNode são extremamente simplificados. Esta escolha de projeto é consistente com a idéia de arquivos *write-once-read-many*, onde os dados são gerados uma única vez, mas lidos diversas vezes em execuções consecutivas de uma aplicação paralela.

Uma vez que o HDFS é projetado para aplicações que acessam grandes quantidades de dados, o acesso a DataNodes distantes dos nós de execução pode causar grande congestionamento na rede e, conseqüentemente, baixo desempenho. Para amenizar este problema, o sistema aplica mecanismos de seleção de réplicas, dando preferência à utilização das réplicas mais próximas ou, se possível, executar na máquina onde os dados residem.

2.2.13 NFS versão 4

A versão 4 do protocolo NFS (NFSv4) (PAWLOWSKI et al., 2000) oferece inúmeras vantagens com relação às versões anteriores. Foram incorporadas primitivas para permitir a migração e replicação de arquivos entre servidores, travamento de blocos de dados, melhores mecanismos de segurança e novos mecanismos para permitir melhor controle de *caching* do lado do cliente.

Uma das principais alterações na versão 4 do protocolo NFS é a remoção de protocolos auxiliares. Enquanto as versões anteriores delegavam funções como montagem de compartilhamentos e controle de travas a servidores específicos, a versão 4 incorporou estas e outras funções no protocolo padrão. A segurança do protocolo também foi melhorada através da inclusão de um mecanismo extensível de autenticação, permitindo controle de autenticidade dos servidores e dos clientes.

Foi incluída uma operação especial ao protocolo para efetuar várias operações em conjunto. A operação *COMPOUND* permite diminuir o custo de transmissão de pequenas mensagens, tornando o NFSv4 mais adequado a redes de larga escala com atrasos de transmissão consideráveis. O custo de tratamento destas operações agregadas também é menor no servidor, uma vez que, diferentemente das versões anteriores, as operações podem ser tratadas seqüencialmente no mesmo processo.

As versões 1, 2 e 3 do NFS utilizam um protocolo sem estado no servidor. Desta forma, eventuais falhas no serviço não causariam falha nos clientes, uma vez que estes poderiam interpretar a falha do servidor simplesmente como um atraso maior no envio da resposta. O NFSv4 opta por abrir mão deste mecanismo simples de recuperação em prol da implementação de mecanismos para garantir atomicidade de operações e garantias de travas em porções de arquivos pelos clientes. Adicionalmente, a inclusão de uma operação específica para abertura de arquivos permite ao servidor delegar algumas operações ao cliente que a efetuou. Este mecanismo visa permitir *caching* agressivo de dados de arquivo e de meta-dados pelo cliente.

O NFSv4 permite que árvores inteiras do sistema de arquivos sejam migradas ou replicadas entre servidores, aumentando a disponibilidade dos arquivos. No caso da migração, quando um cliente acessa um arquivo que foi migrado de seu servidor original, é devolvido um código de erro especial. O cliente pode, então, consultar o atributo *FS_LOCATION* para descobrir que servidor contatar para continuar acessando os dados.

O atributo *FS_LOCATION* também pode ser utilizado para designar localizações alternativas do arquivo para acesso somente para leitura, permitindo a replicação de dados entre servidores. O cliente pode escolher acessar uma réplica em outro servidor se considerar que o servidor original não está respondendo às requisições dentro de um limite de tempo máximo, por exemplo.

No NFSv4, no entanto, apenas um servidor pode gerenciar um arquivo que está sendo alterado por operações de escrita. Desta forma, acessos concorrentes para escrita em arquivo continuam sofrendo com o gargalo encontrado nas versões anteriores do protocolo.

2.2.14 pNFS

pNFS (GIBSON; CORBETT, 2004; GIBSON; WELSH; CORBETT, 2004; HILDEBRAND; HONEYMAN, 2005b) é uma extensão opcional do protocolo NFSv4. O objetivo desta extensão é prover aos clientes mecanismos para efetuar acesso direto aos dispositivos de armazenamento do sistema, evitando o *overhead* causado por protocolos de uso mais geral, como o caso do NFS.

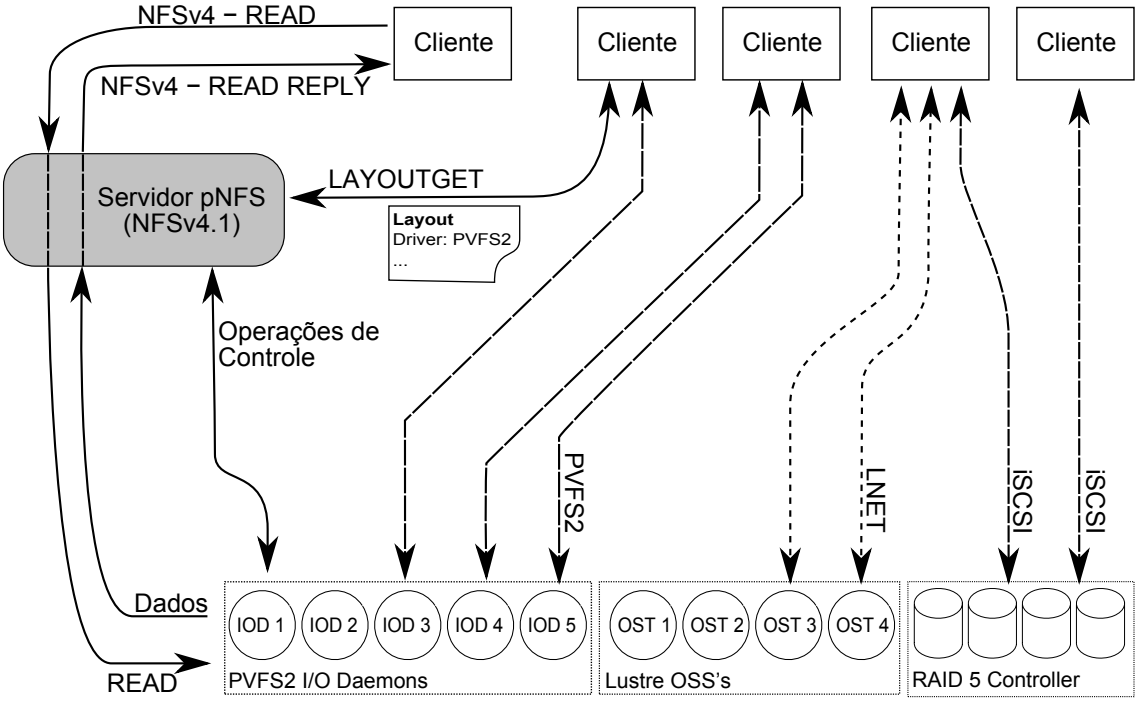


Figura 2.7: Arquitetura do pNFS

Os dispositivos de armazenamento do pNFS podem ser de diversos tipos: controladores RAID acessados via protocolos de bloco como iSCSI, dispositivos de armazenamento de objetos como os OSS's do sistema Lustre ou os servidores de dados do PVFS2. Para que os clientes acessem os arquivos nestes dispositivos, no entanto, eles devem possuir os *drivers* e bibliotecas necessárias.

A Figura 2.7 ilustra a arquitetura do pNFS. Os clientes pNFS comunicam com um servidor para obter informações sobre os meta-dados de um arquivo. Uma vez que os clientes possuam as informações de como acessar os dados, as operações de leitura e escrita são enviadas diretamente aos dispositivos de I/O.

Para suportar este acesso direto, foi introduzida a chamada *LAYOUTGET*. O *layout* retornado pelo servidor é uma estrutura que descreve para o cliente o tipo de protocolo a ser utilizado para acessar os dados, além de informações específicas deste protocolo (como mapeamento de objetos para OSD's, tamanho dos objetos, etc.). Os clientes utilizam um *layout driver* responsável por fazer a tradução das operações de leitura e escrita das aplicações em operações específicas do protocolo utilizado. Desta forma, é necessário possuir um *layout driver* específico para cada protocolo de dados que será utilizado.

Um mesmo cliente pode suportar diversos protocolos de acesso direto – desde que possua os *drivers* adequados – e acessar arquivos via diferentes protocolos de forma concorrente. No entanto, um mesmo arquivo deve residir em um único tipo de armazenamento, não sendo possível, por exemplo, dividí-lo entre controladores iSCSI e OSD's.

Na ausência do *driver* específico que permita fazer o acesso direto, o cliente utiliza as chamadas tradicionais de READ e WRITE do protocolo NFSv4. Desta forma, clientes que não implementam pNFS podem acessar ao sistema de armazenamento, ainda que com menor desempenho.

2.2.15 Split-Server NFS

Ambientes de grade requerem acesso global a grandes repositórios de dados. Enquanto sistemas de arquivos paralelos oferecem alto desempenho em uma rede local, ambientes de grade exigem integração sem interferências, segurança e bom desempenho.

O Split-Server NFS (HILDEBRAND; HONEYMAN, 2005a) propõe modificações no protocolo NFSv4 para permitir tal integração e garantir um bom desempenho nas transferências entre diferentes *sites*. Estas modificações permitem que um mesmo arquivo seja gerenciado por mais de um servidor NFSv4 ao mesmo tempo, permitindo acessos paralelos. O armazenamento dos dados, assim como no pNFS, é feito utilizando sistemas de arquivos paralelos tradicionais.

Este sistema divide a funcionalidade do servidor NFSv4 em dois tipos de servidores. Os **servidores de estado** são responsáveis por manter informações de estado dos arquivos – meta-dados – e tratar as operações sobre estes. Os **servidores de dados** são responsáveis por executar as operações de leitura, escrita e confirmação (*commit*). Estes servidores operam diretamente sobre o sistema de armazenamento utilizado, permitindo obter vazões semelhantes às do sistema de arquivos exportado.

Foi adicionada também uma diretiva *FILE_LOCATION* para indicar quais servidores podem operar sobre um mesmo arquivo. Cada *file location* possui indicação se a localização é apenas para leitura. Esta informação é utilizada para permitir acesso paralelo de leitura e diminuir a necessidade de sincronizações devido a alterações nos meta-dados do arquivo em questão.

O Split-Server NFS possui objetivos e arquitetura semelhantes à do dNFSp. Ambos utilizam NFS para transporte de dados, enquanto utilizam técnicas mais otimizadas para o armazenamento destes. A principal diferença entre estes dois sistemas está na extensão da diretiva *FILE_LOCATION*: enquanto um cliente do dNFSp pode acessar apenas um meta-servidor, os clientes do Split-Server NFS podem ser redirecionados para outros servidores, permitindo um melhor balanceamento de carga.

2.2.16 NFS-CD

Enquanto sistemas de arquivos paralelos objetivam fornecer acesso rápido a dados armazenados em um conjunto bem definido de servidores, o NFS-CD (NFS with Cluster Delegation) (BATSAKIS; BURNS, 2005) visa aumentar o desempenho no acesso aos dados permitindo o compartilhamento entre os próprios clientes. O NFS-CD estende o conceito de delegação presente no NFSv4 para permitir que um arquivo seja delegado a um conjunto de clientes (*cluster delegation*). Estes clientes trocam entre si esta delegação para permitir compartilhamento em leitura e escrita de dados sem necessidade de contatar o servidor.

O NFS-CD também estende o modelo do NFSv4 permitindo que os clientes de uma *cluster delegation* efetuem operações de *COMMIT* sem contatar o servidor. Ao invés de gravar no dispositivo de armazenamento permanente no servidor, os clientes copiam os dados alterados para um determinado número de clientes dentro da delegação. Esta operação é chamada de *fast-commit*, pois permite que as alterações sejam gravadas para o dispositivo permanente assincronamente.

Utilizando estas técnicas, o NFS-CD permite acessos mais rápidos aos dados dos arquivos através da descentralização do acesso. Uma vez que os dados serão transmitidos para um conjunto de máquinas que operam sobre um arquivo, é possível utilizar este esquema para deixar os dados próximos aos clientes que mais os acessam. A replicação dos

blocos de dados em vários clientes também permite um certo grau de tolerância a falhas.

2.3 Comparativo entre os Sistemas de Armazenamento

A Tabela 2.1 apresenta, de forma resumida, as principais características observadas nos sistemas apresentados anteriormente. As seguintes características são apresentadas:

Método de Acesso ao sistema (protocolo, via driver do sistema operacional, etc.)

Tipo de Armazenamento empregado para armazenar os dados nos dispositivos de armazenamento.

Características Principais observadas no sistema

Características de Adaptabilidade disponibilizadas pelo sistema

Sistema	**Método de Acesso**	**Tipo de Armazenamento**	**Características Principais**	**Características de Adaptabilidade**
Harp	NFS	Redundante, cópias integrais distribuídas em diversos servidores	Transparência no acesso via protocolo NFS; Alta Disponibilidade	–
Bigfoot-NFS	NFS	Distribuído, arquivos inteiramente armazenados em servidores NFS	Ausência de servidor de meta-dados para não criar gargalo no sistema	–
Vesta	Protocolo Próprio	*Stripping*/*Round-Robin* entre diversos discos	Distribuição dos dados definida pela aplicação quando da criação do arquivo; Fortemente integrado à arquitetura Vulcan	–
Expand	NFS	*Stripping*/*Round-Robin* feito por uma biblioteca cliente; Dados armazenados em servidores NFS	Permite a utilização de *servidores* NFS sem modificação	–

Sistema	**Método de Acesso**	**Tipo de Armazenamento**	**Características Principais**	**Características de Adaptabilidade**
AdExpand	NFS	*Stripping*/*Round-Robin* feito por uma biblioteca cliente; Dados armazenados em servidores NFS	Permite a utilização de *servidores* NFS sem modificação	Distribuição dos *stripes* do arquivo definida durante sua criação, visando maximizar a utilização de espaço em disco
PVFS	Protocolo próprio através de módulo do *kernel* ou de biblioteca específica	*Stripping*/*Round-Robin* ; Granularidade definida pelo cliente durante a criação do arquivo	Acesso direto aos servidores de dados distribuídos, alta escalabilidade	Migração de arquivos entre servidores de dados para evitar congestionamento
Lustre	Protocolo próprio (LNET/RPC) através de módulo de *kernel*. Comunicação via diversas tecnologias de Rede	Objetos (*Round Robin*)	Gerenciamento centralizado de meta-dados; Divisão em objetos e tamanho de *stripe* definidos na criação do arquivo; Acesso direto aos servidores de dados; Integração com diversas tecnologias de Rede; Alta escalabilidade	Objetos podem ser movidos entre servidores se solicitado via ferramenta de controle pelo Administrador; Distribuição dos objetos pode ser guiada durante a criação dos arquivos de acordo com características da aplicação

Sistema	**Método de Acesso**	**Tipo de Armazenamento**	**Características Principais**	**Características de Adaptabilidade**
ClusterFile	Protocolo Próprio/Biblioteca	Distribuído de acordo com aplicação	Utiliza a divisão dos dados entre os clientes para guiar a distribuição do arquivo nos servidores disponíveis, na tentativa de aumentar a concorrência de acesso	Distribuição dos dados nos servidores acompanha aquela feita entre as instâncias da aplicação
XtreemFS	Protocolo Próprio/Biblioteca	Objetos	Armazenamento de dados voltado para infraestruturas de grade	Replicação automática de dados conforme a distribuição da aplicação nos *sites* e nós de processamento
CEPH	Protocolo Próprio via biblioteca de acesso ou módulo de *kernel*	Objetos	Sistema de armazenamento para Cluster voltado para escala de *petabytes*	Serviço de metadados com divisão de dinâmica de carga; Redistribuição de dados forçada na ocorrência de falhas ou inserção de novos servidores; Localização de Objetos algorítmica
Google File System	Protocolo Próprio via biblioteca de acesso	*Stripping* com replicação	Sistema de arquivos voltado para aplicações específicas	Replicação automática ; Co-alocação de I/O e processamento
HDFS	Protocolo Próprio via biblioteca de acesso	*Stripping* com replicação	Sistema de arquivos para aplicações altamente distribuídas, *write-once, read many*	Co-alocação de I/O e processamento; Posicionamento de processos em nós próximos aos dados

Sistema	Método de Acesso	Tipo de Armazenamento	Características Principais	Características de Adaptabilidade
pNFS	NFSv4 com extensões	Sistemas de armazenamento variados	Extensão do protocolo NFS que permite acessar diretamente os dispositivos de armazenamento de dados	Dependente do sistema de armazenamento utilizado
Split-Server NFS	NFSv4 com extensões	Sistemas de armazenamento variados	Extensão do protocolo NFS para permitir diversas réplicas de um mesmo arquivo	Dados disponíveis em diversos servidores
NFS-CD	NFSv4 com extensões	Discos locais e Memória	Permite aos clientes gerenciar se tornar responsáveis pelos blocos de dados que lêem ou escrevem	Dados são distribuídos aos clientes e gerenciados por estes conforme seus padrões de acesso

Tabela 2.1: Tabela Comparativa entre os Sistemas de Armazenamento apresentados

2.4 Considerações sobre Sistemas de I/O

Sistemas de entrada e saída são uma parte crucial para sistemas que demandam grandes quantidades de dados – principalmente sistemas de alto desempenho. A grande quantidade de clientes presentes em clusters de alto desempenho e o aumento da complexidade das aplicações impõem uma grande demanda para os sistemas de armazenamento.

Para lidar com este problema, diversas soluções já foram desenvolvidas com diferentes propósitos e estratégias. O estudo aqui apresentado inclui algumas soluções presentes na bibliografia e ressalta algumas de suas características.

Dentre as tendências mais importantes observadas, pode-se destacar a preocupação com a adaptação do serviço de I/O às necessidades das aplicações que o utilizam. Esta adaptação pode ser observada em algumas características ressaltadas na Tabela 2.1: a distribuição da aplicação com base nos dados acessados por instância (GFS e HDFS), distribuição dos dados em função dos acessos da aplicação (ClusterFile e NFS-CD), divisão dos dados em *stripes* definida pela aplicação (Vesta, Lustre, PVFS2), replicação e redistribuição dinâmica dos dados (XtreemFS, Ceph, PVFS), dentre outras.

Outros trabalhos abordam temas correlatos a estas estratégias. (QIN et al., 2003) avaliaram a migração de processos dentro de um cluster para melhorar o balanceamento de carga de I/O da aplicação, utilizando também métricas para memória e processamento. Já (LIU et al., 2000) estudaram a migração dos dados armazenados em nós de I/O de métricas de congestionamento.

Assim, fica clara a relevância de mecanismos de dinamismo de dados para sistemas de I/O distribuído. Estas estratégias permitem evitar sobrecarga nos recursos de armazenamento, permitindo uma maior escalabilidade para o sistema de arquivos.

As estratégias observadas nos trabalhos estudados visam melhorar o desempenho de acesso a um conjunto fixo de servidores de dados. Desta forma, a capacidade de I/O do sistema será limitada pela quantidade de recursos disponibilizados pelo administrador

quando da configuração do sistema.

Nem sempre é possível ao administrador conhecer as características de I/O de todas as aplicações executadas em um cluster, tampouco prever o comportamento do desempenho do sistema de armazenamento quando houver a execução concorrente de várias aplicações. Desta forma, o dimensionamento fixo do sistema de arquivos distribuído é, comumente, inadequado, provendo um desempenho aquém do necessário às aplicações.

3 DNFSP

O sistema de arquivos dNFSp é um PFS que tem por objetivo prover uma área de armazenamento de grande capacidade e com alto desempenho nas transferências de dados, ao mesmo tempo que mantém compatibilidade com o protocolo NFS padrão. A utilização deste protocolo permite a comunicação do dNFSp com clientes nas mais variadas arquiteturas e implementações de sistemas operacionais devido à grande disseminação do protocolo NFS. Este capítulo apresenta a arquitetura do sistema e ilustra seu funcionamento.

3.1 Histórico

3.1.1 *Network File System*

O *Network File System* (NFS) é um sistema de arquivos distribuído originalmente desenvolvido pela Sun Microsystems em 1984. Seu objetivo era tornar acessível a um conjunto de clientes os dados armazenados em um único servidor, utilizando a mesma abstração de arquivos e diretórios já presente nos sistemas UNIX de então.

O NFS funciona sobre um mecanismo de RPC (*Remote Procedure Call*), também desenvolvido pela Sun. Assim, as operações sobre arquivos, tais como leitura, escrita, consulta de atributos, conteúdo de diretórios, criação e remoção de arquivos, etc. são implementadas como funções remotas que podem ser invocadas por um cliente – e.g. a camada *vnode* de um sistema Unix, permitindo a utilização do NFS de forma idêntica a um sistema de arquivos normal.

A utilização de um mecanismo de RPC padronizado também tem por vantagem a utilização de formatos de dados independentes de arquiteturas e implementações de sistemas operacionais. Isto permitiu a implementação de clientes e servidores NFS intercomunicáveis em diversas implementações de sistemas Unix, sistemas Windows e Apple.

O NFS tornou-se um sistema de arquivos distribuído bastante difundido e uma opção simples e eficaz para sistemas de *Cluster Beowulf*: os servidores e bibliotecas necessários para sua utilização estão presentes na maioria das distribuições Linux e sua configuração é bastante simples. Devido às necessidades de desempenho das aplicações típicas deste tipo de sistema, o NFS logo começou a mostrar problemas devido à sua natureza centralizada, tornando-se um gargalo para as operações de entrada e saída de dados.

3.1.2 NFSp

O armazenamento centralizado dos sistemas NFS é um dos seus principais gargalos. Com a utilização de diversos nós de processamento efetuando leituras e escritas de grandes quantidades de dados no sistema de arquivos compartilhado, tanto a vazão de rede

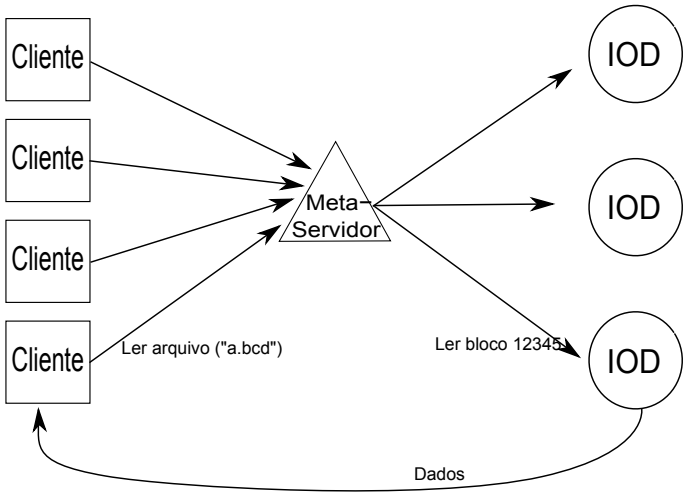


Figura 3.1: Arquitetura do NFSp

quanto a de disco atingem rapidamente seus limites. Para amenizar o problema da vazão de dados no protocolo NFS, foi proposto o sistema de arquivos NFSp.

O NFSp (LOMBARD; DENNEULIN, 2002) é um sistema de arquivos paralelo cujo principal objetivo é fornecer um acesso eficiente aos dados do sistema através do protocolo NFS, não exigindo alterações nos clientes do sistema. A Figura 3.1 apresenta a arquitetura do sistema NFSp. Um servidor central responsável pelos meta-dados – o meta-servidor – é representado como um triângulo. Este servidor é acessado pelos clientes através do protocolo NFSv2. Os IOD's (I/O Daemons), responsáveis pelo armazenamento dos dados, são representados na figura por círculos. Estes recebem do meta-servidor requisições de leitura e escrita e respondem diretamente aos clientes. Os dados são divididos entre os IOD's com uma distribuição cíclica de blocos.

Essa divisão permite que as requisições de leitura de um cliente sejam divididas entre os IOD's, atendidas em paralelo e respondidas diretamente aos clientes requisitantes. Esta resposta direta pode ser feita sem alteração no protocolo NFS através de *spoofing*[1]. Desta forma, as operações de leituras agregam vazão de disco e rede de vários nós de armazenamento. As operações de escrita, no entanto, sofrem com o gargalo no meta-servidor único, uma vez que estas constituem mensagens grandes e acabam sobrecarregando a capacidade de banda e processamento do único meta-servidor.

3.2 Distributed NFSp – dNFSp

O dNFSp (ÁVILA, 2005) – Distributed NFSp – é um sistema de arquivos paralelos baseado no NFSp, mantendo também a compatibilidade com o protocolo NFS. No dNFSp, no entanto, a carga antes associada a um único meta-servidor é dividida entre várias máquinas, através da distribuição do tratamento de meta-dados. Desta forma, é possível escalar o sistema também para operações de escrita, ao preço de eventuais sincronizações entre os meta-servidores para garantir acesso concorrente a um mesmo arquivo (KASSICK et al., 2005).

[1] *Spoofing*: Geração de pacotes IP com endereço de origem falsificado. Neste caso, permite aos servidores de dados responder diretamente aos clientes como se fossem o meta-servidor – o destinatário da requisição original.

A Figura 3.2 apresenta a arquitetura do dNFSp. No exemplo, seis clientes estão associados a três meta-servidores. Os dados são armazenados por cinco IOD's, agrupados em cinco VIOD's (explicados na Seção 3.3). Como o protocolo utilizado nos clientes ainda é o NFS padrão, eles desconhecem a distribuição de dados e acessam somente seus respectivos meta-servidores.

Os meta-servidores se comunicam para manter os meta-dados sincronizados e acessam os dados gravados nos IOD's configurados no sistema. Assim como no NFSp, o IOD designado para responder a requisição responde diretamente ao cliente, evitando consumir recursos dos meta-servidores novamente.

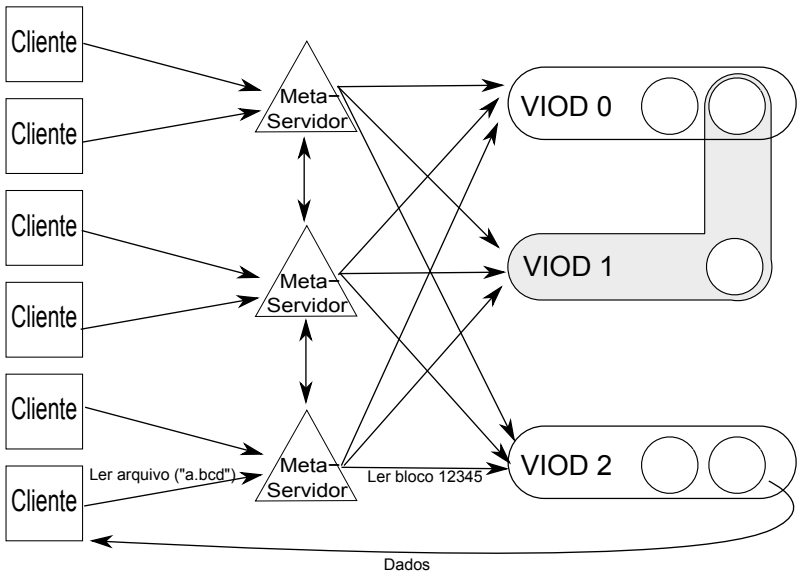


Figura 3.2: Arquitetura do dNFSp

A divisão de dados fixa entre um conjunto pré-determinado de IOD's força a configuração de um sistema estático durante a criação do sistema de arquivos. Isto limita a liberdade de um administrador para redistribuir a carga dos servidores conforme as necessidades das aplicações em execução no ambiente distribuído.

3.3 VIOD's e Dinamismo

Para permitir um certo grau de dinamismo nos servidores de armazenamento, o dNFSp utiliza a abstração de *Virtual IOD's* (VIOD's). Esta abstração permite aos meta-servidores a divisão do arquivo sem a predeterminação do servidor onde os dados serão armazenados.

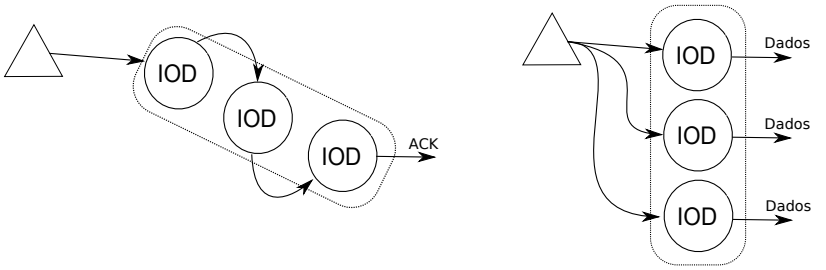
O fracionamento de arquivos é feito entre uma quantidade pré-definida de VIOD's. Uma tabela é mantida junto aos meta-servidores informando que IOD's atendem às requisições de quais VIOD's.

A implementação atual do dNFSp permite a utilização de vários IOD's por VIOD, efetuando replicação de dados. A replicação é feita por um processo de *store-and-forward*: o meta-servidor envia o bloco a ser escrito para um IOD do VIOD que irá gravar os dados em seu disco e, depois, enviar ao próximo servidor. Este processo é repetido até que todos os servidores possuam cópias dos dados. O último servidor a escrever a sua cópia dos dados envia a mensagem de confirmação para os clientes. Este processo pode ser visto na Figura 3.3(a).

Como a escrita garante a disponibilidade dos dados em todos os IOD's de um VIOD, qualquer um deles poderá responder às subseqüentes operações de leitura. Desta forma, o meta-servidor, ao receber operações de leitura, as envia uma para cada IOD, na tentativa de aumentar o desempenho de leitura. A leitura no caso de um VIOD com replicação é ilustrada na figura 3.3(b).

O dNFSp também permite que um IOD atenda às requisições de mais de um VIOD. Este caso é representado na Figura 3.2, onde um mesmo IOD responde às requisições dos VIOD's 0 e 1. Esta estratégia visa permitir o espelhamento em um número mínimo de IOD's para fins de tolerância a falhas. Deve-se observar que uma configuração onde um dos servidores de dados é associado a mais VIOD's que os outros terá um limitante em desempenho neste servidor. Isto se deve à necessidade de tratamento de um maior número de requisições devido à sobrecarga de VIOD's.

A sobrecarga de VIOD's em IOD's também permite manter o sistema expansível: utilizando um grande número de VIOD's distribuídos um número menor de IOD's, é possível expandir o sistema conforme as necessidades de I/O das aplicações em execução. Este dinamismo permite a adaptação do sistema de armazenamento às necessidades das aplicações em execução no cluster, uma vez que é possível modificar o conjunto de servidores durante a utilização do dNFSp.



(a) Escrita (b) Leitura

Figura 3.3: Processos de Escrita e Leitura em VIOD com Replicação

3.4 Ferramenta de Gerenciamento – *IOD Management*

A manutenção da tabela de VIOD's, bem como o monitoramento dos IOD's ativos no sistema é feita pela ferramenta *iod_mgmt*. Esta ferramenta consiste em um *daemon* executado junto aos servidores de meta-dados. Através dela, são recebidos comandos para a modificação do sistema de armazenamento.

Uma das instâncias do serviço é definida como a instância mestre durante a inicialização do sistema. É responsabilidade do mestre receber os comandos de reconfiguração do sistema e repassá-los para os *daemons* em execução nos outros meta-servidores. Cada *iod_mgmt* se comunica com seu meta-servidor através de um *socket* UNIX, refletindo nas configurações do meta-servidor a nova situação do sistema.

É também de responsabilidade do *iod_mgmt* mestre verificar o funcionamento dos servidores de dados. Periodicamente, são enviadas mensagens de *ping* para os IOD's a fim de verificar sua disponibilidade. Quando um IOD deixa de responder um número pré-determinado de vezes, assume-se que ele encontra-se indisponível e os meta-servidores são instruídos a descartá-lo de futuras requisições.

O iod_mgmt é utilizado para registrar, junto ao sistema de arquivos, os nós pertencentes a uma mesma aplicação. Esta associação permite identificar acessos de uma mesma aplicação e, desta forma, efetuar reconfigurações de I/O específicas para cada uma delas.

3.5 Servidores Exclusivos para Aplicação

Algumas aplicações podem demandar uma grande capacidade de armazenamento e alta vazão na transferência de dados do sistema de arquivos, interferindo com o desempenho de outras com menores requisitos. Com a utilização do dinamismo de servidores, torna-se possível adaptar o PFS para atender a tais aplicações de forma diferenciada, diminuindo seu impacto no sistema como um todo.

Uma funcionalidade já estudada no dNFSp é a utilização de servidores exclusivos por aplicações. Neste tipo de configuração, um conjunto de nós tem acesso a todo o sistema de arquivos compartilhado através do dNFSp. Os dados escritos por tais nós, no entanto, serão armazenados em servidores previamente designados para atender exclusivamente às requisições destes clientes (HERMANN et al., 2006). Tais servidores são chamados *Servidores de Aplicação* (*Application IOD's, AppIOD's*).

Quando os clientes efetuam leituras sobre arquivos que os AppIOD's desconhecem (por não terem sido lá gravados), as requisições são encaminhadas para os IOD's permanentes[2], uma vez que o meta-servidor não mantém controle da localização de arquivos individuais. Isto torna os AppIOD's mais eficientes para aplicações dominadas por operações de leitura.

A utilização de AppIOD's é ilustrada na Figura 3.4. Um conjunto de nós que constitui uma aplicação A_0 escreve, no total, C_0 bytes de dados no sistema compartilhado. Durante a execução, os outros nós do cluster fazem acesso normal ao sistema de armazenamento. Enquanto ambos conjuntos de nós compartilham o serviço de meta-dados, o serviço de armazenamento é provido por conjuntos diferentes de servidores.

Após o término da aplicação, o sistema é instruído a fazer a cópia dos dados gerados pela aplicação para os servidores de dados permanentes. Neste momento, os IOD's receberão uma carga extra devido à sincronização.

A Figura 3.5 ilustra os três momentos distintos no uso de servidores de aplicação. Em um tempo t_0, é disparada uma aplicação A_0 que irá escrever C_0 bytes de dados em seus servidores exclusivos até o momento t_1. Neste mesmo período de tempo, os nós do sistema movimentaram C' bytes de dados nos servidores do sistema. Após o fim da aplicação em t_1, os servidores de aplicação necessitam sincronizar os dados gravados com os servidores do sistema. Assim, é necessário enviar para os servidores de dados C_0 bytes, enquanto os clientes de outras aplicações movimentam C'' bytes. Após o final da sincronização em t_2, os nós do sistema seguem utilizando o sistema de arquivos, movimentando C''' bytes de dados.

Conforme apresentado em (HERMANN et al., 2006), há um ganho significativo de desempenho tanto em aplicações para as quais existem IOD's exclusivos quanto nas quais utilizam os IOD's permanentes do sistema. A Figura 3.6 apresenta a banda passante para os quatro períodos de tempo na utilização de servidores exclusivos. O teste utilizou um sistema dNFSp com 9 servidores, cada um servindo como meta-servidor e IOD. Os 27 clientes foram divididos em 3 aplicações que escrevem dados no sistema compartilhado. Para a aplicação A_0, foram inseridos 9 AppIOD's após o instante t_1. Entre t_1 e t_2, a vazão

[2]Servidores Permanentes: Os IOD's definidos pelo administrador do sistema, em contraste àqueles inseridos pelo usuário como AppIOD's.

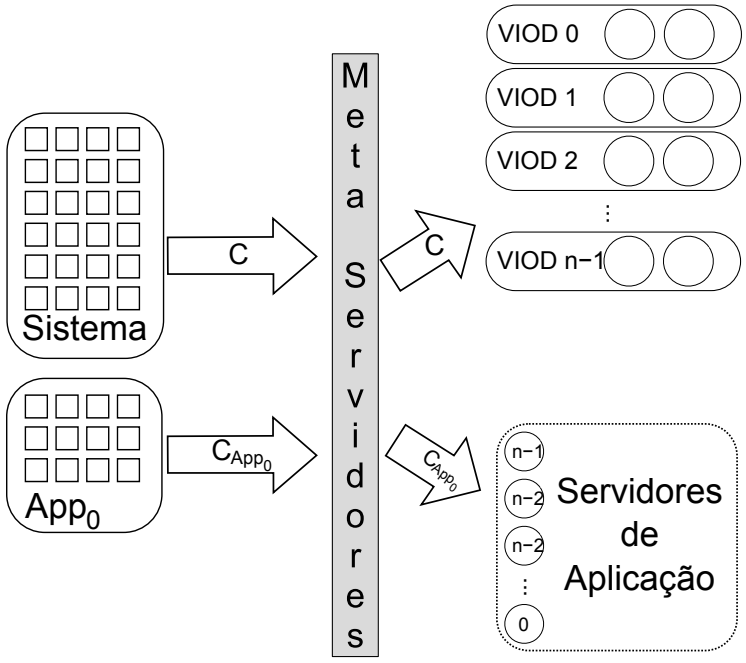


Figura 3.4: Utilização do dNFSp com servidores exclusivos de aplicação: O fluxo de dados de uma aplicação configurada é desviado para servidores de dados específicos.

da aplicação A_0 teve um ganho de aproximadamente 90%, enquanto as aplicações A_1 e A_2 apresentaram um ganho de 25%. Entre t_2 e t_3 é feita a sincronização dos AppIOD's com os servidores permanentes. Apesar da maior carga junto aos IOD's, as aplicações A_1 e A_2 apresentam um ganho em sua vazão, uma vez que os meta-servidores dispõem de maior vazão de dados com o fim de A_0.

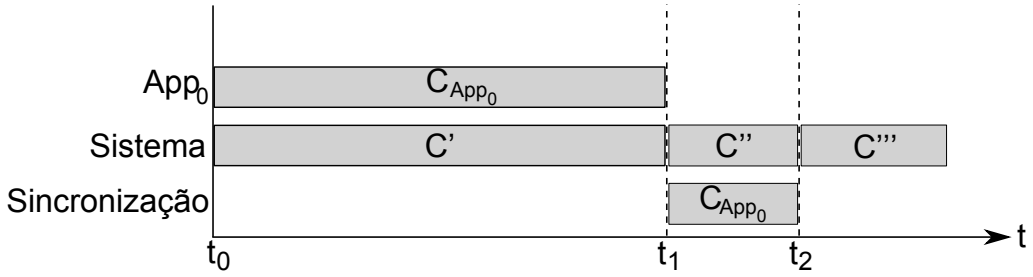


Figura 3.5: Execução de aplicação com Servidores de Aplicação

3.6 Considerações sobre Desempenho

O dNFSp é um PFS que procura oferecer alto desempenho no acesso a dados **sem requisitar alterações no lado dos clientes**. Para tanto, os servidores de meta-dados são utilizados como uma camada de abstração, recebendo as requisições dos clientes e as encaminhando aos servidores correspondentes. Assim, seu desempenho será limitado tanto pela capacidade de rede dos servidores de meta-dados quando da capacidade de rede e disco dos IOD's – permanentes ou exclusivos de uma aplicação.

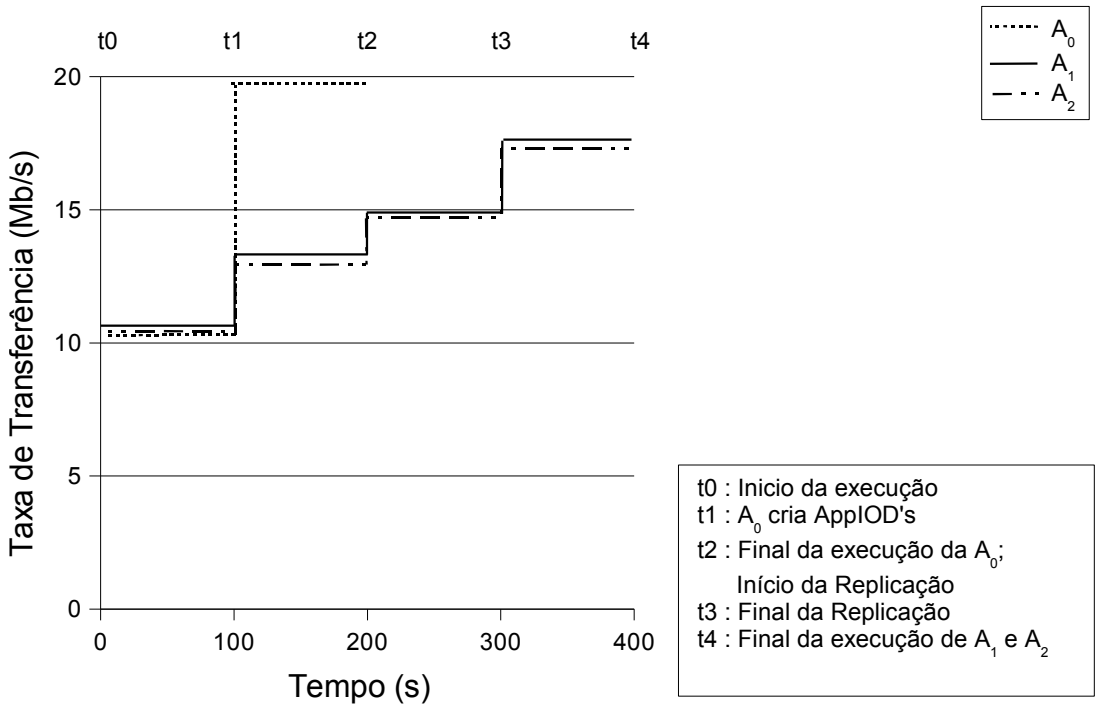


Figura 3.6: Impacto da utilização de AppIOD's

Neste sentido, há a necessidade de se considerar o quanto o desempenho do sistema como um todo será afetado pela disponibilidade de recursos em cada camada do dNFSp. Para analisar este desempenho, este trabalho leva em consideração a banda disponível em cada uma das camadas do sistema, o conjunto de clientes associado a cada conjunto de servidores e o tipo de requisição (leitura ou escrita) efetuada por estes clientes. A Figura 3.7 ilustra a disponibilidade de banda em cada camada do dNFSp estudada neste trabalho.

A banda disponível na camada dos meta-servidores é designada por B_{meta}. Uma vez que o dNFSp não provê dinamismo nesta camada, esta banda será constante durante a utilização do sistema. Concorrem pelo acesso aos meta-servidores todos os clientes do cluster: uma vez que o protocolo utilizado pelos clientes é o NFSv2, não há maneira de instruí-los sobre localizações alternativas dos arquivos, como no protocolo NFSv4. Assim, mesmo clientes que utilizam servidores de aplicação terão seus acessos tratados por este conjunto não exclusivo de servidores.

No nível dos servidores de dados, consideramos dois tipos de recursos que irão tratar as requisições dos clientes: os IOD's permanentes do sistema e os IOD's de aplicação. A capacidade destes conjuntos de servidores são designadas por B_{viod} e B_{App}, respectivamente.

Os IOD's de aplicação, durante a execução desta, necessitam tratar apenas as requisições dos nós pertencentes à aplicação em questão. Já os IOD's serão responsáveis pelas requisições dos nós restantes, pelas requisições de leitura dos nós de aplicação sobre dados que não foram gravados nos servidores de aplicação e pela eventual sincronização de dados dos servidores de aplicação após seu término.

Uma vez que todas as operações dos clientes devem ser primeiramente processadas

pelos meta-servidores, operações de escrita terão seu desempenho atrelado à banda B_{meta}: a utilização de servidores exclusivos e a quantidade de servidores permanentes irão diminuir o tempo de tratamento das requisições na camada de armazenamento. A transmissão das requisições de escrita, por outro lado, será limitada pela quantidade de meta-servidores acessada pelos clientes.

Já operações de leitura são influenciadas pela quantidade de servidores de dados disponíveis: devido aos tamanhos destas requisições, a banda dos meta-servidores é pouco utilizada em comparação com àquela dos servidores de dados, os quais necessitam enviar grandes quantidades de dados para os clientes.

Em casos de uso onde a capacidade de rede dos meta-servidores já se encontrar esgotada por requisições de escrita de outras aplicações, a utilização de AppIOD's não trará ganhos significativos. Em situações onde grande parte das requisições aos meta-servidores constitui-se de operações de leitura, no entanto, haverá banda disponível para que os clientes de uma aplicação se beneficiem de recursos exclusivos de I/O.

3.7 Conclusões

O sistema de arquivos paralelo dNFSp provê acesso a um conjunto distribuído de servidores de dados, agregando vazão de várias máquinas para um acesso com alto desempenho. A utilização do protocolo NFS permite a utilização de uma grande gama de clientes de diversas famílias de sistemas UNIX sem qualquer modificação no sistema operacional ou na aplicação.

Devido à utilização deste protocolo, é necessária a presença de uma camada de abstração do sistema de armazenamento distribuído – os meta-servidores. A ausência de mensagens de reconfiguração no protocolo NFSv2 exige que os clientes se comuniquem apenas com um meta-servidor determinado quando de sua inicialização. A reconfiguração dinâmica de servidores deve, portanto, se ater à camada de I/O do dNFSp.

Com a quantidade fixa de servidores de meta-dados, a capacidade de rede destes será um limitante na capacidade de I/O que será possível extrair com eventuais reconfigurações do sistema de I/O. Este limite de rede, no entanto, afeta de forma diferente as operações de leitura e escrita. A reconfiguração da camada IOD, como utilização de servidores exclusivos por aplicação, deve levar em conta o tipo das operações efetuadas sobre o sistema de arquivos, de forma a evitar reconfigurações que acabem por acarretar mais sobrecusto do que ganho em desempenho.

Uma vez que clusters podem ter diversas aplicações em execução concorrente – porém compartilhando um mesmo espaço de armazenamento – o tipo de operação dominante em cada uma delas pode ser distinto. Com conhecimento a respeito das operações de I/O destas aplicações, é possível efetuar uma reconfiguração mais eficaz, aumentando o desempenho para todas as aplicações em execução.

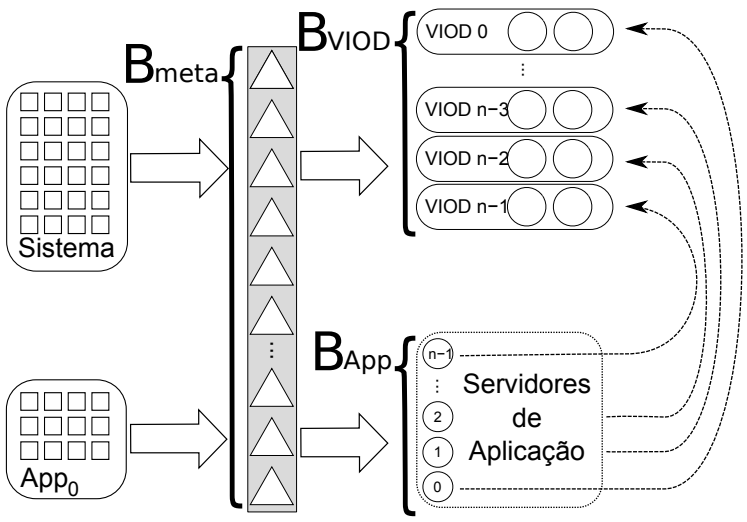


Figura 3.7: Bandas disponíveis nos diferentes elementos da arquitetura do dNFSp

4 PADRÕES DE ACESSO DE APLICAÇÕES PARALELAS

Enquanto a visão de dados de um arquivo como uma seqüência não estruturada de *bytes* é interessante sob o ponto de vista de armazenamento, aplicações paralelas utilizam tipos de dados complexos para representar as estruturas sobre as quais trabalham. Estas estruturas podem ser, por exemplo, matrizes multidimensionais de algum tipo composto (como tuplas de números em ponto flutuante), onde cada instância da aplicação paralela é responsável por efetuar operações sobre um subconjunto desta estrutura e armazenar o resultado – também de forma estruturada – em um arquivo de saída ou de troca com outras instâncias da aplicação.

A gravação destes dados de forma bem estruturada em um arquivo compartilhado, além de ser essencial para a posterior análise dos resultados, permite que aplicações que necessitem comunicar grandes volumes de dados entre suas instâncias utilizem o sistema de arquivos para tal fim. O conhecimento da estrutura do arquivo por todos os nós da aplicação permite que um nó qualquer saiba exatamente em qual posição do arquivo os dados processados por seus vizinhos foram gravados.

A estrutura da aplicação também irá refletir os momentos em que acessos ao sistema de armazenamento são gerados. Estes acessos podem estar concentrados apenas no início e fim da aplicação, refletindo sua entrada e saída de dados, ou esparsos ao longo da execução, conforme novos dados são gerados ou novos dados de entrada são necessários.

À maneira como as aplicações efetuam estes acessos ao sistema de arquivos é dado o nome de *padrão de acesso* da aplicação paralela. Com o conhecimento dos padrões de acesso de uma aplicação, é possível otimizar o sistema de armazenamento para prover uma distribuição de dados que melhor se adapte à maneira como a aplicação irá tratá-los. Este capítulo mostra alguns padrões de acesso comumente encontrados em aplicações científicas para ambientes de cluster.

4.1 Padrões de Acesso

O padrão de acesso de uma aplicação pode ser definido como o histórico de acessos feitos ao sistema de armazenamento ao longo de sua execução. Este histórico de acesso pode ser definido tanto pelos momentos em que os dados são acessados quanto pelas posições do arquivo acessadas pelas instâncias das aplicações.

Aplicações de alto desempenho possuem características próprias quanto à maneira como acessam os dados armazenados em sistemas de arquivos compartilhados. Diversas instâncias de um programa paralelo necessitam acessar um conjunto compartilhado de dados de forma coordenada, freqüentemente constituindo uma seqüência de operações bem característica da maneira como os dados são armazenados e tratados pela aplicação.

À seqüência de acessos passados e futuros feita por uma aplicação ao sistema de

arquivos é dado o nome de "padrão de acesso". Quando este padrão de acesso é conhecido ou previsível, é possível utilizar este conhecimento para otimizar a maneira como são feitos tais acessos, melhorando o desempenho no sistema de armazenamento[1].

Há dois principais tipos de observação dos padrões de acesso de uma aplicação: *espacial* e *temporal*. A observação espacial consiste em analisar que posições de um arquivo compartilhado (em termos de *offsets* em *bytes*) as instâncias de uma aplicação paralela acessam. Já a observação temporal consiste em analisar em que momentos a aplicação efetua os acessos e o tempo necessário para que o sistema de armazenamento os atenda.

A análise do padrão de acesso é feita a partir da seqüência de requisições de escrita e leitura efetuadas pelas instâncias de uma aplicação. Com as informações de posição e tamanho de cada operação, é possível definir quais porções do arquivo são acessadas por quais instâncias da aplicação; observando o tempo de chegada destas requisições nos servidores de dados, é possível identificar a quantidade de acessos feitos em diferentes momentos da execução.

Além do tipo de operação, é interessante observar a uniformidade das operações enviadas aos servidores. Esta uniformidade diz respeito tanto ao tipo de operações quanto ao tamanho das requisições. Isto permite identificar, por exemplo, fases de leitura ou escrita de uma aplicação, bem como os momentos em que a leitura é feita de forma massiva quanto quando é feita em grão fino.

É necessário também distinguir o ponto de observação dos acessos de uma aplicação. Esta observação pode ser feita sob dois ângulos distintos: globalmente ao sistema de arquivos compartilhado ou localmente, sob o ponto de vista das instâncias individuais da aplicação.

Um exemplo da diferença dessas duas visões é quando uma aplicação divide um arquivo de dados em porções iguais entre suas instâncias. A visão local do padrão de acesso espacial, neste caso, seria puramente seqüencial: cada cliente abre um arquivo e o lê ou grava, a partir de um ponto arbitrário, uma porção seqüencial do arquivo. A visão global, no entanto, permite observar que o arquivo foi lido inteiramente, através de leituras seqüenciais vindas de diferentes instâncias da aplicação.

No sistema de arquivos dNFSp, não é possível efetuar uma análise local aos clientes sem interferir no protocolo NFS ou na aplicação. Assim, o ponto de observação deve ser junto aos meta-servidores, oferecendo uma visão quase global: um único meta-servidor pode observar os acessos de todos os seus clientes, mas para observar os acessos de toda uma aplicação pode ser necessária a agregação de informações proveniente dos outros servidores.

A classificação de aplicações apresentada na Seção 4.4 é baseada na visão global dos acessos de cada aplicação. São consideradas as características temporais e espaciais de seu comportamento. Desta forma, é possível estudar suas necessidades de desempenho de I/O, bem como o tipo de operação mais significativo para o seu desempenho.

4.2 Análise Espacial

O padrão espacial de acessos dos dados reflete diretamente a maneira como estes são organizados pela aplicação e como são acessados durante sua execução. A análise do padrão de acesso é feita em termos de seqüência de tuplas do tipo $< offset, size, read/write >$.

A principal característica que este tipo de informação permite observar é a *seqüencialidade do acesso*, ou seja, a tendência de a aplicação ler seqüências contíguas de dados

[1] Melhor distribuição de requisições nos servidores, *caching*, *write-back*, etc.

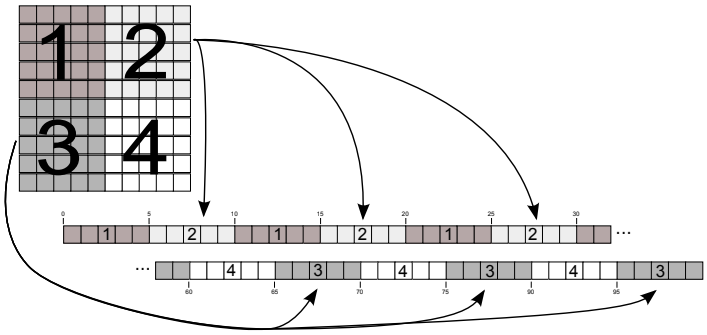


Figura 4.1: Padrão de Acesso Espaçado no Acesso a uma Matriz

que representam as porções de interesse de suas estruturas de dados.

A Figura 4.1 ilustra a divisão de uma matriz bidimensional de $10\,x\,10$ elementos entre quatro instâncias de uma aplicação qualquer. Através dos acessos feitos por um único cliente, é possível identificar operações de leitura ou escrita sobre porções contíguas do arquivo constantemente espaçadas.

4.3 Análise Temporal

A análise temporal do padrão de acesso de uma aplicação paralela tem por objetivo estudar o comportamento do sistema em termos de quantidade e tipo de operações executadas ao longo de sua execução. O principal interesse da análise temporal é identificar a carga que uma aplicação paralela gera no sistema de armazenamento durante sua execução.

Esta análise permite identificar se uma aplicação necessita constantemente do sistema de armazenamento para sua execução ou se a utilização deste é sazonal. Aliando esta análise temporal à observação dos tipos de operações e da uniformidade de tamanhos, é possível ter uma noção do impacto que a aplicação terá no sistema de armazenamento.

A Figura 4.2 mostra os acessos feitos por 5 clientes do benchmark MPI-IO-21. Pela figura, podemos observar que as requisições dos clientes chegam na camada meta-servidor em rajadas separadas por longos intervalos de tempo.

4.4 Taxonomia de Aplicações Paralelas de Acordo com seus Padrões de Acesso

Aplicações paralelas podem ser classificadas de acordo com seus aspectos temporais e espaciais de seus padrões de acesso. Para referência, utilizamos padrões de acessos observados na bibliografia (SMIRNI; REED, 1997; MADHYASTHA; REED, 1996; TRAN; REED, 2001; SEELAM et al., 2008; WANG et al., 2004) e em aplicações comumente utilizadas para medir desempenho de sistemas de arquivos paralelos (PARALLEL I/O BENCHMARKING CONSORTIUM, 2009; OLDFIELD, 2009; WONG; WIJNGAART, 2003).

Levando em consideração o padrão de acesso temporal, as aplicações encontradas na bibliografia podem ser classificadas nas seguintes categorias:

I/O-Bound : Este tipo de aplicação se caracteriza por ter uma taxa constante de acessos ao sistema de armazenamento ao longo de sua execução.

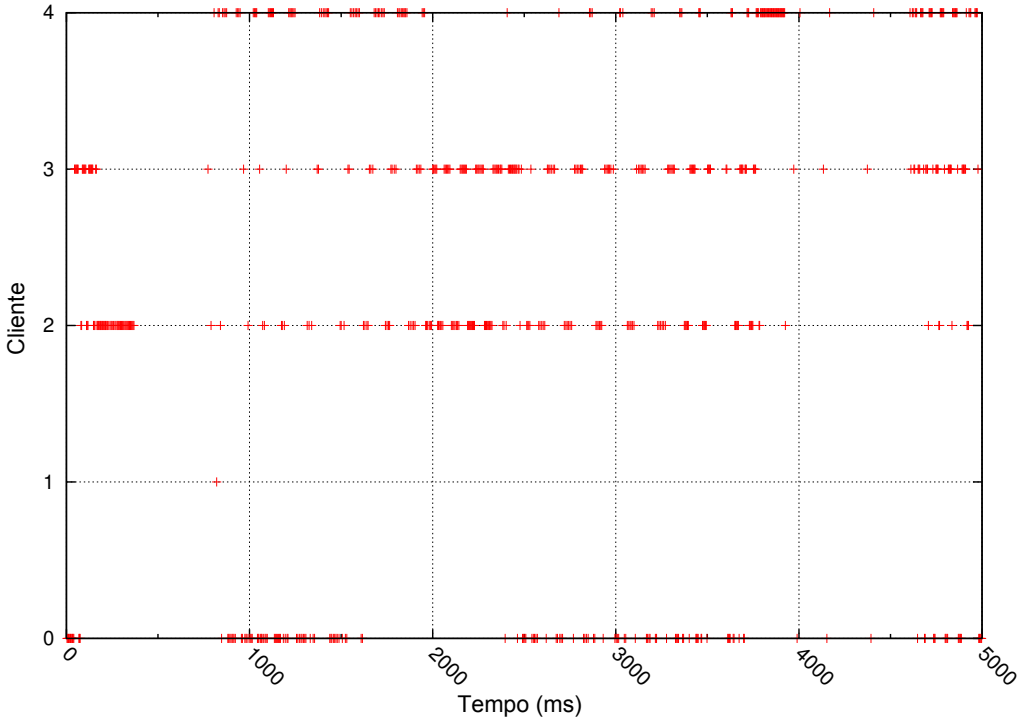


Figura 4.2: Acessos de diferentes clientes em um trecho da execução do benchmark MPI-IO-21

Esta classe representa o caso extremo de acesso ao sistema de arquivos: dados são gerados pelos clientes e enviados ao PFS de forma a utilizar toda a banda de rede e disco disponíveis nos servidores.

Esta classe serve de referência para comparação de desempenho e pode ser caracterizada por qualquer benchmark que faça uso do PFS sem interrupção – como o MPI-IO-Test-21 e o núcleo de I/O do FLASH (FRYXELL et al., 2000). Algumas aplicações podem ser capazes de sobrepor eficientemente suas fases se I/O e processamento através de escritas em *background* (MA; ZHAO; LIU, 2009). Nestes casos, uma aplicação que teria um comportamento periódico acaba por apresentar um comportamento I/O Bound sob o ponto de vista do sistema de armazenamento.

Checkpointing I/O : Esta classe representa as aplicações que possuem fases perceptivelmente distintas de processamento e I/O. Durante a fase de I/O, os acessos ao sistema de arquivo são intensos e feitos por todos os clientes da aplicação. Durante a fase de processamento, poucos – ou nenhum – acessos são feitos pelos clientes ao sistema de arquivos, razão pela qual, do ponto de vista de I/O, esta representa uma fase de inatividade.

Nesta classe incluem-se as aplicações que utilizam arquivos de checkpoint. O benchmark paralelo BT-IO (WONG; WIJNGAART, 2003) e software de simulação de flashes astrofísicos termonucleares FLASH (FRYXELL et al., 2000) (em sua execução completa) se encaixam nesta categoria.

End-Of-Execution Output : Este tipo de aplicação se caracteriza por produzir dados de saída somente ao final de sua execução. Nesta categoria, as aplicações passam o

período inicial de sua execução apenas acessando dados de entrada. Próximo ao final de sua execução, inicia-se uma fase de escrita com acessos intensos ao PFS.

Este tipo de aplicação se assemelha às aplicações com acessos periódicos, uma vez que há claros períodos de acesso aos dados de entrada e um único período de saída. Elas se diferenciam, no entanto, no tipo de operação dominante: enquanto as execuções periódicas geram dados ao longo de toda sua execução, as aplicações desta categoria **lêem dados** durante toda a sua execução. Uma aplicação que exemplifica esta categoria é o MPI-Blast (DARLING; CAREY; FENG, 2003).

Out Of Core : Algumas aplicações precisam trabalhar com mais dados do que é possível manter nas memórias dos nós de processamento durante a execução da aplicação. Este tipo de aplicação necessitará, então, realizar operações *out-of-core* sobre seus dados: após processar um subconjunto dos dados, estes serão armazenados e novos dados serão lidos para continuar seu processamento. Assim, serão observadas fases de leitura e escrita intercaladas. O MPQC (JANSSEN et al., 2006) é um exemplo deste tipo de aplicação, bem como o MADCAP (BORRILL et al., 2007).

Estas classes de aplicação possuem diferentes características quanto aos períodos em que irão fazer uso intensivo do sistema de arquivos e aos tipos de operações que irão dominar tais períodos. Através da análise de seu desempenho quando executadas sobre o dNFSp, é possível identificar quais padrões de acesso são dependentes de cada um dos sub-sistemas deste PFS. O estudo da execução concorrente destas classes de aplicação também permite identificar as oportunidades a serem exploradas com a reconfiguração do sistema de armazenamento.

4.5 Desempenho no dNFSp

Como discutido na Seção 3.6, a utilização do protocolo NFS padrão no dNFSp exige o envio de todas as mensagens para o sistema de I/O através dos meta-servidores, para então serem repassadas aos IOD's. Esta dupla transmissão de mensagens é responsável por um menor desempenho nas operações de escrita em relação às de leitura devido ao tamanho das mensagens.

Como o custo desta dupla transmissão é proporcional ao tamanho das mensagens, as operações de leitura não são gravemente penalizadas: o aumento da vazão devido à distribuição dos servidores é capaz de compensar esta desvantagem. Já as mensagens de escrita têm um sobrecusto maior nesta dupla transmissão.

Estas operações, no entando, tendem a sobrecarregar partes diferentes do dNFSp. Operações de leitura possuem pouco impacto na utilização de banda de rede na camada de meta-servidores. O maior custo numa operação de leitura é o acesso ao disco e a transmissão dos dados para o cliente, operações levadas a cabo nos IOD's. A banda dos meta-servidores, por sua vez, estará sub-utilizada. Já operações de escrita possuem maior potencial de esgotar a banda disponível tanto nos meta-servidores quanto nos IOD's devido ao tamanho das mensagens de requisição.

Para as classes de aplicação apresentadas, podemos ressaltar as seguintes características quando executadas sob o sistema de arquivos dNFSp:

I/O Bound – Devido à grande taxa de envio de requisições, esta classe representa uma alta carga aos servidores de dados. No caso de as requisições serem de leitura, haverá pouca ocupação de banda junto aos meta-servidores.

Em caso de escrita, no entanto, tanto os meta-servidores quanto os servidores de dados terão alta carga de trabalho devido à retransmissão das requisições e à escrita em disco propriamente dita.

Checkpointing I/O – Como esta classe se caracteriza por fases alternadas de I/O e inatividade, durante a fase de I/O, o comportamento será similar à da classe I/O Bound. Durante as fases de inatividade, no entanto, a baixa (ou inexistente) atividade dos clientes não irá causar sobrecarga aos servidores de dados ou meta-servidores.

O custo de I/O desta classe de aplicação será proporcional à razão $\frac{Tempo\ gasto\ em\ I/O}{Tempo\ de\ Inatividade}$. Esta relação e suas implicações para a reconfiburação do sistema de I/O é descrita no suas conseqüências para o sistema de armazenamento são estudadas na Seção 5.4.

End-of-Execution Output – Como em grande parte da execução esta classe efetua poucos acessos ao sistema de armazenamento, este tipo de aplicação possui pouco potencial de sobrecarregar os servidores. Sua fase de geração de dados, porém, deve se comportar como uma aplicação I/O Bound, ocupando grande parte dos recursos dos recursos nos servidores.

Out of Core – Devido às fases intercaladas de leitura e escrita, esta classe de aplicação pode consumir recursos de todo o sistema de armazenamento.

A Tabela 4.1 resume o tipo de operação de I/O dominante para cada classe de aplicação durante sua execução. No sistema de arquivos dNFSp, as aplicações cujo tipo de acesso é caracterizado por operações de escrita geram um grande impacto no sistema de armazenamento devido à saturação da banda de rede junto aos meta-servidores. É, portanto, de grande importância estudar configurações que permitam melhorar o desempenho do sistema de armazenamento quando da presença deste tipo de aplicação e suas interações com outros tipos de acesso.

Classe de Aplicação	Escrita	Leitura
I/O-Bound (Escrita)	X	
I/O-Bound (Leitura)		X
Checkpointing I/O	X	
End-Of-Execution Output	X	
Out Of Core	X	X

Tabela 4.1: Classes de Aplicação e Tipo Dominante de Operações

5 MODELO DE RECONFIGURAÇÃO AUTOMÁTICA PARA O DNFSP

O dimensionamento de um sistema de arquivos distribuído para um cluster de alto desempenho requer um balanço delicado entre a utilização de recursos para I/O e disponibilidade de processadores para executar as aplicações distribuídas. A utilização de muitos recursos dedicados para o sistema de armazenamento resulta num grande desempenho que pode nunca ser necessário devido à escassez de recursos de processamento. A alocação de poucos recursos, no entanto, pode acarretar um grande custo às aplicações que demandam grande capacidade de I/O no sistema.

Este dimensionamento é ainda mais dificultado pela diversidade de aplicações que podem ser executadas concorrentemente no mesmo ambiente distribuído, cada qual com suas características específicas de I/O. Com o desempenho diverso de operações de leitura e escrita no dNFSp, criar uma configuração pré-determinada que atenda tanto às necessidades de aplicações dominadas por operações de leitura quanto por de escrita pode se tornar impossível.

A reconfiguração dinâmica do sistema de armazenamento permite adaptá-lo às necessidades das aplicações durante sua execução, possibilitando uma melhor adequação aos requisitos de desempenho e às capacidades do ambiente. Para que esta adaptação seja vantajosa, é necessário definir as situações nas quais o sistema de armazenamento pode se beneficiar de uma reconfiguração, bem como os tipos de reconfiguração que podem ser feitos.

Neste trabalho, o tipo de reconfiguração utilizada é a inserção de servidores exclusivos para aplicação (AppIOD). Esta estratégia permite desafogar servidores de dados permanentes do dNFSp com a utilização de um conjunto secundário de máquinas para o armazenamento de dados de uma aplicação. Com isso, o desempenho das aplicações em execução é aumentado junto com a quantidade de recursos de I/O disponíveis.

A utilização de AppIOD's possui maior vantagem quando as aplicações que os utilizam são dominadas por operações de escrita. Uma vez que operações de leitura já possuem um melhor desempenho que as de escrita no dNFSp, o modelo proposto tem por objetivo detectar a necessidade de reconfiguração e selecionar aplicações para utilização de servidores exclusivos.

Para fazer tal adaptação, é necessário entender o comportamento do subsistema de I/O durante a execução das aplicações alvo para selecionar aplicações cujo acesso ao sistema de arquivos seja dominado por escritas. Neste trabalho, a adaptação do sistema é feita levando em conta as classes de aplicações já apresentadas no Capítulo 4: classes de aplicações com tendência a gravar muitos dados são selecionadas para reconfiguração, permitindo um melhor desempenho de aplicações que efetuam muitas operações de

leitura.

5.1 Reconfiguração do Sistema de Armazenamento

Um sistema de arquivos paralelo para um cluster de alto desempenho constitui uma plataforma implantada pelos administradores utilizando servidores dedicados para tal tarefa. O dimensionamento em termos de espaço de armazenamento e velocidade de transmissão é feito com base nos conhecimentos destes administradores sobre o perfil das aplicações utilizadas no ambiente, as disponibilidades de recursos financeiros e até mesmo restrições de espaço físico.

Vários eventos podem tornar este sistema inadequado ou torná-lo sub-utilizado. Eventos de falha e manutenções, por exemplo, podem tornar o sistema indisponível ou forçá-lo a operar com desempenho abaixo do esperado. Falhas que tornem o hardware indisponível, no entanto, têm se tornado menos comuns e manutenções que tornam indisponíveis serviços essenciais como I/O são eventos geralmente programados com antecedência (BUISSON, 2006). Em função disso, mecanismos de reconfiguração apenas para contornar falhas podem mostrar-se desnecessários para muitos ambientes.

A execução de várias aplicações concorrentes em um mesmo cluster, no entanto, é uma situação costumeira. Quando estas aplicações necessitam do sistema de arquivos para acessar seus dados de entrada ou para armazenar os resultados de sua execução, o armazenamento compartilhado pode se tornar um ponto de contenção. A reconfiguração dinâmica deste sistema, portanto, permite adequar um sistema de armazenamento outrora sub-dimensionado para uma configuração que permita às aplicações serem concluídas dentro dos prazos solicitados junto ao escalonador.

É necessário garantir a disponibilidade dos dados das aplicações antes e após uma reconfiguração. Recursos dinamicamente alocados para o armazenamento devem, então, estar disponíveis para servir estes dados durante a execução das aplicações que o utilizam. Reconfigurar dinamicamente o conjunto de servidores permanentes, portanto, pode melhorar o desempenho do PFS, mas gera um custo associado a redistribuição dos dados entre os novos servidores, o que pode interferir com a disponibilidade de recursos para a execução de aplicações.

A reconfiguração do sistema de I/O com recursos temporários, como AppIOD's, permite a utilização de recursos outrora dedicados a processamento, aumentando o desempenho das aplicações enquanto necessário, sem exigir os que tais recursos sejam permanentemente alocados para I/O. Como os servidores permanentes são liberados da carga de I/O gerada pela aplicação, outras aplicações acabam por perceber um ganho em seu desempenho de I/O.

O ganho da aplicação reconfigurada será proporcional à quantidade de servidores utilizados. A implementação estudada por (HERMANN, 2006) exige que a quantidade de AppIOD's utilizados seja o número de VIOD's configurados utilizados no dNFSp. Tal escolha advinha do fato de tal quantidade ser o número ideal de servidores de I/O no sistema a menos de servidores de *backup*. Eventuais falhas destes servidores poderiam levar a uma situação ainda funcional do sistema de arquivos, porém com desempenho menor, onde um único IOD atenderia às requisições de vários VIOD's. A utilização de servidores exclusivos, então, garantiria à aplicação um desempenho equivalente ao do sistema em condições ideais.

Aplicações dominadas por escritas – as principais beneficiadas da estratégia de servidores exclusivos – possuem na quantidade de meta-servidores disponíveis um fator limi-

tante do ganho de desempenho em I/O. Uma vez que a banda de I/O ultrapassar a banda de rede disponível nos meta-servidores, o ganho de desempenho com AppIOD's será menos perceptível.

Os recursos utilizados para I/O temporário são nós normais de um cluster. A utilização de muitos recursos para AppIOD's pode interferir com a disponibilidade de nós para a execução de aplicações. Desta forma, deve-se estudar o quanto é ganho com a utilização de diferentes números de AppIOD's, de forma a não desperdiçar recursos de processamento que não irão trazer ganhos significativos para a reconfiguração.

Outra questão que surge com a utilização de recursos de processamento para I/O é a utilização exclusiva ou compartilhada de tais recursos. Com a utilização compartilhada, permitir-se-ia execução de aplicações durante à utilização do recurso como I/O (co-alocação). Esta estratégia pode ser viável quando garante-se que tais aplicações façam pouco uso de recursos de rede e disco locais ao nó de processamento. Neste trabalho, tal tarefa é considerada pertinente ao sistema de escalonamento e de seus recursos de monitoramento. Os nós alocados como AppIOD são, portanto, considerados exclusivos sob o ponto de vista do sistema de reconfiguração.

5.2 Saturação do Sistema de Armazenamento

Para evitar a utilização de recursos de I/O quando as aplicações em execução não demandam desempenho do sistema de arquivos, a reconfiguração só acontece após ele ser considerado **saturado**. Neste trabalho, considera-se o sistema saturado quando o desempenho de I/O das aplicações em execução é diminuido em função do início de uma nova aplicação ou de uma fase de I/O intensa de outra aplicação. Esta definição comporta não apenas o esgotamento da capacidade de rede e disco disponíveis nos servidores e clientes, mas também os efeitos causados pela execução concorrente de aplicações com comportamentos distintos, como observado em (KASSICK; BOITO; NAVAUX, 2009).

Para cada aplicação A_i, são observadas as bandas $B_w(A_i)$ e $B_r(A_i)$ para escrita e leitura, respectivamente. Estes valores representam a banda agregada dos nós de uma aplicação durante um período de tempo Δ e são medidos a intervalos regulares a partir de informação coletada nos meta-servidores.

A banda é considerada *estável* se a banda média durante um período $S \times \Delta$ anterior ao período de avaliação estiver dentro de um intervalo de tolerância de E da banda do instante de avaliação. O sistema será considerado *saturado* de a banda para o intervalo de avaliação estiver abaixo do intervalo de tolerância a partir do *valor máximo* já observado para a banda da operação.

Durante este período, não devem haver eventos de fim de aplicações que tenham feito acessos ao sistema de arquivos durante o intervalo. Caso tais aplicações estivessem contribuindo para a saturação observada, seu fim pode resultar em no retorno a desempenhos satisfatórios nas outras aplicações. No caso da ocorrência de tal evento, a alteração na banda deve ser sustentada durante o um período S a partir do fim da aplicação.

A Figura 5.1 ilustra esta estratégia. Na figura, duas aplicações (A_0 e A_1), exibidas na parte de baixo do gráfico. A altura dos retângulos de aplicação corresponde à quantidade de nós utilizados pela aplicação, enquanto o comprimento horizontal representa a duração do trabalho submetido ao escalonador do cluster.

Até o instante t_0, A_0 executa com exclusividade, ocupando completamente os recursos do sistema de arquivos. A partir do instante t_0, com o início de A_1, a banda individual de A_0 é reduzida além do intervalo de tolerância E. Como este menor desempenho de A_0

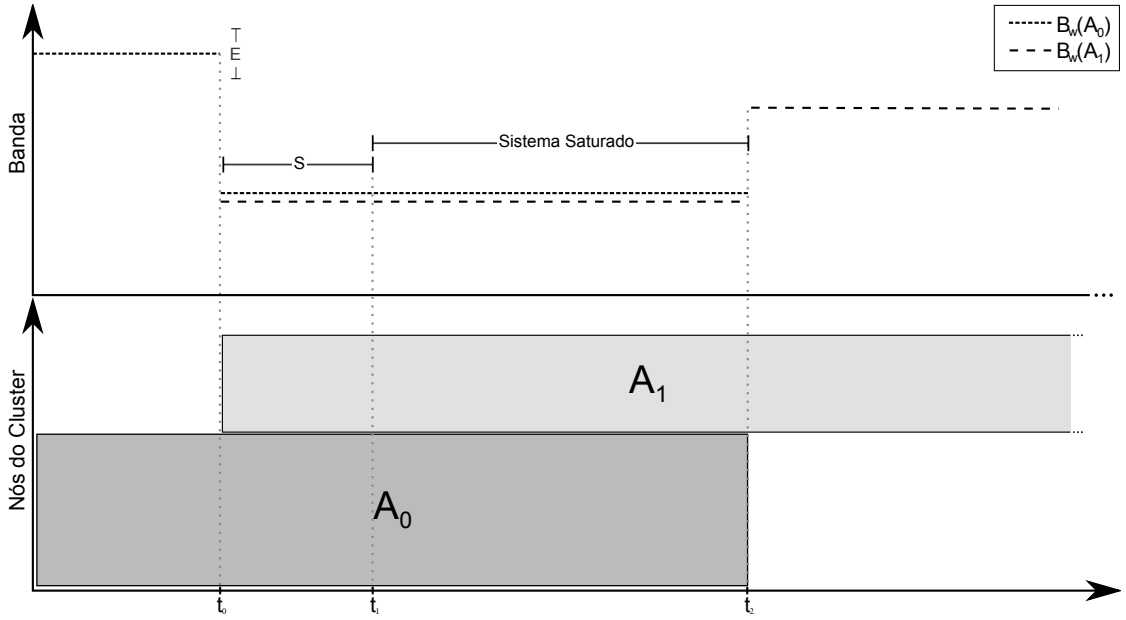


Figura 5.1: Critérios para Saturação do Sistema de Arquivos

é mantido após um período S, a partir de t_1 o sistema de armazenamento passa a ser considerado saturado. A partir do instante t_2, com o fim de A_0, a aplição A_1 pode utilizar exclusivamente os recursos do sistema de armazenamento e obter um melhor desempenho de I/O.

O sistema deixa de ser considerado saturado em duas condições: quando há apenas uma aplicação em execução ou quando as bandas de leitura e escrita voltam ao valor anterior ao momento em que o sistema foi considerado saturado.

A partir do instante de saturação, é necessário avaliar as possibilidades de reconfiguração de acordo com as características de I/O das aplicações em execução e da disponibilidade de recursos disponíveis.

5.3 Decisão de Reconfiguração

Quando o início de fases de I/O das aplicações tornar o sistema saturado, deve-se avaliar as características desta modificação de desempenho, visando evitar a reconfiguração do sistema devido a situações temporárias ou variações que podem não ser consideradas significativas. Desta forma, são aplicadas as seguintes restrições para que uma reconfiguração ocorra:

- **O sistema de armazenamento deve ser considerado saturado.** Com o critério de saturação apresentado anteriormente, deve existir mais de uma aplicação em execução e, durante um período S, a banda de I/O de alguma das aplicações em execução deve apresentar uma queda maior do que E.

- **A reconfiguração deve ocorrer antes de um intervalo R que precede fim da execução da primeira aplicação.** R constitui um intervalo de segurança antes do qual uma reconfiguração não deve ser capaz de trazer ganhos significativos em comparação com seu custo.

O intervalo *R* deve ser maior ou igual ao intervalo *S*. Desta forma, é possível avaliar o desempenho de uma reconfiguração após um intervalo onde o desempenho pode ser considerado estável.

- **Devem haver nós que possam ser utilizados como armazenamento temporário**. Um nó é considerado livre para utilização no sistema de armazenamento apenas se ele não executar nenhuma aplicação no momento da reconfiguração.

- **Dentre as aplicações em execução, deve haver uma cujo acesso ao sistema de armazenamento seja dominantemente de escrita.** Com isto, exige-se a existência de uma aplicação para a qual seja possível a inserção de servidores exclusivos sem necessidade de criação de cópias adicionais dos dados.

O sistema atendendo às condições levantadas, mais de uma aplicação de escrita pode ser considerada para a utilização com servidores exclusivos. Neste trabalho, optamos por efetuar *apenas uma reconfiguração por vez*, ou seja, uma vez que o sistema atenda às condições para a reconfiguração, são inseridos servidores exclusivos para apenas uma aplicação. Após um intervalo *S*, a situação é reavaliada e, se o sistema continuar sendo considerado saturado, uma nova aplicação pode ser selecionada para reconfiguração.

5.4 Seleção de Aplicação a Reconfigurar

Uma alteração na configuração do sistema de armazenamento, quando este é considerado saturado, é capaz de melhorar o desempenho de I/O das aplicações em execução. Tal alteração, no entanto, irá acarretar um custo com a utilização de recursos extra e com movimentações de dados devido à sincronização. Portanto, o mecanismo de reconfiguração deve levar em conta as vantagens e desvantagens de reconfigurar cada uma das aplicações em execução.

Com a utilização de servidores de aplicação apenas para aplicações dominantemente de escrita pode-se desconsiderar os custos associados à criação dos servidores: não é necessária a geração de cópias dos dados nos novos servidores e a inicialização destes é feita em poucos segundos. O custo de finalização, por outro lado, será proporcional à da quantidade de dados que deverão ser movidos dos servidores exclusivos para os permanentes.

Para modelar o ganho e o custo de uma reconfiguração, são considerados o *tempo* e a *quantidade* de nós utilizados para a reconfiguração. A utilização de AppIOD's acarreta em um **ganho** no tempo de execução das aplicações, devido à maior velocidade de escrita dos dados, e um **custo** associado à utilização de recursos para I/O.

O ganho e o custo são medidos em **nós-hora** (número de nós $\times$ tempo). Um ganho de *um nó-hora* implica que a utilização de AppIOD's resultou no equivalente à liberação de um nó de execução uma uma hora de antes do fim previsto da aplicação. Um custo de um nó-hora significa que o emprego de AppIOD's exigiu o equivalente à utilização de um nó de processamento durante uma hora de forma exclusiva para I/O.

Esta métrica permite uma comparação direta com o tamanho de aplicações em execução no cluster, que também podem ser medidos em nós-hora como o *número de nós utilizados* multiplicado pelo *tempo solicitado ao escalonador*, conforme a fórmula 5.1. Adicionalmente, o custo passa a ser medido com o de um *job* tradicional, simplificando a análise da ocupação do cluster. Esta definição de tamanho de aplicação é observada em escalonadores utilizados em ambientes de produção, como no OAR.

$$JobSize(A_i) = \#Nodes(A_i) \times Duration(A_i) \tag{5.1}$$

A Figura 5.2 ilustra o ganho e o custo da reconfiguração segundo a métrica utilizada. O sistema de armazenamento é considerado saturado a partir de t_0 e em $t_{reconfiguration}$ é tomada a decisão de inserir servidores de aplicação - no exemplo, para a aplicação A_0. A alocação para A_0 possui fim previso em $t_{End}(A_0)$. Com o ganho de desempenho em I/O devido à reconfiguração, a aplicação termina mais cedo em $t'_{End}(A_0)$.

Havendo mais de uma aplicação que possa ter seu I/O desviado para servidores de aplicação, seleciona-se a que possuir **maior ganho em comparação ao custo**. Neste trabalho, este ganho é chamado de *ganho total* e é definido na fórmula 5.2 A aplicação com maior $TotalCost$ é aquela que, tendo seu I/O reconfigurado para servidores exclusivos, irá liberar a maior quantidade recursos, medidos em nós-hora, ao fim de sua execução e da cópia dos dados para os servidores permanentes do sistema.

$$TotalGain(A_i) = Gain(A_i) - Cost(A_i) \tag{5.2}$$

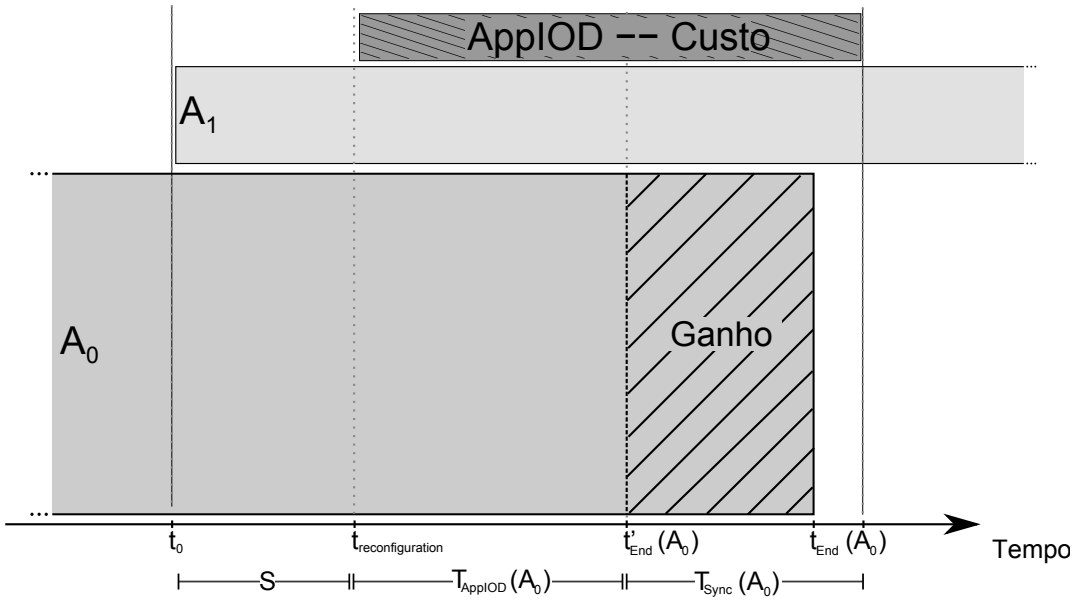


Figura 5.2: Ganho e Custo em termos de Nós-Hora. T_{AppIOD}: Tempo durante o qual a aplicação utiliza os servidores exclusivos; T_{Sync}: Tempo gasto com a sincronização dos dados entre os servidores exclusivos e os servidores permanentes do sistema.;t_{End}: Final da aplicação segundo o escalonador; t'_{End}: Final da aplicação com a reconfiguração.

No momento em que se é decidido pela reconfiguração, o sistema de já deve possuir algumas informações a respeito da aplicação para estimar o ganho que a reconfiguração poderá trazer para o sistema. Esta estimativa leva em conta que **o comportamento da aplicação após a reconfiguração se manterá inalterado**.

As informações utilizadas são as seguintes:

- $\beta(A_i)$ – Razão de I/O: Percentual do tempo de execução da aplicação que foi gasto em fases de I/O. Aplicações I/O Bound terão esta razão próxima a 1, enquanto aplicações com fases distintas de processamento e I/O terão valores menores.

- $T_r(A_i)$ – Tempo restante de execução da aplicação. Esta informação é obtida a partir do escalonador, de acordo com os tempos solicitados pelo usuário no momento da submissão do job.
- γ – Fator de melhoria no tempo de I/O: a razão entre o tempo gasto em I/O com a configuração normal sobre o tempo que seria gasto com a reconfiguração.

5.4.1 Estivativa de Ganho

$$Gain(A_i) = \#Nodes(A_i) \times \left[T_r(A_i) - T'_r(A_i)\right] \tag{5.3}$$

O *ganho* da reconfiguração é definido na fórmula 5.3. T_r, definido na fórmula 5.4, representa o tempo que a aplicação irá gastar em I/O mais um tempo de inatividade (processamento de dados nos clientes). T'_r, conforme a fórmula 5.5, representa o tempo estimado que a aplicação executará com o ganho em seu tempo de I/O.

O tempo $T_{I/O}$ da aplicação é estimado com base na razão β observada na execução da aplicação anterior ao momento da reconfiguração. Da mesma forma, estima-se o tempo de inatividade pela razão $(1-\beta)$.

Assim, estima-se $T_{I/O} = \beta(A_i) \times T_r(A_i)$. O ganho da reconfiguração em nós-hora é dado em 5.6. Esta métrica de ganho dá vantagens às aplicações com maior quantidade de nós alocados, maior tempo restante de execução e maior taxa de I/O durante sua execução.

$$T_r(A_i) = T_{I/O}(A_i) + T_{idle}(A_i) = \beta \times T_t(A_i) + (1-\beta) \times T_r(A_i) \tag{5.4}$$

$$T'_r(A_i) = T_{I/O}(A_i) \times \gamma + T_{idle}(A_i) = \gamma \times \beta \times T_r(A_i) + (1-\beta) \times T_r(A_i) \tag{5.5}$$

$$Gain(A_i) = \#Nodes(A_i) \times [\beta(A_i) \times T_r(A_i) \times (1-\gamma)] \tag{5.6}$$

5.4.2 Estimativa de Custo

O custo de uma reconfiguração é dado em função da quantidade de recursos alocados como servidores e do tempo que estes serão utilizados exclusivamente por uma única aplicação. Este tempo é referente ao tempo restante de execução da aplicação reconfigurada e do tempo necessário à sincronização dos dados com os servidores permanentes.

Neste trabalho, no entanto, optou-se por não considerar o tempo de sincronização para a métrica do custo. A devolução dos recursos pode ser postergada pelo escalonador até que haja necessidade destes processadores para a execução de novas aplicações ou até que o sistema deixe de ser considerado saturado, evitando a interferência no desempenho de I/O. Outra alternativa é reutilizar estes servidores para I/O exclusivo de novas aplicações que sejam consideradas saturadas.

O custo da reconfiguração de I/O para uma aplicação é dado pela fórmula 5.7.

$$Cost(A_i) = n_{AppIOD} \times T'_r(A_i) = n_{AppIOD} \times (\beta \times \gamma \times T_r(A_i) + (1-\beta) \times T_r(A_i)) \tag{5.7}$$

5.4.3 Ganho Total

A partir de 5.6 e 5.7, a fórmula do ganho total com a reconfiguração de de uma aplicação, conforme 5.2 é dada por:

$$TotalGain(A_i) = \left(\#Nodes(A_i) + n_{AppIOD}\right) \times \beta \times T_r(App_i) \times (1-\gamma) - n_{AppIOD} \times T_r(A_i)$$

Será eleita para utilização com IOD's exclusivos a aplicação com o maior $TotalGain$. Como γ é constante para qualquer uma das aplicações selecionadas, ele não irá influenciar a escolha da aplicação com melhor ganho. Pode-se, portanto, desconsiderar o fator $(1-\gamma)$ da fórmula e fazer a seleção conforme a Equação 5.8.

Assim, a seleção prioriza as aplicações que gastam mais tempo bloqueadas em operações de I/O, com maior quantidade de nós e que utilizarão os recursos de I/O por mais tempo.

$$TotalGain^{*}(A_i) = \#Nodes(A_i) \times \beta(A_i) \times T_r(A_i) - n_{AppIOD} \times T_r(A_i) \tag{5.8}$$

5.5 Disponibilidade de Recursos em Ambientes de Alto Desempenho

Uma questão que surge com a proposta de utilizar recursos de processamento para I/O é a existência de nós ociosos em um cluster. Para verificar a relevância do modelo proposto, foram consultados os traços disponibilizados em (FEITELSON, 2009). Estes traços representam clusters de diversas instituições entre os anos de 1993 e 2007. Estes cluster são utilizados em ambientes de produção com diferentes grupos de usuários e tipos de aplicação-alvo.

Dentre os traços disponíveis, optou-se pelo obtido do cluster Atlas do *Lawrence Livermove National Laboratory*. Este cluster possui 1152 nós com 8 processadores AMD Opteron com freqüência de 2.4 GHz e 16 GB de memória.

O traço contém registro das submissões de trabalhos entre novembro de 2006 e junho de 2007. A escolha por este traço deveu-se à sua atualidade e à variedade de trabalhos com diferentes números de processadores e tempos de execução. A Figura 5.3 apresenta a distribuição dos tamanhos dos trabalhos listados no traço e a distribuição dos tempos de execução destes trabalhos. Nota-se a abundância de trabalhos que fazem uso apenas de uma fração dos recursos do cluster. Nota-se também uma dominância de trabalhos com curta duração: trabalhos com menos de 10 minutos respondem por aproximadamente 60% das alocações. Trabalhos com duração significativa, no entanto, também encontram-se representados, visto que trabalhos com tempos de 1 hora a 4 dias representam aproximadamente 30% da carga do cluster.

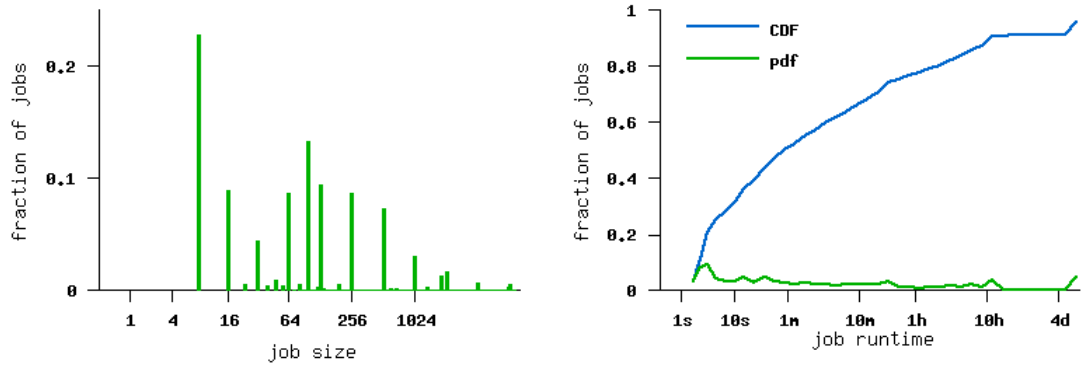


Figura 5.3: Distribuição de tamanho e duração de trabalhos submetidos ao cluster Atlas. Imagem disponível em `http://www.cs.huji.ac.il/labs/parallel/workload/l_llnl_atlas/index.html`.

Uma primeira análise deste traço consistiu em dividí-lo em períodos de menor duração e avaliar a utilização do cluster nestes intervalos. A divisão em intervalos e o cálculo do percentual de utilização foi feito com a ferramenta *swf_explore*[1]. Para cada intervalo, é calculado o percentual de utilização do cluster, ou seja, a razão entre as horas-cluster utilizadas sobre as horas-cluster disponíveis. O intervalo utilizado neste trabalho foi de 1 *hora*.

A Figura 5.4 apresenta, para cada percentual de utilização do cluster, a quantidade de intervalos no traço. Primeiramente, nota-se uma grande quantidade de intervalos com percentual de utilização abaixo de 1%: dos 5541 intervalos, 1463 apresentaram esta taxa de utilização. Por esta segunda figura, podemos observar outras taxas de utilização significativas, especialmente as em torno de 75% e 90%.

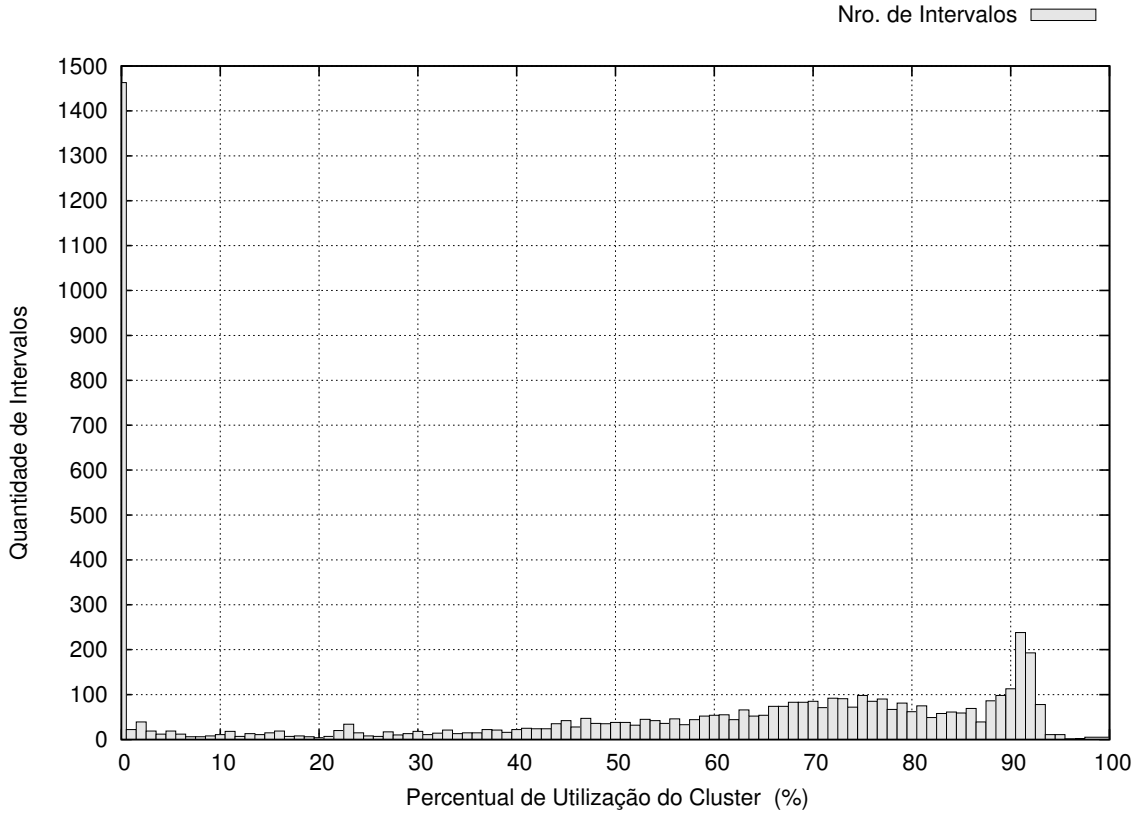


Figura 5.4: Utilização do Cluster Atlas, Intervalos de 1 hora

Na Figura 5.5, é apresentada o percentual de ocorrência destas taxas de utilização do cluster, juntamente com seu percentual cumulativo. Pode-se notar que os intervalos com utilização abaixo de 10% representam quase 30% dos intervalos. Os intervalos com taxa entre 70 e 80% representam a segunda maior concentração, com 15% do total de intervalos, enquanto os intervalos $[80,90[$ e [90,100%[representaram, respectivamente, 11.80 e 11.73% dos intervalos. Cabe ressaltar ainda que aproximadamente 61% dos intervalos apresentaram taxa de utilização do cluster abaixo de 50%, o que indica a existência de muitos recursos ociosos durante grande parte do funcionamento deste cluster.

No cenário do cluster Atlas, portanto, há uma grande quantidade de momentos em que existem recursos disponíveis. Estes recursos podem, portanto, ser utilizados durante seus

[1]Disponível em `http://gforge.inria.fr/scm/?group_id=1881`

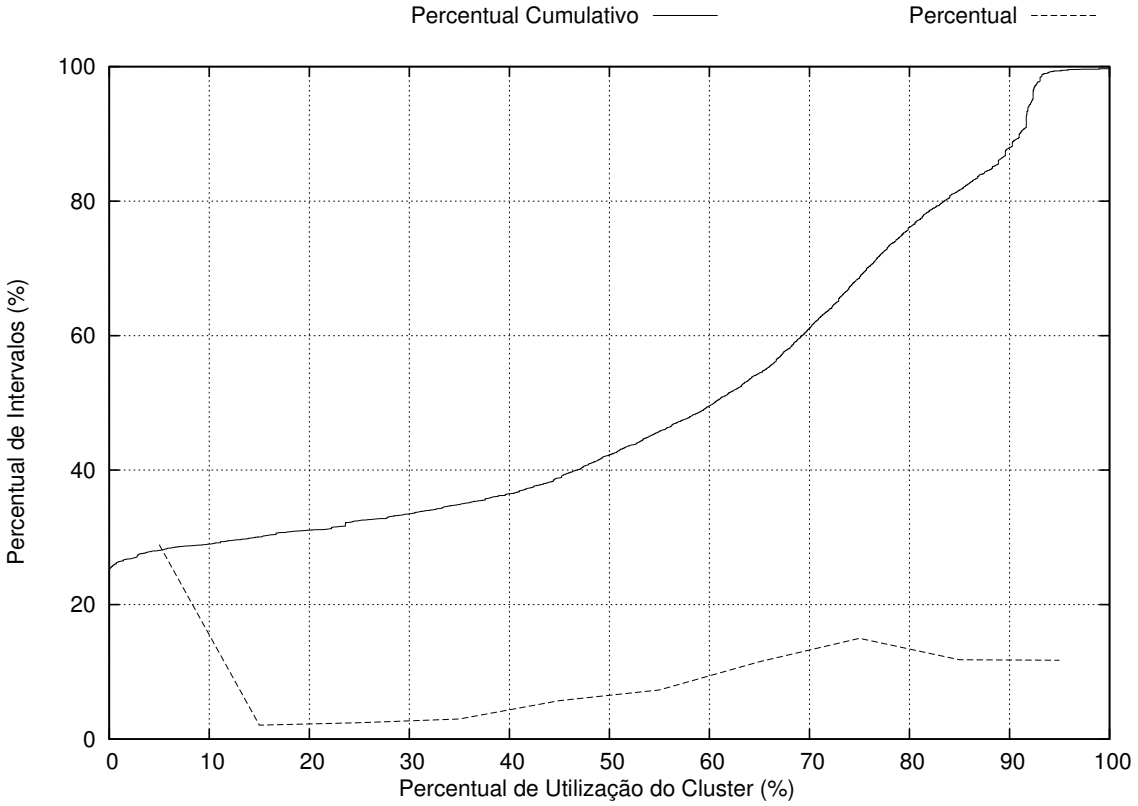


Figura 5.5: Distribuição do Percentual de Utilização do Cluster Atlas, Intervalos de 1 hora

períodos de ociosidade para melhorar o desempenho de I/O das aplicações em execução.

5.6 Considerações sobre o Modelo de Reconfiguração

Neste capítulo, foi apresentado o modelo de reconfiguração proposto neste trabalho. Este modelo visa definir os momentos onde uma reconfiguração pode ser iniciada, bem como qual aplicação deve ter seu I/O reconfigurado para melhorar o desempenho do sistema.

Foi apresentada uma métrica para medir o ganho que cada aplicação obteria com a reconfiguração de seu I/O. Esta métrica, utilizada junto a um sistema de gerência de recursos, permite avaliar a quantidade de recursos gastos com o I/O e o ganho em tempo de execução que poderia ser obtido com a utilização de recursos exclusivos.

São utilizadas informações a respeito das aplicações cuja obtenção não é considerada escopo deste trabalho. O valor para a razão β para uma aplicação, especificamente, deve ser obtida com ferramentas capazes de detectar comportamentos temporais no fluxo de I/O e definir claramente os início e fim de uma fase de I/O ou ociosidade. A detecção online destes padrões de acesso é estudada em trabalhos encontrados na bibliografia como (TRAN; REED, 2001), (BYNA et al., 2008) e (RIBLER; SIMITCI; REED, 2001).

6 IMPLEMENTAÇÃO

Para prover suporte ao nível de dinamismo necessário ao ambiente proposto, o monitoramento online do desempenho do sistema de arquivos, a tomada de decisão de reconfiguração e a execução de aplicações, foram feitas diversas intervenções no sistema de arquivos dNFSp, bem como o desenvolvimento de ferramentas auxiliares a estas tarefas. As seguintes seções apresentam estas ferramentas e modificações do dNFSp, bem como algumas decisões de projeto tomadas durante o desenvolvimento deste trabalho.

6.1 Very Simple Scheduler

Um sistema de arquivos distribuído com capacidade de adaptar seu número de servidores às características de suas aplicações exige um gerenciamento diferenciado dos recursos de um cluster. Este gerenciamento, no entanto, não deve ser tarefa executada pelo sistema de armazenamento, e sim do sistema de gerenciamento de recursos do ambiente. No cenário proposto por este trabalho, o gerenciador de recursos, além de reservar recursos para as aplicações dos usuários, deve ser capaz de reservar recursos de armazenamento quando estes forem considerados necessários.

Enquanto sistemas de gerenciamento como OAR, PBS e SGE são amplamente conhecidos e utilizados e validados em ambientes de produção, sua utilização em ambientes experimentais é dificultada pela complexidade de seus recursos e pré-requisitos de instalação. Além disso, a adição de novas funcionalidades a estes sistemas é dificultada por sua complexidade.

Adicionalmente, estes sistemas efetuam a execução de trabalhos sob a demanda de comandos de usuário (e.g. *oarsub*, *pbssub*, ...). Em um ambiente de testes onde as aplicações a executar já estão pré-definidas, tornar-se-ia necessário um programa auxiliar para executar os comandos de submissão de trabalho de acordo com um *schedule* pré-definido.

O *Very Simple Scheduler* – (VSS) é um sistema de gerenciamento de recursos simplificado que provê funcionalidades básicas à utilização de um cluster: a execução de programas *batch* com exclusividade sobre uma quantidade solicitada de recursos de processamento. Os tempos de início de trabalho, duração e quantidades de recursos são definidas através de um *traço de execução*, utilizado como entrada do escalonador. Os recursos do cluster são definidos em um arquivo de entrada que é mapeado para um conjunto de nós *virtuais*.

Os programas a executar para cada job são definidos em um arquivo de *mapeamento de jobs*. A execução de ações não previamente programadas – como a solicitação de recursos exclusivos para armazenamento e sua posterior liberação – são feitas através de um programa auxiliar que se comunica com o *daemon* do escalonador através de chamada

remota de procedimentos. Estas características serão explicadas nas seções seguintes.

6.1.1 Traços de Execução

Os trabalhos a executar no VSS são fornecidos através de traços de submissões. Esta decisão visa permitir a reprodução de situações reais nos testes executados. Os traços utilizados são originários de clusters de produção e representam situações reais de carga de trabalho e duração dos jobs.

O formato de traço utilizado é um subconjunto das informações presentes no *Standard Workload Format* (CHAPIN et al., 1999). O VSS utiliza as as seguintes informações para a execução dos trabalhos:

Hora de Início do Trabalho (*start_time*) indica o momento em que o trabalho foi selecionado para execução no cluster.

Número de Processadores (*req_procs*) representa o número de **processadores** reservado para o trabalho.

Hora de Fim do Trabalho indica quando os recursos associados a um trabalho são liberados para o sistema de gerenciamento.

Enquanto informações adicionais estão presentes na especificação – como, por exemplo, *hora de **submissão** do trabalho* e tempo *requerido de execução* – escolheu-se utilizar apenas os três campos supracitados, uma vez que vários traços não contém tais informações. Desta forma, o VSS assume que os trabalhos são sempre submetidos no momento em que entram em execução e que o tempo solicitado sempre reflete o tempo utilizado.

6.1.2 Mapeamento de Trabalhos para Programas

Os formato de traço utilizados como entrada não indica que programas são executados em cada trabalho. Para executar diferentes programas de acordo para cada trabalho presente no traço, VSS utiliza um arquivo de configuração chamado de *mapeamento*.

Neste arquivo, um *JobID* é associado a dois programas. O primeiro programa, chamado de *script*, representa o comando a executar no início do trabalho. O segundo, chamado de *cleanup*, é executado ao fim do trabalho e é responsável por garantir a sanidade dos nós após o fim do programa *script*. Estas tarefas incluem excluir arquivos e eliminar processos. Enquanto tais tarefas geralmente são garantidas pelo ambiente de gerenciamento de recursos, considerou-se mais simples a utilização de scripts adaptados para cada caso de teste.

Outra tarefa importante efetuadas pelos programas *script* e *cleanup* é informar ao dNFSp, através do comando *iod_mgmt* do início de uma nova aplicação e associar a ela os nós de execução. Este passo é de extrema importância, já que esta associação é utilizada pela ferramenta de monitoramento para avaliar o desempenho de uma aplicação.

O mapeamento fornecido não precisa representar todos os trabalhos do traço. Os trabalhos omitidos são considerados *trabalhos virtuais* pelo escalonador. Neste caso, eles não executam nenhum script em seu início e não necessitam alocar recursos para sua execução. No entanto, o VSS associa a estes trabalhos a quantidade solicitada de *recursos virtuais*.

6.1.3 Gerência de Recursos

Como o VSS utiliza entrada de traços de execução para lançar trabalhos em um conjunto de nós, há casos em que a quantidade de nós disponíveis para os testes não são

suficientes para a execução de todas as aplicações presentes nos traços. Isto se torna mais problemático quando o traço utilizado provém de um ambiente de produção com milhares de nós e o ambiente de testes no qual o VSS é executado possui poucos recursos.

Para permitir a utilização de tais traços, o VSS trabalha sobre uma quantidade de *recursos virtuais*. Estes recursos representam processadores e devem ser da mesma quantidade do ambiente do qual o traço de execução foi extraído.

Um subconjunto destes recursos é mapeado para os *recursos reais* presentes no ambiente de testes. Aplicações que possuem mapeamento para um *script* são executadas apenas em *recursos reais*. Quando o mapeamento de uma aplicação não está definido, ela utilizará *recursos virtuais*[1].

A criação dos recursos virtuais se dá no início da execução e é feita com base em dois parâmetros do escalonador: o **número de nós** presentes no cluster ao qual o traço faz referência e o **número de cores** presente em cada nó. O VSS assume que os nós são homogêneos na quantidade de cores.

Para cada nó real do escalonador são mapeados todos os cores de um nó do cluster simulado. Se a quantidade de cores do cluster simulado e do ambiente de testes diferir, um nó de testes poderá executar mais processos do que os processadores que possui. Enquanto isto representa um problema para aplicações com processamento pesado, não é considerado um problema grave para aplicações dominadas por I/O, visto que estas passarão a maior parte de sua execução bloqueadas em operações sobre os dados.

O objetivo de incluir aplicações virtuais na simulação dos traços é tornar a visão do sistema de monitoramento fiel ao que foi observado na prática em termos de aplicações em execução. No ambiente proposto, as aplicações virtuais representam aplicações que não utilizaram o sistema de armazenamento, enquanto as aplicações reais são registradas junto ao sistema de armazenamento e seus acessos monitorados durante sua execução.

6.1.4 Comandos do Escalonador

A lista de trabalhos executada no VSS é imutável. A única alteração permitida é a solicitação de recursos para utilização como I/O e sua liberação ao final da aplicação. Estas tarefas são efetuadas com a ferramenta *vss_client*.

O *vss_client* se comunica através de XML-RPC com o escalonador. Os seguintes comandos são suportados:

start_io <JobID> inicia o serviço de I/O exclusivo para a aplicação com identificador *JobID*. Este comando é utilizado pela ferramenta de monitoramento do sistema de armazenamento quando esta decide pela reconfiguração do I/O para uma aplicação.

free_io <Hostname> libera um nó previamente utilizado como I/O exclusivo de uma aplicação. Este comando é chamado pelo IOD de aplicação assim que ele concluir a sincronização com os servidores permanentes.

list_running apresenta uma listagem das aplicações em execução e o tempo restante até o final de sua execução.

[1] Uma aplicação dita *virtual* pode utilizar recursos reais se, no instante em que ela é iniciada, não houverem recursos virtuais suficientes

6.2 Controle de Alteração de Dados para AppIOD

Em sua implementação original, os AppIOD's deveriam estar presentes já no início da aplicação. Neste caso, a sincronização efetuada ao final da execução necessitava apenas enviar todos os dados dos arquivos de dados para os servidores exclusivos.

No presente trabalho, os IOD's exclusivos podem ser iniciados após o início de uma aplicação. Se esta já gerou dados nos servidores originais do sistema, estes seriam sobrescritos no processo de sincronização. Para evitar esta perda de dados, foi implementado junto aos IOD's um esquema simples de marcação dos blocos escritos baseado em vetores de bits.

Para cada arquivos de dados, é associado um *bitmask-file*. Cada bit deste arquivo representa um bloco de *1 Kilobyte* dentro do arquivo de dados.

Quando uma operação de escrita é efetuada por um AppIOD, ele irá marcar como 1 o bit correspondente ao *offset* escrito, indicando que o bloco foi alterado neste servidor. Operações de leitura verificam o bit referente ao offset requisitado e respondem a requisição caso o bit esteja marcado. Caso contrário, a operação de leitura é repassada a um servidor original do sistema de arquivos para que este possa completar a requisição.

O processo de sincronização é feito de forma semelhante. Ao iniciar a sincronização, o AppIOD percorre o *bitmask-file* e envia para o servidor correspondente apenas os blocos que nele foram alterados.

Esta abordagem foi escolhida devido à sua simplicidade de implementação e por não apresentar variações de desempenho em função do *offset* da requisição de leitura ou escrita. Ela possui, no entanto, a desvantagem de o arquivo de *bitmask* crescer linearmente com o tamanho do arquivo de dados, podendo ocupar até $256MB$ quando o arquivo de dados possuir $2GB$.

Outra desvantagem óbvia é a possibilidade de corrupção dos dados quando o tamanho das requisições feitas pela aplicação for menor do que *1 kilobyte* ou quando estas operações forem efetuadas de forma não alinhada com este tamanho. Esta limitação, no entanto, foi considerada contornável, uma vez que várias estratégias de I/O paralelo estimulam a operação de grandes quantidades de dados em cada operação e que bibliotecas de I/O dedicadas podem garantir este alinhamento de dados.

6.3 dNFSp Performance Monitor

O **dNFSp Performance Monitor** é uma ferramente desenvolvida neste trabalho para observar o desempenho de I/O das aplicações que utilizam o sistema de arquivos. Sua implementação é dividida em duas partes: a *libpfm*, uma biblioteca com as estruturas e funções básicas para a coleta das informações a respeito dos acessos dos clientes, e a aplicação *pfmond*, responsável por buscar estas informações junto aos servidores, efetuar a análise conforme proposta no modelo e aplicar as regras de reconfiguração, solicitando junto ao gerenciador de recursos a execução do serviço de I/O exclusivo.

6.3.1 libPFM – Coleta dos dados junto aos Meta-servidores

A *libpfm* foi desenvolvida para gerenciar as estruturas de dados necessárias ao monitoramento do desempenho das aplicações. Desta forma, ela deve ser capaz de armazenar informações sobre os acessos dos clientes de uma aplicação. Estas informações serão, posteriormente, analisadas pelo serviço *pfmond*.

No dNFSp, clientes de uma mesma aplicação estarão distribuídos nos meta-servidores.

Devido à natureza *stateless* do protocolo NFS, um meta-servidor não armazena qualquer informação a respeito dos clientes que o utilizam.

A *libpfm* se responsabiliza por manter um mapeamento entre clientes e suas aplicações. Para tanto, integrou-se a *libpfm* a chamadas previamente disponíveis no dNFSp referentes à criação de aplicações: *create_execution*, *add_application_nodes* e *delete_execution*.

Estas chamadas foram desenvolvidas no trabalho de (HERMANN, 2006) para prover suporte ao redirecionamento de I/O de aplicações paralelas. Elas são invocadas através da ferramenta *iod_mgmt*, a qual se responsabiliza por contactar todos os meta-servidores, tornando a aplicação e os nós por ela utilizados conhecidos em todo o sistema. Neste trabalho, estas chamadas foram integradas à *libpfm* e algumas estruturas substituídas para que fosse possível armazenar as informações a respeito de desempenho.

A chamada **create_execution** registra junto à *libpfm* um novo identificador de aplicação. Este identificador é passado como parâmetro pela ferramenta *iod_mgmt* e, no ambiente de testes é o mesmo identificador do trabalho do escalonador VSS.

O comando **add_application_nodes** associa **endereços de rede IP** com um identificador de aplicação. Todos os acessos provindos dos endereços listados serão associados à aplicação especificada. Esta estratégia impede a utilização de um mesmo nó por mais de uma aplicação. Escolheu-se, portanto, não trabalhar com esta situação.

As operações de escrita e leitura nos meta-servidores foram instrumentadas para fazer chamada à função *pfm_trace_op*. Esta função recebe como entrada o endereço IP do cliente, o tipo de operação (leitura ou escrita) e a quantidade de dados solicitada. A função *pfm_trace_op* procura o endereço do cliente em uma *hashtable* e, se encontrada uma aplicação registrada para o cliente, a quantidade de dados solicitada é acumulada em um campo da estrutura que descreve a aplicação. Em intervalos regulares, esta informação é solicitada pelo daemon *pfmond* e as quantidades de dados transferidas pelas aplicações são zeradas.

6.3.2 O Performance Monitor *Daemon*

A função da *libpfm* junto aos meta-servidores é exclusivamente coletar quantidades de dados lidas e escritas por clientes de uma aplicação. A coleta, agregação e processamento destas informações é função do *pfmond*. Uma vez que os meta-servidores acumulam a quantidade de dados lidos e escritos apenas pelo clientes dos quais recebem requisições, o *pfmond* deve contatar estes servidores periodicamente e agregar os dados parciais para calcular o desempenho de I/O das aplicações. O *Performance Monitor Daemon* utiliza as estruturas e funções da *libpfm*, porém com algumas diferenças.

O *pfmond* faz requisições periódicas aos meta-servidores. O tempo entre duas requisições é definido pelo parâmetro *delta* e reflete o parâmetro Δ descrito no Capítulo 5.

O *daemon* contata todos os meta-servidores e solicita informações sobre as aplicações que estão registradas no sistema de armazenamento. Para cada aplicação, é armazenado em um vetor chamado *samples* onde cada elemento representa a soma das quantidades de dados informadas pelos meta-servidores. Este vetor é utilizado para calcular as bandas de leitura e escrita das aplicações com uma resolução Δ.

A Figura 6.1 ilustra o funcionamento básico do *pfmond*. A intervalos de tempo de duração Δ, os meta-servidores são contatados e enviam ao *daemon* apenas as quantidades acumuladas durante este intervalo. Ao receber estar informações, o programa adiciona uma posição ao vetor *samples* das aplicações para as quais ele recebeu informações e armazena na nova posição a soma das quantidades enviadas pelos meta-servidores. Este vetor informa, portanto, as bandas de leitura e escrita observados durante os últimos in-

tervalos de duração Δ.

O vetor *samples* possui um tamanho fixo de 256 posições. Para evitar limitações no tempo de execução de uma aplicação, as informações mais antigas são descartadas quando o vetor é completamente preenchido.

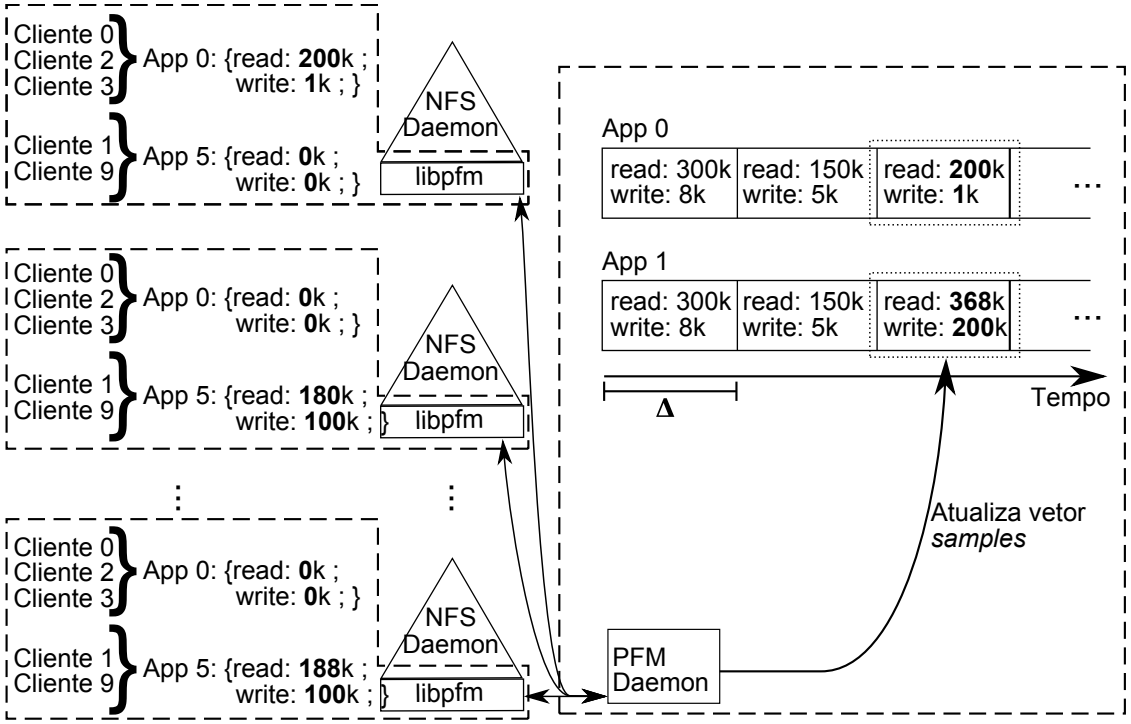


Figura 6.1: Coleta de dados e armazenamento no vetor *samples*: A intervalos regulares de Δ, o Performance Monitor Daemon contata os meta-servidores e atualiza a última posição do vetor.

Após a coleta dos dados, o *daemon* passa à fase de análise. Nesta fase, é utilizado um mapeamento entre aplicações e classes de aplicação, conforme descritas na Seção 4.4. Este mapeamento é feito através de um arquivo fornecido no início da execução do *pfmond*.

A análise também conta com informações a respeito das aplicações em execução através da chamada *list_running*, previamente descrita na Seção 6.1.4. Com isso, o *daemon* possui uma visão *on-line* das aplicações em execução e o tempo que ainda estarão em execução de acordo com o tempo solicitado na submissão.

Com estas informações, o *pfmond* verifica a estabilidade da banda de leitura e escrita das aplicações conhecidas, calcula o ganho de reconfiguração e utiliza as outras métricas descritas na Seções 5.3 e 5.4. Caso as condições forem atendidas, o programa utiliza a chamada *start_io* do VSS para efetuar o redirecionamento de I/O para a aplicação escolhida.

6.4 Visão geral

A Figura 6.2 mostra a interação entre os diferentes componentes necessários à reconfiguração automática proposta nesse trabalho. A ferramenta VSS utiliza um traço de submissões para definir a execução de trabalhos nos nós utilizados no experimento. Para

cada trabalho *real*, o programa *iod_mgmt* é utilizado para informar aos meta-servidores da existência de uma nova aplicação e dos clientes a ela associados. Durante a execução dos testes, o *PFM Daemon* comunica periodicamente com os meta-servidores e coleta as informações obtidas pela *libpfm*. Neste momento, também são solicitadas ao escalonador informações sobre as aplicações em execução. Quando detectada a necessidade de reconfiguração de I/O, o *pfmond* faz a chamada *start_io* do VSS, solicitando recursos para utilizar como I/O exclusivo para alguma aplicação. Ao final de uma aplicação, o escalonador notifica os meta-servidores através do *IOD Management* que, por sua vez, notifica os AppIOD's para que iniciem a sincronização dos dados com os servidores permanentes. Quando esta sincronização é finalizada, os AppIOD's notificam o escalonador para liberar o nó para execução de novas aplicações.

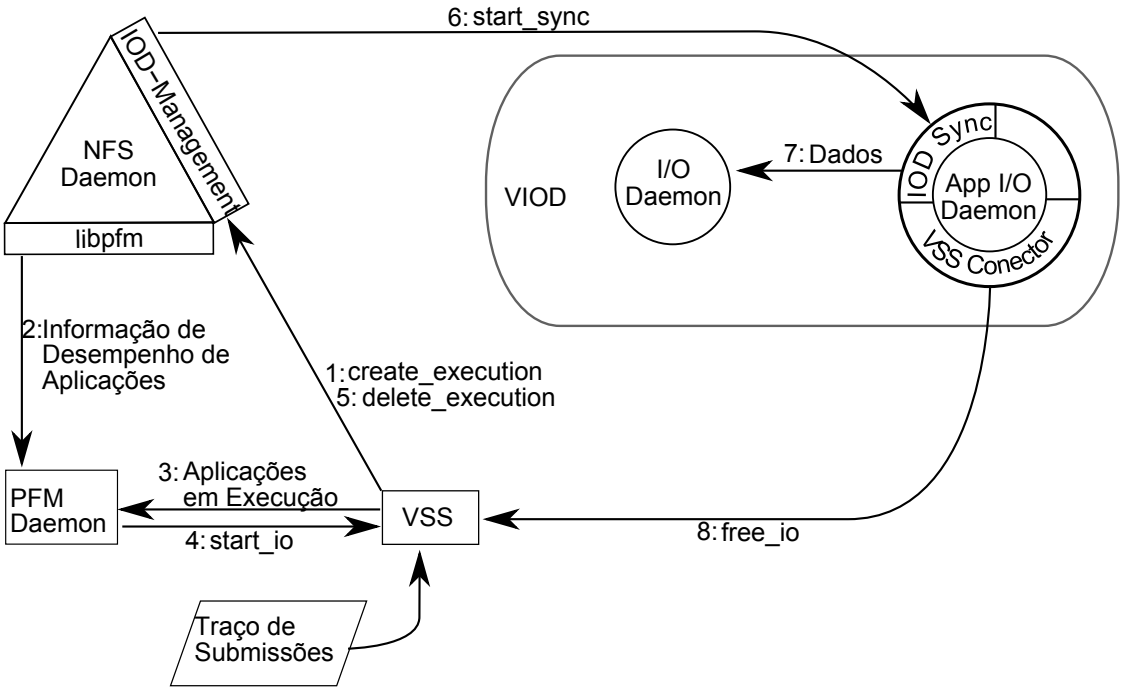


Figura 6.2: Diagrama das Interações entre o VSS, o Daemon PFM e o sistema de arquivos dNFSp

7 AVALIAÇÃO

Este capítulo apresenta a avaliação do protótipo descrito no Capítulo 6. Além da avaliação do impacto do monitoramento no sistema de arquivos, é feita uma avaliação do comportamento de aplicações temporais com relação aos seus acessos de I/O e a como o sistema de armazenamento reage uma vez que é feita a reconfiguração.

7.1 Ambiente de Testes

Para a execução dos testes, foi utilizado o cluster *Pastel*, parte do *site* de Toulouse da plataforma Grid5000 (BOLZE et al., 2006). Este cluster é constituído de 80 máquinas Sun modelo *Sun Fire X2200 M2*, com $8GB$ de memória RAM e processadores *AMD Opteron 2218* com freqüência de *2.6GHz*. Os nós são interconectados por rede Gigabit Ethernet através de um roteador Cisto 7006.

Cada máquina conta com um disco SATA *Hitachi HDS7225S V5DO* de *250GB* com vazão bruta de até $62,69MB/s$ para leitura[1] e $34,67MB/s$ para escrita[2]. Estes valores foram medidos sem a influência de *caches* de disco ou sistemas de arquivos.

Para os testes, foram utilizadas 4 máquinas atuando, cada, como meta-servidor e IOD. Os clientes foram distribuídos homogeneamente entre os servidores, com opções de montagem *wsize* e *rsize* (tamanho de blocos para leitura e escrita) de $8KB$.

7.2 Benchmark de I/O

Para gerar a carga característica das classes de aplicação aqui utilizadas foi utilizado o *benchmark* MPI IO Test 21. Esta ferramenta utiliza a biblioteca MPI e as chamadas da MPI-IO para escrever ou ler uma determinada quantidade de **objetos** em um ou vários arquivos de dados. Um **objeto** é uma região contígua escrita em um arquivo. Os objetos possuem um **tamanho** regular e podem ser escritos de forma contígua ou entrelaçada. Quando gerenciados de forma contígua, uma única região do arquivo conterá todos os objetos de uma instância do benchmark ($\{Obj1_{clt\ 1}, Obj2_{clt\ 1}, \cdots Obj1_{clt\ 2}, \cdots\}$). No modo entrelaçado (*strided*), são armazenados, alternadamente, os objetos de todos os clientes com todos os objetos lidos ou escritos por uma mesma instância do benchmark ($\{Obj1_{clt\ 1}, Obj1_{clt\ 2}, \cdots Obj2_{clt\ 1}, Obj2_{clt\ 2} \cdots\}$). Cada objeto é escrito ou lido em uma única operação *Write* ou *Read*.

O *MPI IO Test 21* pode utilizar um único arquivo compartilhado para todos os clientes (estratégia *N-1*) ou um arquivo por instância (estratégia *N-N*). Nos testes, optou-se por

[1] Valor fornecido pela ferramenta *hdparm*

[2] Valor medido com a ferramenta *dd*

utilizar a estratégia *N-N*. Esta estratégia permite escrever mais dados do que seria possível com um único arquivo compartilhado, uma vez que o protocolo NFSv2 possui limite de 2GB para o tamanho dos arquivos.

O benchmark foi modificado para introduzir um tempo de espera entre a leitura ou a escrita de dois objetos. Isto foi feito para possibilitar a simulação de aplicações da classe *checkpoint I/O*, simulando, do ponto de vista de I/O, o tempo gasto pela aplicação processando os dados.

7.3 Desempenho do dNFSp com Servidores de Aplicação

A utilização de servidores exclusivos traz um aumento na vazão do sistema de arquivos com a inclusão de mais recursos de I/O. Este aumento, no entanto, estará limitado a diversos fatores: a relação entre a quantidade de servidores de meta-dados, vazão individual dos discos, quantidade de clientes, etc. Desta forma, faz-se necessário testar como a utilização de AppIOD's influencia o desempenho do dNFSp.

7.3.1 Desempenho para Operações de Escrita e Leitura

Para estes testes, foram utilizados até 48 clientes, divididos em conjuntos de 24 clientes chamados de *execução 1* e *execução 2*. O teste consiste em duas fases: na primeira, as duas execuções efetuam operações de escrita, gravando, cada uma, até 512MB. Foi imposto um tempo limite de 120 segundos nesta fase: caso o *benchmark* ultrapasse este tempo, ele será encerrado. Este limite visa garantir que o desempenho será medido apenas durante o tempo em que as aplicações executam concorrentemente. Na segunda fase, a *execução 2* efetua a leitura dos dados previamente gravados, enquanto a *execução 1* efetua, novamente, operações de escrita. Desta forma, foi possível observar o impacto que a utilização de AppIOD's possui tanto para as aplicações que têm seu I/O reconfigurado quanto para as que continuam utilizando o sistema permanente para operações de escrita ou de leitura.

A Figura 7.1 apresenta a banda combinada de de escrita para as duas execuções sobre o dNFSp sem a utilização de AppIOD's. O *eixo x* representa a quantidade de nós utilizados na execução 1. Cada barra representa a quantidade de clientes utilizados na segunda execução.

Para a execução 1 com 8 e 16 clientes, a utilização de banda aumentou com o maior número total de clientes, indicando que o sistema ainda não se encontrava completamente saturado. Com 24 clientes na execução 1, o desempenho permaneceu dentro da faixa de tolerância de 10% para as três configurações da execução 2.

Na Figura 7.2 o teste é repetido, porém adicionando 4 servidores de aplicação para a execução 1. Para 8, 16 e 24 clientes, o desempenho de escrita do sistema aumentou com o aumento na quantidade de clientes. Para o caso de 48 clientes ao total – 24 em cada execução – observou-se um desempenho de $173,11 MB/s$, representando um ganho de 28%.

Na Figura 7.3 é apresentada a banda combinada de escrita e leitura (a segunda fase do teste) sem a utilização dos servidores exclusivos. A maior vazão de dados na presença de operações de leitura é uma característica do dNFSp devido à menor ocupação da capacidade de rede dos meta-servidores. Com o número fixo de servidores, no entanto, as capacidades de rede e disco são rapidamente ocupadas com o aumento do número de clientes. O aumento do número de clientes no teste atinge rapidamente a capacidade de transferência dos servidores: a partir de 24 clientes no total, o desempenho do teste

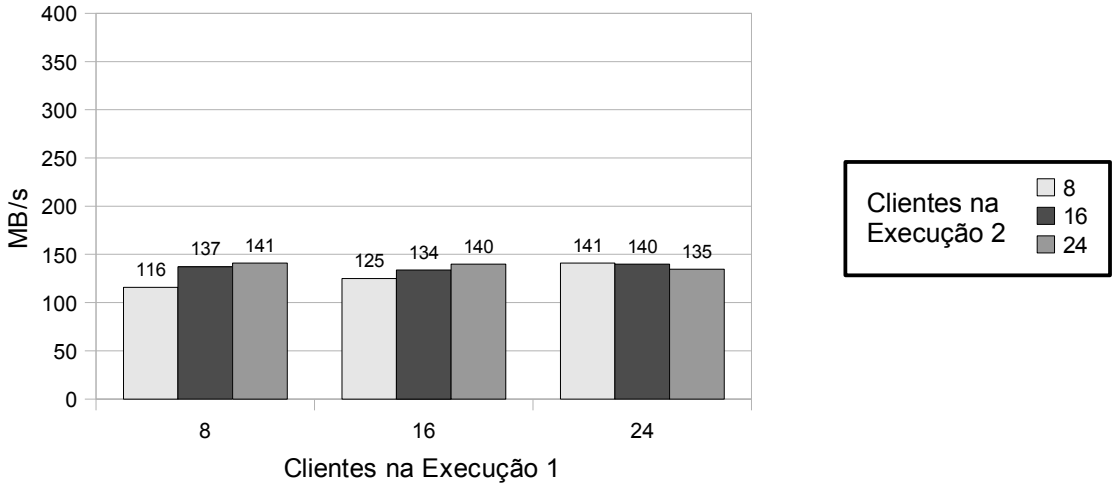


Figura 7.1: Banda Combinada da Fase 1 (Escrita/Escrita) sem utilização de AppIOD.

apresenta pouca variação.

O mesmo caso é ilustrado na Figura 7.4 com a inserção de 4 servidores exclusivos para a execução 1. Neste caso, a presença de mais recursos de I/O garante um aumento na vazão de dados como aumento na quantidade de clientes efetuando operações de leitura. Em comparação com o caso sem a utilização de AppIOD's, observou-se um aumento de desempenho de até 17% no caso de 16 clientes na execução 1 e 24 na 2.

Na Figura 7.5 são apresentadas as vazões de escrita da execução 1 para as duas fases do teste. O número de clientes na execução 2 é fixado em 32, enquanto o número de clientes na execução 1 varia de 8 a 32. São utilizados 4 servidores exclusivos para a execução 1. Observa-se que o ganho em desempenho de I/O para aplicação reconfigurada será maior quando as outras aplicações presentes no sistema forem caracterizadas por fases de leitura. Na visão global do sistema, no entanto, conforme visto na Figura 7.4, o aumento de clientes na execução 1 não produz diferenças significativas no desempenho total do sistema. Desta forma, o aumento em desempenho para a execução 1 acaba sendo pago por uma queda em desempenho para a segunda execução.

7.3.2 Dimensionamento do Sistema de Armazenamento Temporário

A implementação do dNFSp exige que sejam utilizados tantos servidores exclusivos quanto VIOD's. Esta configuração limita a quantidade máxima de AppIOD's a um valor definido durante a inicialização do sistema de arquivos.

É possível, no entanto, avaliar o ganho com a utilização de mais AppIOD's do que servidores de dados permanentes. Para tanto, inicializa-se o sistema com uma quantidade de VIOD's múltipla da quantidade de IOD's permanentes. Assim, cada IOD atende às requisições de mais de um VIOD.

Para avaliar o ganho de utilizar uma maior quantidade de servidores exclusivos, utilizou-se o *MPI IO Test 21* com a mesma concorrência de escritas e leituras da sessão anterior. As duas instâncias do benchmark executam com 24 clientes. O dNFSp foi inicializado com 4, 8, 12 e 16 VIOD's, com 4 servidores de dados permanentes. Para fins de comparação, apresentou-se também o desempenho do teste com a utilização do dNFSp sem

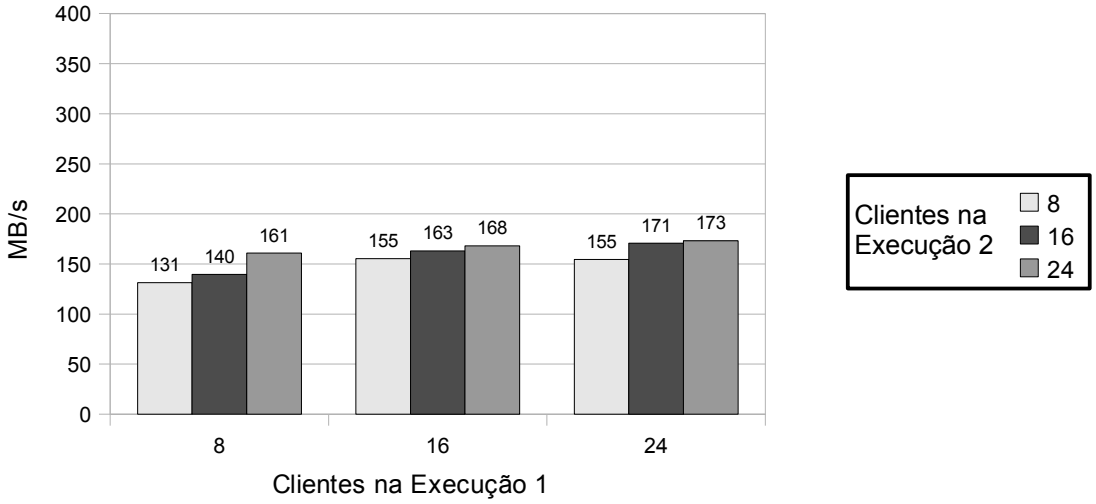


Figura 7.2: Banda Combinada da Fase 1 (Escrita/Escrita) com AppIOD's para a Execução 1.

AppIODs.

Os resultados para as operações de escrita são apresentados na Figura 7.6. O caso *4Normal* é referente à execução do teste sem AppIOD's. Enquanto a escrita concorrente no dNFSp apresentou um desempenho de $67,48 MB/s$ para a primeira instância do benchmark, a escrita concorrente com as leituras da segunda instâncias teve um desempenho de apenas $36,60 MB/s$. Com o compartilhamento de servidores de dados, o maior desempenho nas operações de leitura acaba por penalizar as operações de escrita.

Nesta execução do teste, a inclusão de 4 servidores exclusivos para a execução 1 (4AppIOD) resultou num aumento de 13% no desempenho da execução 1. Já no caso de escritas concorrentes com operações de leitura, o aumento foi de 82%.

A utilização de mais servidores exclusivos, no entanto, não apresenta um ganho de desempenho proporcional ao da utilização dos recursos. Com 16 servidores exclusivos, obteve-se um aumento de 70% e 252% para as fases 1 e 2 do teste com relação ao desempenho sem a utilização de AppIOD's.

O desempenho das operações de leitura é apresentado na Figura 7.7. Após a inserção dos servidores exclusivos, a segunda instância do benchmark acessa os IOD's permanentes com exclusividade. Uma vez que ela não faz uso dos servidores inseridos, a execução 2 não apresenta variação significativa com o aumento do número de AppIOD's. A pequena queda observada, no entanto, indica que o aumento da utilização de rede dos meta-servidores devido à maior vazão na execução 1 começa a interferir no desempenho de leitura.

7.3.3 Considerações de Desempenho com AppIOD

Através dos testes apresentados, foi possível verificar o ganho de desempenho que a utilização de servidores exclusivos pode trazer na configuração do dNFSp apresentada.

Os resultados indicam que, em situações onde o acesso aos servidores é predominantemente de operações de escrita, a inclusão de mais servidores pode apresentar um ganho de até 28% no desempenho global do dNFSp, mesmo com o gargalo existente no número

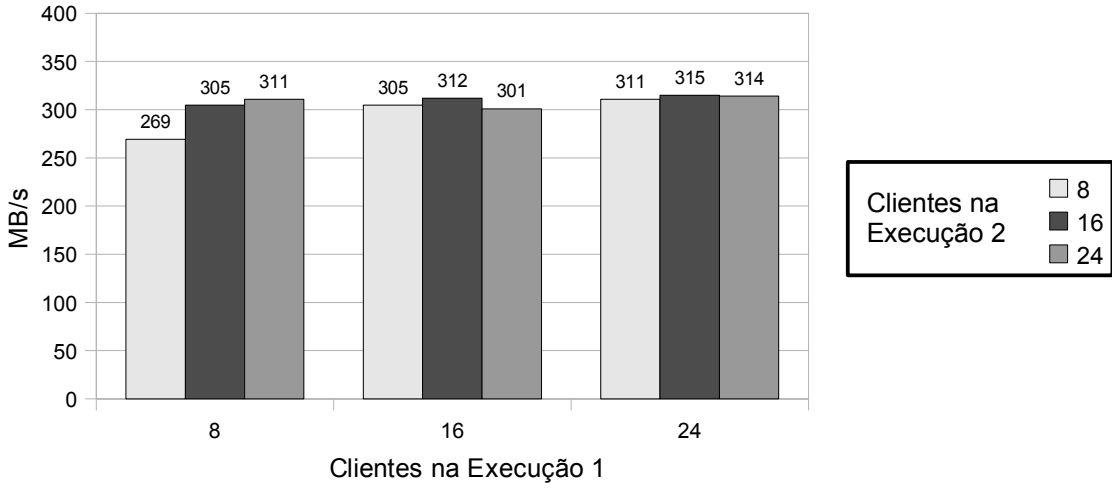


Figura 7.3: Banda Combinada da Fase 2 (Escrita/Leitura) sem utilização de AppIOD.

fixo de meta-servidores. No caso da execução concorrente de aplicações com operações de leitura e escrita, o impacto da utilização de servidores exclusivos é proporcionalmente menor, atingindo um máximo de 17%.

Observando o desempenho individual das aplicações (Figura 7.5), a combinação de operações de leitura e escrita trouxe maior ganho de desempenho para a aplicação com servidores exclusivos. A inserção de mais servidores também alterou o comportamento observado na Figura 7.6, onde a concorrência de leitura e escrita penaliza o desempenho das escritas em favor do das leituras.

A utilização de mais servidores de aplicação do que meta-servidores, no entanto, apresenta, para a aplicação reconfigurada, um ganho não proporcional à quantidade de recursos utilizados. A inserção de muitos servidores exclusivos para uma única aplicação pode, também, levar a uma saturação na capacidade de rede dos meta-servidores, tornando novas reconfigurações contraproducentes. Os testes indicam, no entanto, não é necessário restringir o sistema a efetuar apenas uma reconfiguração: é possível inserir AppIOD's para mais de uma aplicação e ainda assim melhorar o desempenho de I/O.

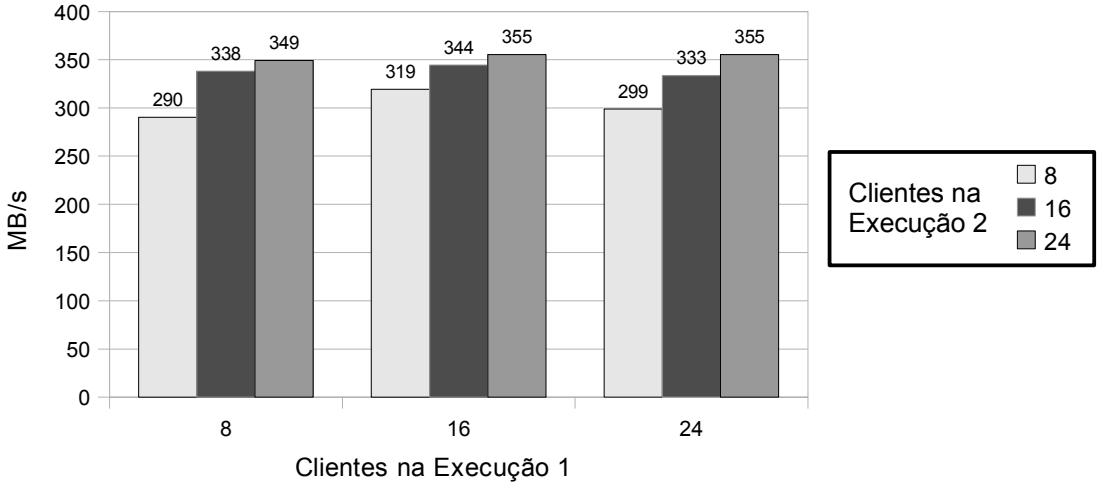


Figura 7.4: Banda Combinada da Fase 2 (Escrita/Leitura) com AppIOD's para a Execução 1.

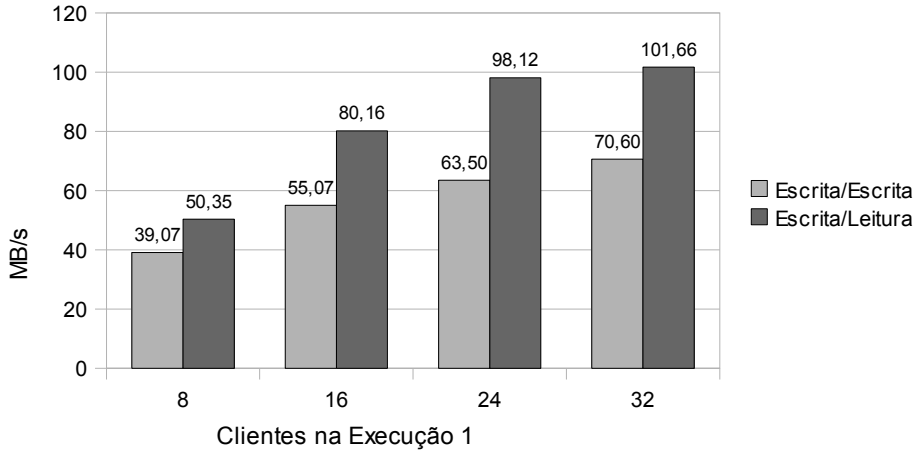


Figura 7.5: Comparativo das Vazões de Escrita da Execução 1 para as Fases 1 (Escrita/Escrita) e 2 (Escrita/Leitura) do teste. Execução 1 com 4 servidores exclusivos; Execução 2 com 32 clientes.

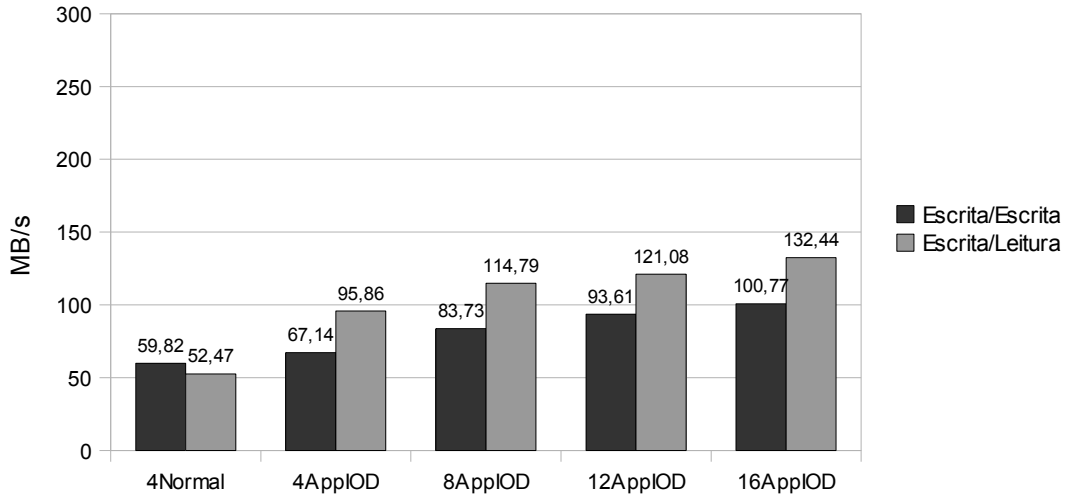


Figura 7.6: Banda de Escrita para Fases 1 e 2 do teste. 4Normal: dNFSp sem AppIOD's; 4AppIOD: dNFSp com 4 servidores permanente e 4 servidores exclusivos.

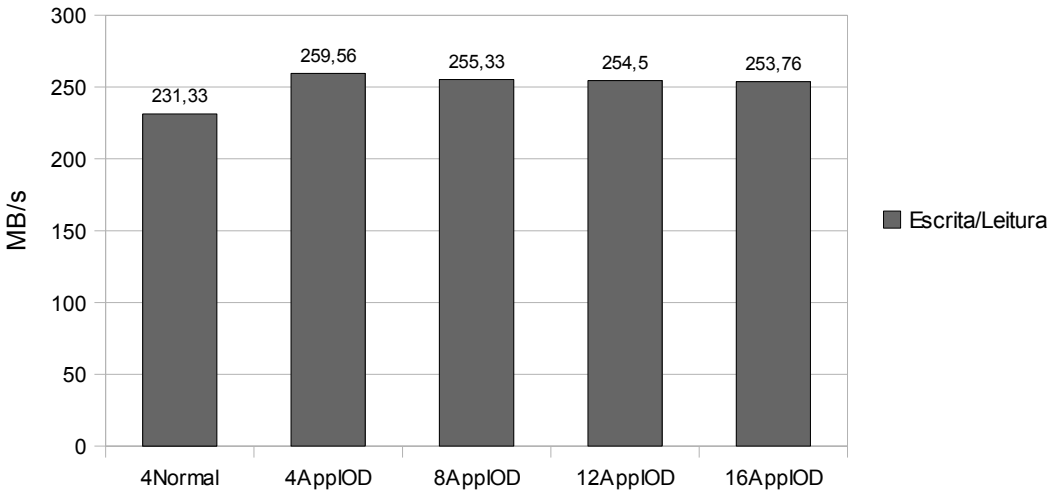


Figura 7.7: Banda de Escrita para Fase 2 (Leitura) da execução 2. 4Normal: dNFSp sem AppIOD's; 4AppIOD: dNFSp com 4 servidores permanente e 4 servidores exclusivos.

7.4 Reconfiguração Automática

Com a utilização do *pfmond*, são inseridos servidores exclusivos para melhorar o desempenho de I/O do dNFSp quando os acessos ao sistema de arquivos feitos por uma aplicação resultar em queda no desempenho de outra. A observação do desempenho da aplicação, no entanto, exige que sejam efetuadas consultas periódicas aos servidores, o que pode impactar no desempenho de I/O das aplicações.

Além disso, com a execução concorrente de aplicações com padrões de acesso a dados distintos, se faz necessário avaliar como esta influência será detectada pela ferramenta de monitoramento e que valores o *pfmond* tomará como base para avaliar se o desempenho de I/O de uma aplicação encontra-se saturado. Em detectando esta saturação e efetuando a reconfiguração dinâmica, deve-se avaliar o ganho que o aumento do número de recursos de I/O trouxe para as aplicações.

Esta seção faz a avaliação destes casos, utilizando as ferramentas descritas no Capítulo 6 para efetuar testes com comportamento temporal e seu comportamento com a utilização do dNFSp.

7.4.1 Impacto da Biblioteca PFM no dNFSp

Para avaliar o impacto causado pelo monitoramento do desempenho do sistema de arquivos, foram executados testes com a utilização da *libpfm* e do *pfmond*. A Figura 7.8 mostra a banda de I/O obtida no benchmark *MPI IO Test 21* com 24 e 48 clientes sobre um sistema dNFSp com 4 servidores. O benchmark foi executado durante um tempo máximo de 2 minutos durante os quais foram feitas, exclusivamente, operações de escrita. Foi utilizado o modo *N-N* e objetos de $1MB$. Os valores apresentados são médias de 6 execuções.

Foram avaliadas quatro situações. Primeiramente, foi medido o desempenho do dNFSp sem a biblioteca PFM. Em seguida, foram habilitadas apenas as operações de *trace*, executadas a cada operação de *read* ou *write* processada pelos meta-servidores.

O impacto da transmissão das informações de desempenho foi testado em dois casos: no primeiro, todos os clientes são registrados como pertencentes a uma única aplicação. Neste caso, o *pfmond* receberá apenas uma mensagem por meta-servidor com as informações de quantidades de dados transferidos. No segundo caso, criou-se uma aplicação por cliente, forçando a transmissão de 24 ou 48 mensagens para o *pfmond*. Em ambos os casos, a freqüência de consulta foi de 30 segundos.

Sem a interferência das operações da *libpfm*, observou-se uma banda de $147,75MB/s$ para 24 clientes e $151,01MB/s$ para 48. Os desvios padrões foram de $4,76$ e $4,45$, respectivamente. O maior desempenho com 48 clientes é indicativo de que a capacidade de rede e de disco do sistema de armazenamento não foram completamente saturada com 24 clientes. Os valores, porém, encontram-se dentro de um intervalo de tolerância de 10% e a diferença não é considerada significativa.

A utilização das operações de *trace* causou um pequeno impacto segundo estes testes. Com a transmissão de informações para o *pfmond*, os desempenhos foram de $145,88MB/s$ e $143,95$, também dentro do intervalo de tolerância.

O caso da transmissão de informações sobre várias aplicações para o *pfmond* também não apresentou impacto significativo no desempenho. Para os dois números de clientes testados, a banda de I/O obtida foi de $147,11MB/s$ e $146,89MB/s$, respectivamente, com desvios padrão de $5,18$ e $3,56$. Estes valores encontram-se dentro do intervalo de tolerância de 10% dos valores sem a utilização da biblioteca.

Os testes indicam que a quantidade de dados transmitida para o *daemon* responsável pela análise dos dados não compromete o desempenho de I/O quando a carga nos servidores é dominantemente de operações de escrita. Esta situação é aquela que causa maior ocupação da capacidade de rede dos meta-servidores. Uma vez que a transmissão de uma ou muitas mensagens para o *pfmond* não causou impacto significativo, a pequena diferença entre os casos com e sem a utilização da biblioteca deve ser atribuída à momentânea interrupção do serviço de I/O para a obtenção dos dados de desempenho.

Devido a restrições da biblioteca RPC da época de seu desenvolvimento, o meta-servidor é constituído de apenas uma *thread*. Visando paralelizar o preprocessamento feito antes da transmissão das mensagens ao *pfmond*, foi implementada uma *thread* responsável exclusivamente pela comunicação com o *daemon*. Devido à concorrência por algumas estruturas de dados, as operações de *trace* e esta *thread* devem obter os *locks* associados antes de alterar as informações de desempenho. Desta forma, o atendimento das requisições de I/O nos meta-servidores é interrompido sempre que há transmissão dos dados para o *pfmond*.

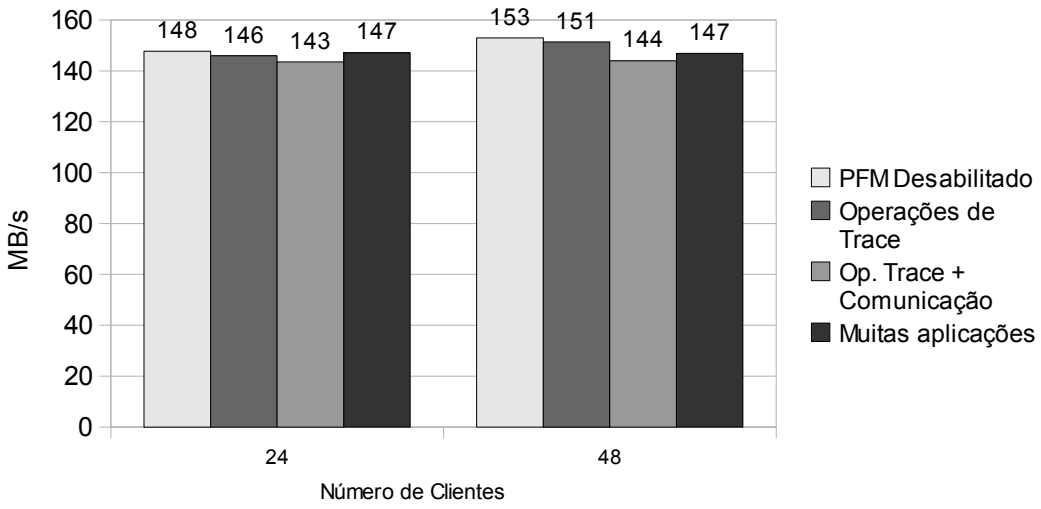


Figura 7.8: Impacto da Biblioteca PFM no desempenho do dNFSp

Para avaliar o impacto da freqüência de consulta aos meta-servidores, foi feito um teste variando o intervalo de consulta do *pfmond*. Os resultados são apresentados na Figura 7.9. Foram testados intervalos de 1 a 60 segundos para 24 e 48 clientes. Para cada caso, avaliou-se o desempenho de I/O para quando todos os clientes constituem uma única aplicação e para quando cada cliente é registrado como uma aplicação independente.

Nota-se que, para 24 clientes, o desempenho sofre pouca variação tanto no caso com apenas uma aplicação quanto no caso para várias. A utilização de intervalos mais curtos se mostra mais impactante no caso com 48 clientes. As diferenças, no entanto, são consideradas pouco significativas: com intervalos de 1 segundo, obteve-se um desempenho de 154,50 MB/s, enquanto intervalos de 60 segundos apresentaram desempenho de 160,43 MB/s.

A análise de desempenho das aplicações não causa, portanto, impacto significativo no desempenho de I/O do dNFSp.

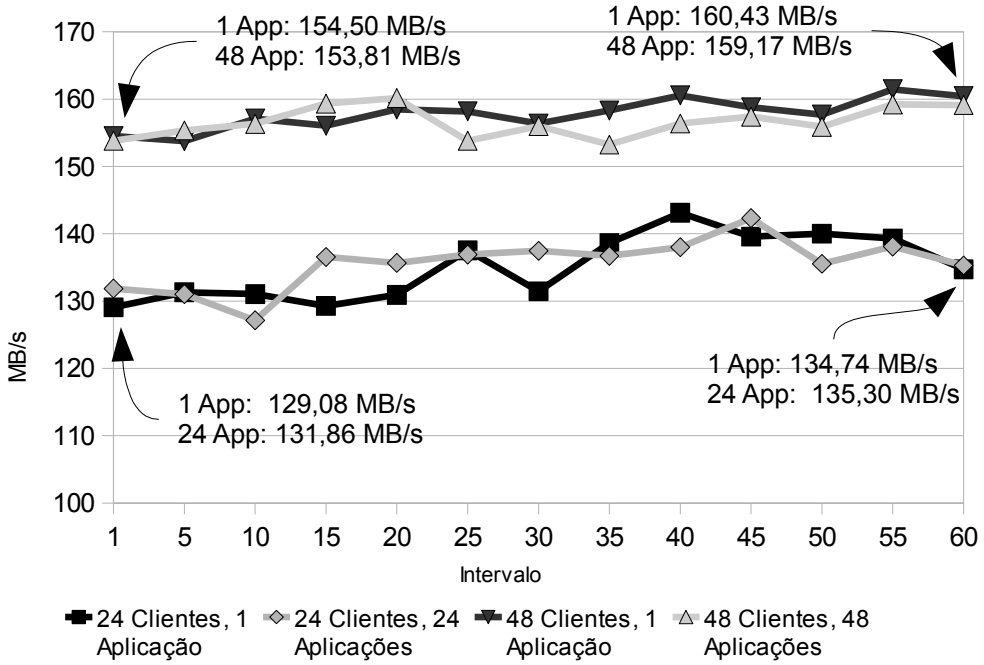


Figura 7.9: Desempenho do dNFSp com diferentes intervalos de avaliação

7.4.2 Histórico de Execução e Estabilidade do Desempenho

Para avaliar a métrica apresentada no Capítulo 5 foram executados testes com a execução de uma única aplicação no sistema de arquivos. A aplicação em questão representa a classe *I/O Bound* e foi executada com 24 clientes efetuando escrita de objetos de 1MB durante 300 segundos. Após este tempo, o VSS permite mais 30 segundos de execução antes de terminar o benchmark nos nós com o comando *kill*.

A Figura 7.10 apresenta o histórico da banda de I/O de 3 execuções distintas do teste. O intervalo de consulta aos meta-servidores utilizado foi de 10 segundos. Foi utilizado um intervalo de tolerância de 10%. Estão marcados, considerando períodos de estabilidade de 1 e 2 minutos, os períodos nos quais o sistema passa a ser considerado instável (*(1)* e *(3)*) e os momentos onde o sistema passa a ser considerado estável e saturado (*(2)* e *(4)*).

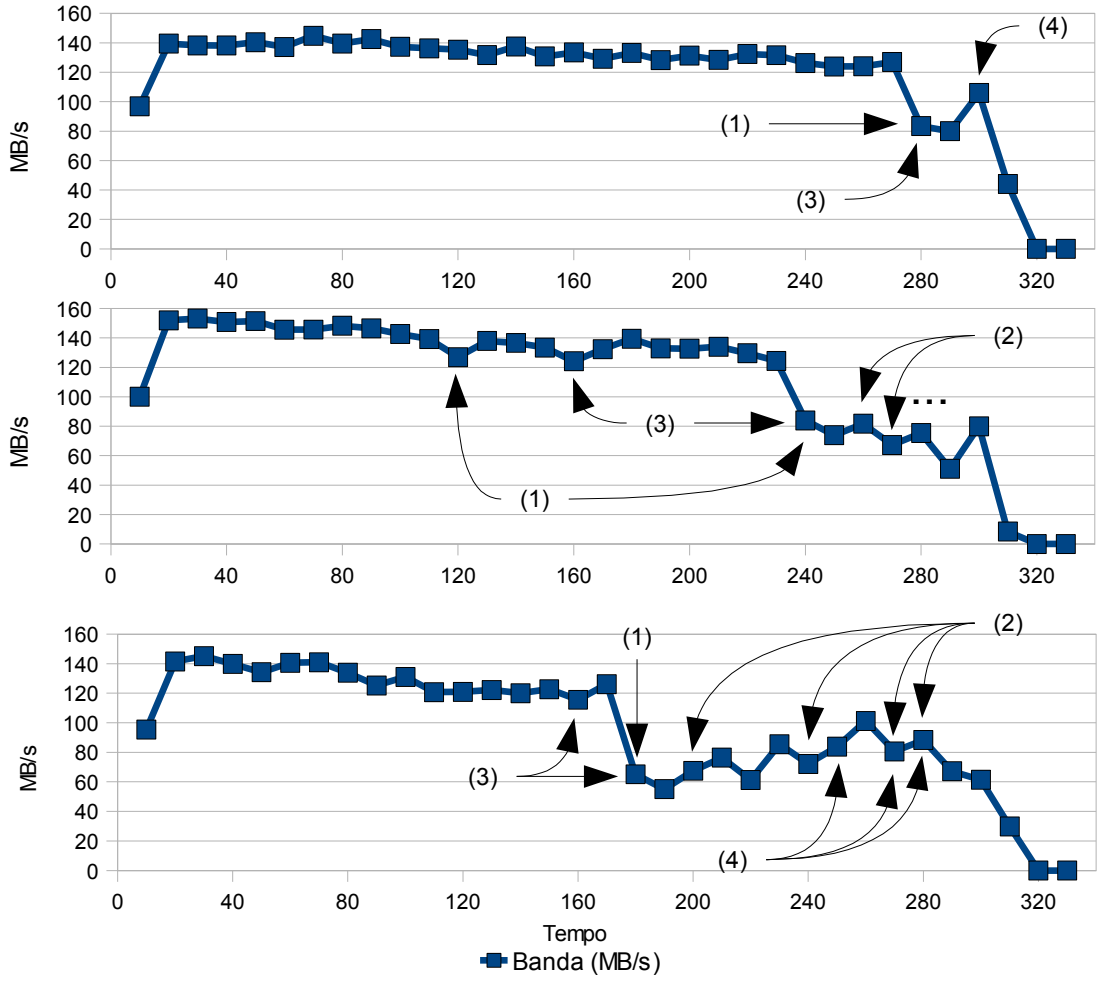


Figura 7.10: Histórico de Banda de I/O para única aplicação, $\Delta = 10$.
(1): Instabilidade com $S = 1min$;
(2): Estável e Saturado com $S = 1min$;
(3): Instabilidade com $S = 2min$;
(4): Estável e Saturado com $S = 2min$

Os três casos apresentam um período inicial onde há estabilidade no desempenho do benchmark. Após este período, o desempenho apresenta queda e, eventualmente, volta a se estabilizar com um desempenho menor. Com a métrica de estabilidade e saturação

apresentada anteriormente, as variações de desempenho são detectadas pela ferramenta.

No primeiro e segundo caso, a queda ao final da execução pode ser explicada devido ao comportamento do benchmark. O teste é executado com um número máximo de objetos de 2048, de forma a gravar arquivos do tamanho máximo permitido pelo dNFSp. Não são utilizadas barreiras entre as operações, o que pode permitir que alguns clientes ocupem o sistema de arquivos em detrimento a outros, concluindo a gravação antes. Desta forma, a queda observada ao final da execução é resultado de uma menor quantidade de clientes ainda em operação, subutilizando os recursos disponíveis.

O terceiro caso, no entanto, apresenta esta queda após 180 segundos de execução. Como o desempenho volta a aumentar nos intervalos seguintes, assume-se que este comportamento foi resultado de interferências externas (e.g. rede).

Estas variações, no entanto, não devem forçar a ferramenta a iniciar uma reconfiguração, uma vez que não estão associadas a uma mudança de comportamento de outra aplicação. A ferramenta evita a configuração se, quando uma aplicação A_i for detectada *saturada*, não houver outra aplicação A_j que também esteja saturada ou cujo desempenho se tornou instável durante o período de estabilidade.

Há uma maior variação de desempenho ao longo do tempo quando, em clientes com vários processadores ou cores, cada processo efetua suas operações de I/O de forma independente. A Figura 7.11 apresenta o desempenho do benchmark executando 8 processos por cliente. Para que cada cliente escreva, no total, a mesma quantidade de dados que no teste anterior, foi utilizado o mesmo tamanho de objetos de 1MB, mas o número de objetos foi definido como 256.

O comportamento com 4 processos por cliente é exibido na Figura 7.12. Esta configuração também apresentou instabilidade mesmo com o acesso exclusivo ao sistema de arquivos.

Uma vez que os IOD's não fazem distinção entre as requisições, seja por arquivo ou por cliente de origem, e que os meta-dados dos arquivos são mantidos na memória do servidor, a instabilidade deve ser causada pela interação dos processos nos clientes. A maior quantidade de clientes, aliada às políticas de *write-behind* do cliente NFS presente no sistema Linux acabam produzindo este desempenho variável do ponto de vista do *pfmond*.

Devido a este comportamento por parte dos clientes, o desempenho de I/O da execução com múltiplos processos de I/O por nó obteve desempenhos inferiores aos dos casos com um único processo. Para 8 processos por nó, obteve-se uma banda de $90.66 MB/s$, enquanto o caso de 4 processos por nó apresentou desempenho de $86.21 MB/s$. Com apenas um processo por nó, o desempenho observado foi de $119.34 MB/s$.

Apesar das variações, o *pfmond* foi capaz de detectar como desempenho máximo da aplicação valores próximos aos observados na Seção 7.3. A Figura 7.13 apresenta a média e o desvio padrão dos valores detectados como máximos em 6 execuções do teste. Nos 3 casos, o valor máximo foi detectado próximo a $140 MB/s$, próximo ao valor máximo obtido com os testes anteriores.

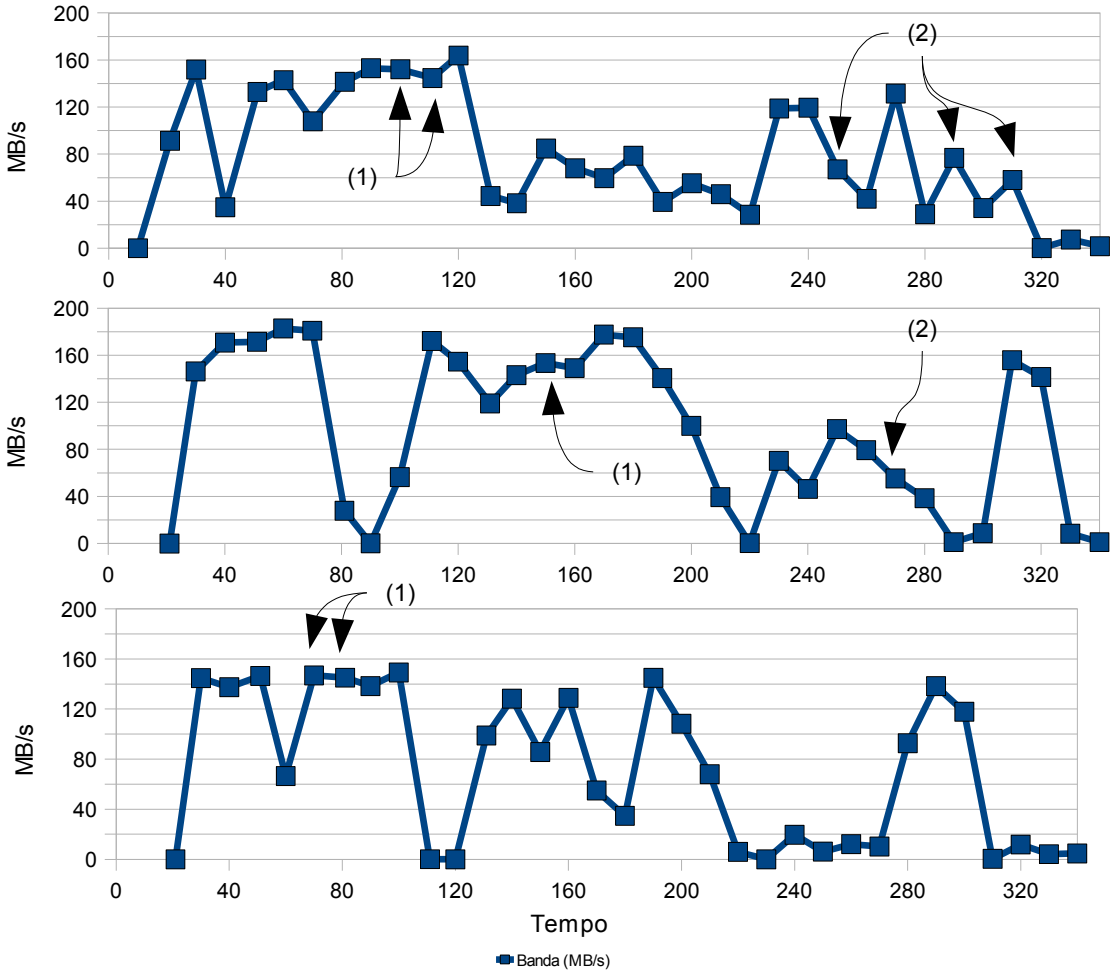


Figura 7.11: Histórico de Banda de I/O para única aplicação, 8 processos por cliente, $\Delta = 10$.
(1): Estável com $S = 1min$;
(2): Estável e Saturado com $S = 1min$

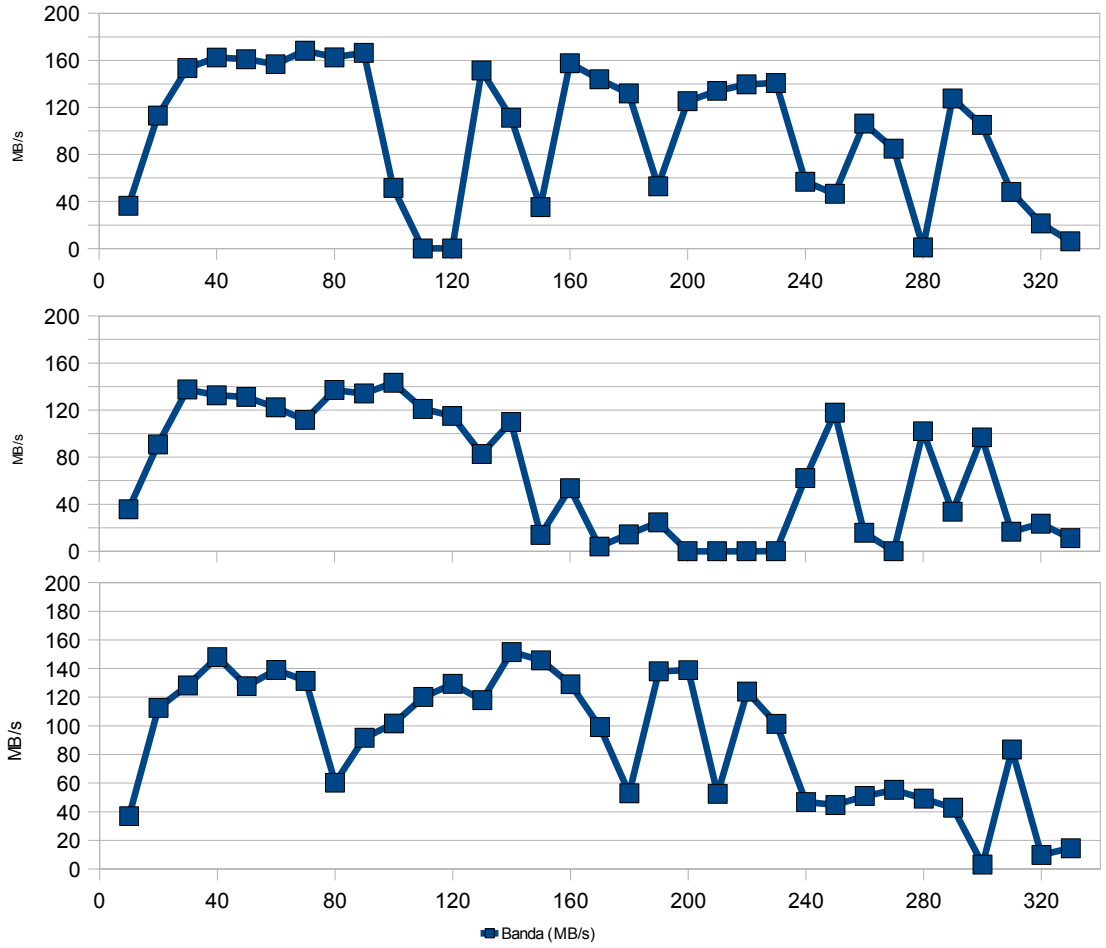


Figura 7.12: Histórico de Banda de I/O para única aplicação, 4 processos por cliente

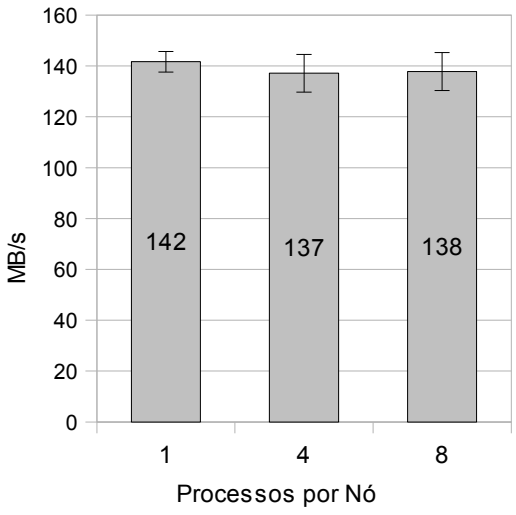


Figura 7.13: Desempenhos Máximos conforme detectados pelo *pfmond*

7.4.3 Desempenho com a Reconfiguração Automática

A inserção de servidores exclusivos aumenta o desempenho de I/O das aplicações ao longo de sua execução. Para avaliar o ganho que a reconfiguração dinâmica obtém, foi utilizado o *benchmark MPI IO Test 21*, em conjunto com o *Very Simple Scheduler* para simular um traço extraído do cluster Atlas.

O diagrama de Gantt do cenário utilizado é apresentado na Figura 7.14. Este traço, detalhado na Tabela 7.1, apresenta um período de 1h20min em cuja primeira hora há utilização de apenas 10% dos recursos do cluster. Devido à quantidade de nós disponíveis para simulação, apenas os trabalhos 0 e 1 são utilizados.

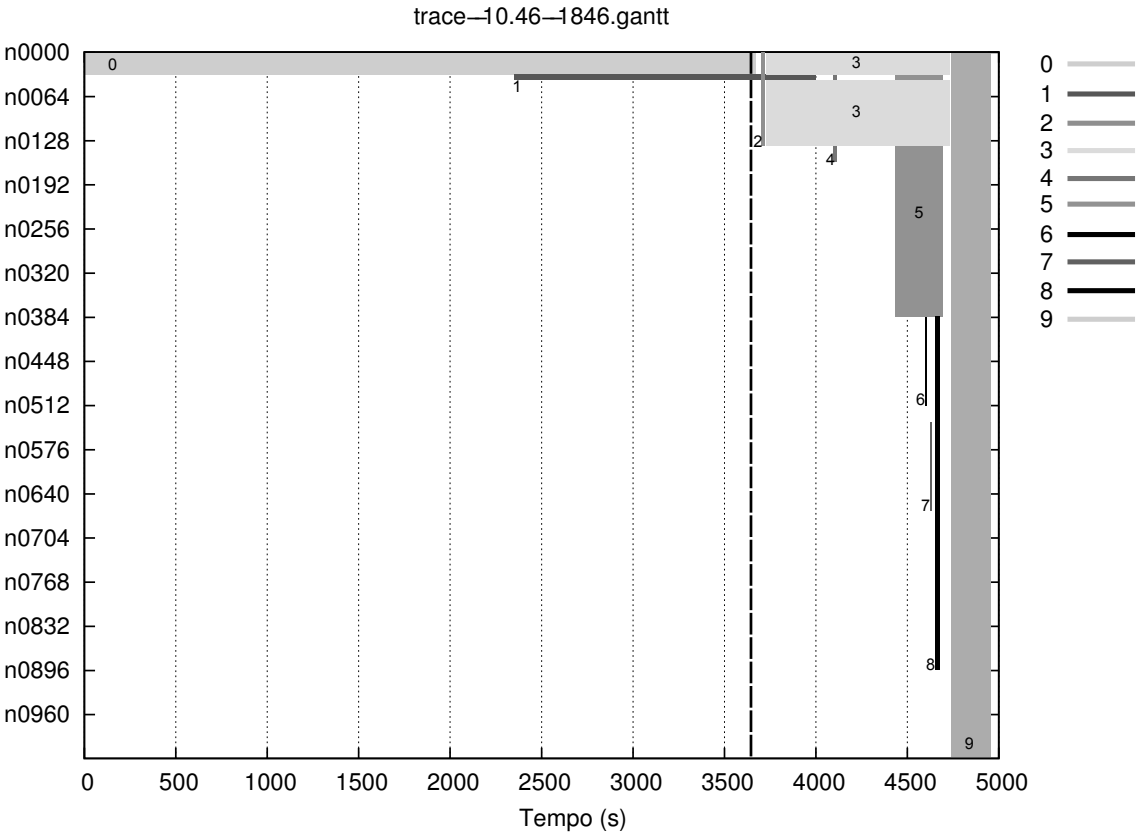


Figura 7.14: Diagrama de Gantt para um intervalo de 1h20min. do traço LLNL-Atlas

Uma vez que o objetivo é estudar a ferramenta quando há concorrência no sistema de arquivos, o traço é simulado apenas a partir de seus 38 minutos. Isto visa permitir à aplicação 0 iniciar e, em seguida, ter seu desempenho de I/O influenciado pela execução da aplicação 1.

A aplicação 1 executa um código da classe *I/O Bound* para a operação de escrita. Já a aplicação 0 é mapeada para uma com comportamento temporal, também com operações de escrita (*Checkpoint I/O*). Estas classes de aplicação foram escolhidas para avaliar o comportamento da reconfiguração e a detecção de saturação dos recursos de I/O quando há variação na carga ao longo da execução.

Para a aplicação 0, são testados diferentes tamanhos de objetos e diferentes tempos de inatividade, o que resulta em distintos valores para β. Para cada tamanho de objeto e intervalo de inatividade, foram testados os desempenhos de I/O quando a aplicação é

Id	Início	Tempo de Execução	Número de Recursos
0	0	1h 0min 28seg	256
1	39min	27min 28 seg	64
2	1h 2min	16seg	1024
3	1h 2min	16min 37seg	1024
4	1h 9min	1seg	128
5	1h 14min	4min 18seg	2048
6	1h 16min	6seg	1024
7	1h 17min	8seg	2048
8	1h 17min	28seg	4096
9	1h 18min	3min 36seg	8192

Tabela 7.1: Traço Atlas-10.46–1846

Tamanho de Objeto	I=10	20	40	60	80
128MB	0,97	0,94	0,92	0,86	0,82
16MB	0.81	0.69	0.52	0.42	0.36
8MB	0,68	0,52	0,35	0,26	0,21

Tabela 7.2: Valores de $\beta(A_0)$ para intervalos de inatividade I de 10, 20, 40 60 e 80 segundos– 8 processos por nó

executada com exclusividade sobre o dNFSp e, com base nos tempos informados pelo *MPI IO Test 21*, calculados os valores para $\beta(A_0)$. A Tabela 7.2 apresenta os valores para os tamanhos de objeto testados. Este valor é fornecido ao *pfmond* via um arquivo de entrada e representa o comportamento da aplicação independente das interferências causadas por outras.

Já os trabalhos de 2 a 9 são considerados trabalhos virtuais. Estes trabalhos não executam nenhuma aplicação e são apenas utilizados para, depois da execução das aplicações reais, avaliar o impacto causado pela reconfiguração de I/O no cenário.

Os testes foram executados de duas maneiras distintas: a versão *multi-core* e a *single-core*. Na versão *multi-core*, cada nó do cluster executa 8 instâncias do benchmark, cada uma gravando objetos de 128, 16 ou 8MB. Na versão *single-core*, é utilizada apenas uma instância do benchmark em cada nó, mas o tamanho de objeto é ajustado para que, em cada fase de I/O, seja gravada a mesma quantidade de dados que na versão *multi-core*. Foram avaliados, para o caso *single-core*, tamanhos de objeto de 1GB ($8 \times 128MB$) e 64MB ($8 \times 8MB$). Os valores para β destes casos são apresentados na Tabela 7.3.

Tamanho de Objeto	I=10	20	40	60	80
1GB (8 x 128MB)	0,97	0,93	0,90	0,86	0,79
64MB (8 x 8MB)	0.61	0.44	0.28	0.20	0.16

Tabela 7.3: Valores de $\beta(A_0)$ para intervalos de inatividade I de 10, 20, 40 60 e 80 segundos – 1 processos por nó

7.4.4 Impacto da Reconfiguração no Cenário Multi-core

Esta seção apresenta a avaliação da reconfiguração dinâmica quando as aplicações executadas utilizam vários processos escritores. A Figura 7.15 apresenta os desempenhos para as aplicações 0 e 1, conforme o traço apresentado, quando a aplicação de identificador 0 (A_0) utiliza objetos de 128MB. A aplicação A_1, por sua vez, utiliza objetos de $1MB$ e escreve ao todo 2GB de dados por instância do benchmark – neste caso, 128GB. Devido à quantidade de processadores utilizados por A_0 ser maior do que a utilizada por A_1, aliado aos valores de β de A_0, a inserção de servidores exclusivos é sempre feita para A_0.

São apresentados, para comparação, os desempenhos obtidos na execução das aplicações sem a inserção de servidores exclusivos. Foram utilizados 4 servidores para o sistema de arquivos dNFSp. No momento da reconfiguração, foram utilizados 4 nós do sistema como AppIOD.

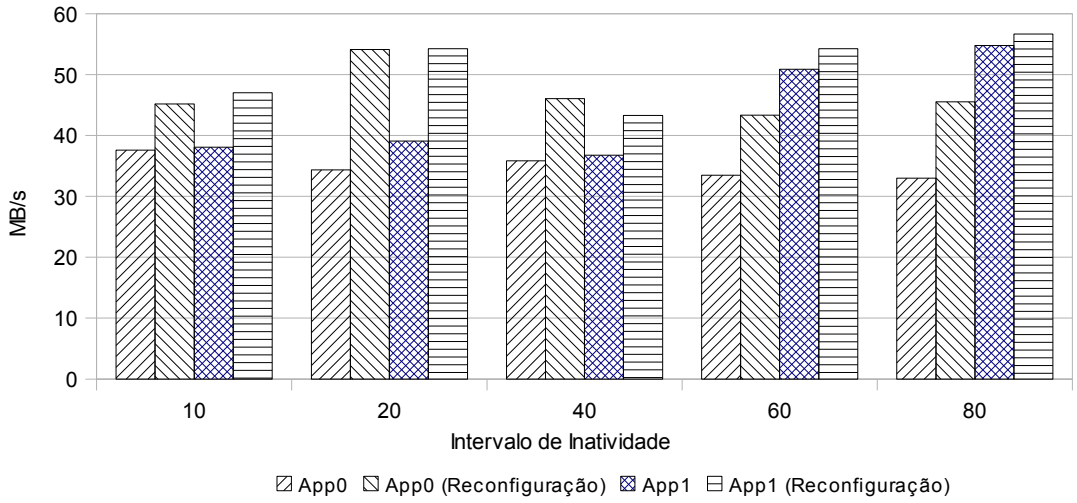


Figura 7.15: Desempenho de I/O para as aplicações 0 e 1 com e sem servidores de aplicação. Execução *multi-core*, objetos de 128MB.

Observa-se que, para os intervalos de inatividade testados, houve aumento de desempenho em ambas aplicações quando é feita a inserção de automática de servidores de aplicação. Com intervalos de 60 e 80, no entanto, o ganho apresentado por A_1 com a inserção de AppIOD's para A_0 mostra-se menor do que com os intervalos de 10 a 40. Isto indica que, com tais intervalos, A_1 já é capaz de aproveitar-se dos períodos de inatividade de A_0 e obter um maior desempenho.

A_0, por sua vez, apresenta ganho com a utilização dos recursos exclusivos em todos os intervalos. Os valores dos casos com e sem utilização de AppIOD's apresentam, no entanto, pouca variação em função do período de inatividade. Isto indica que a *vazão de escrita*[3] da aplicação aumenta com o maior tempo de espera.

As causas de tal aumento do desempenho de A_0 podem ser observadas nas Figuras 7.16 e 7.17. Estas figuras apresentam o histórico de desempenho das duas aplicações em 3 execuções do teste com intervalos de 40 e 80 segundos. Pode-se notar que, para

[3]**Vazão de Escrita:** Desempenho de I/O levando em conta apenas o tempo gasto com as operações de leitura e escrita

intervalos de 40 segundos de inatividade em A_0, o desempenho das duas aplicações apresenta grande instabilidade quando comparados aos observados no caso de intervalos de 80 segundos. O maior desempenho de A_0 é reflexo desta estabilidade. A_1 acaba por também apresentar melhor desempenho quando A_0 inicia suas fases de menor atividade.

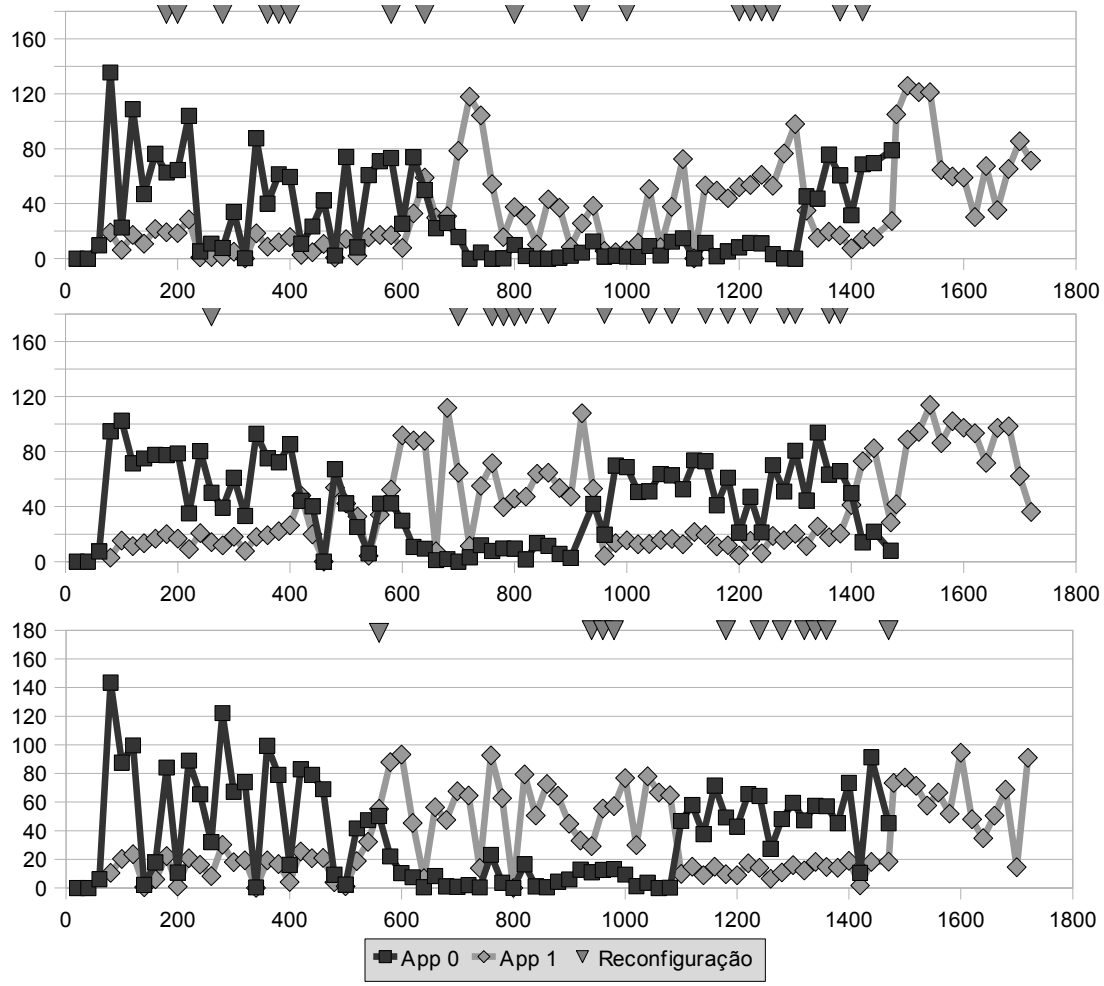


Figura 7.16: Históricos de banda para as aplicações 0 e 1 no dNFSp, sem a utilização de AppIOD. Inatividade de 40 segundos na Aplicação 0.

A Figura 7.18 apresenta este teste para objetos de 16MB em A_1. Devido ao menor valor de β, quando A_0 apresenta intervalos de inatividade de 40 segundos, a ferramenta seleciona para reconfiguração a aplicação A_1. Devido ao menor tamanho de objetos, o desempenho de I/O de A_0 diminui em função dos maiores intervalos de inatividade. Com a presença de servidores exclusivos, este desempenho apresentou aumento de mais de duas vezes para os casos de teste.

Ao contrário dos casos de teste com objetos de $128MB$, o aumento no intervalo de inatividade em A_0 não representou aumento no desempenho de A_1 quando as duas aplicações compartilham servidores de dados. A utilização de AppIOD's para estes casos permitiu corrigir esta queda de desempenho para os intervalos de 60 e 80 segundos.

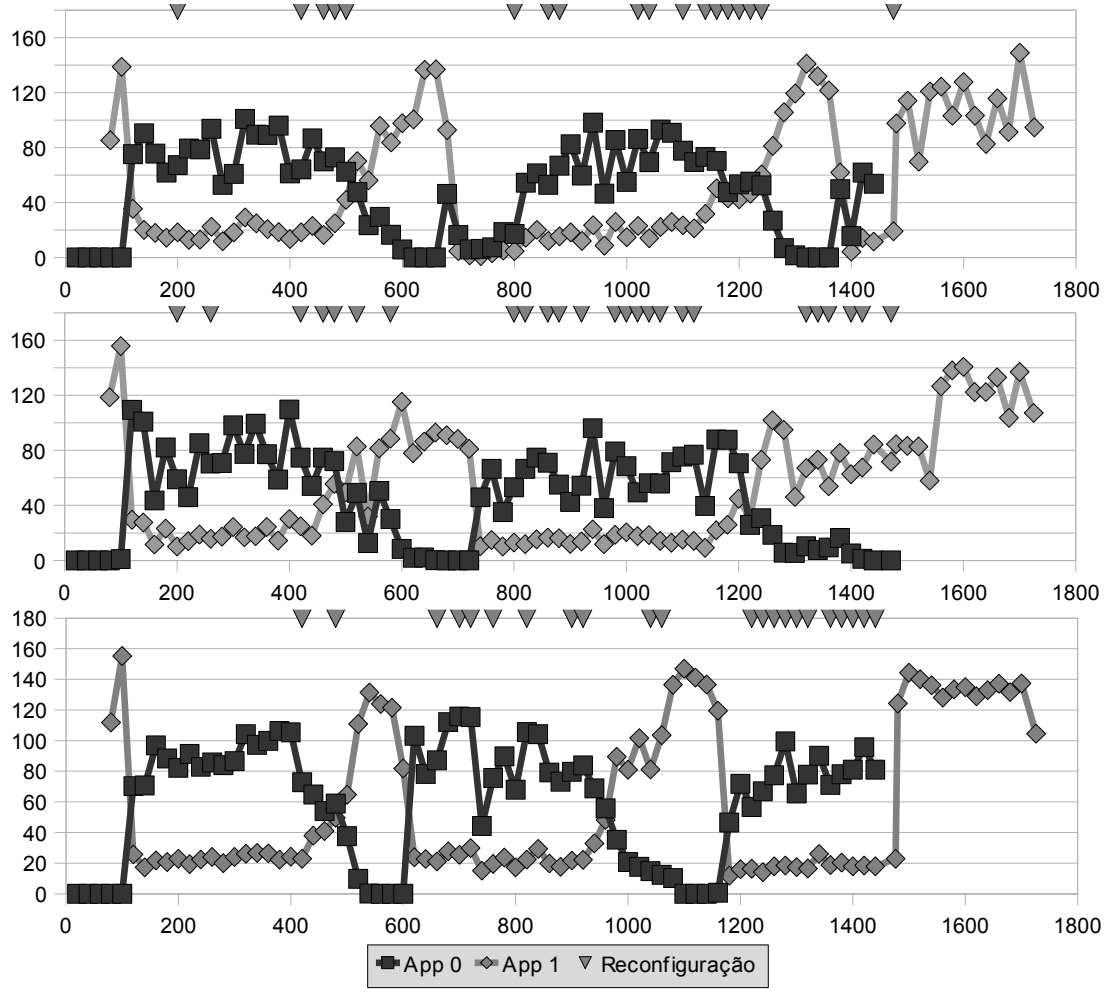


Figura 7.17: Históricos de banda para as aplicações 0 e 1 no dNFSp, sem a utilização de AppIOD. Inatividade de 80 segundos na Aplicação 0.

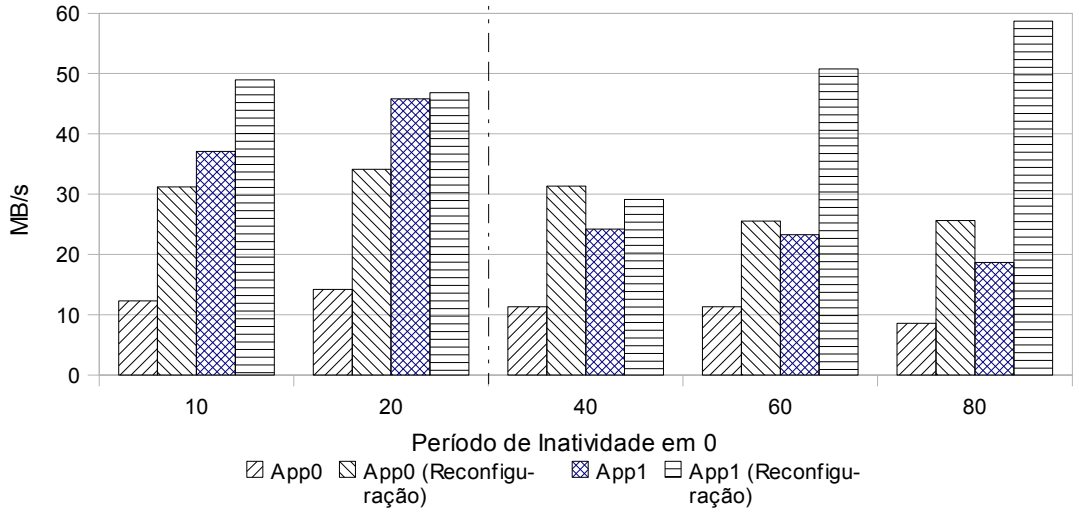


Figura 7.18: Desempenho de I/O para as aplicações 0 e 1 com e sem servidores de aplicação. Execução *multi-core*, objetos de 16MB.

7.4.5 Utilização Single ou Multi-core

Como visto na Seção 7.4.2, a utilização de diversos processos por nó causa variação no desempenho de I/O durante a execução da aplicação. Esta seção avalia o comportamento desta instabilidade e sua influência no acesso ao sistema de arquivos com a reconfiguração dinâmica.

A Figura 7.19 apresenta o desempenho de I/O para as aplicações do cenário descrito anteriormente. São apresentados lado a lado, para cada intervalo de inatividade em A_0, os desempenhos de A_0 e A_1 com execução *single-core* e *multi-core*.

Para a execução *multi-core* de A_0, são executados 8 processos por cliente, efetuando, cada, a escrita de no máximo 16 objetos de 128MB. Entre cada objeto escrito, é feita uma barreira de sincronização para garantir o comportamento temporal da aplicação distribuída.

No caso de execução *single-core*, um único processo é executado por cliente, efetuando escrita de no máximo 128 objetos de 128MB. A cada 8 objetos escritos, é efetuada uma barreira, de forma a imitar o comportamento da execução *multi-core*.

A aplicação A_1, no caso da execução *multi-core*, também utiliza 8 processos por cliente, porém sem a utilização de barreiras. Cada processo escreve até 2048 objetos de 1MB. No caso da execução *single-core*, o benchmark é repetido 8 vezes, com cada cliente escrevendo a mesma quantidade de objetos em arquivos distintos.

Para todos os casos testados, a estratégia *single-core* mostrou-se mais vantajosa para A_0 do que a estratégia *multi-core*. A diferença em desempenho, no entanto, diminui com intervalos de 40 segundos, voltando a crescer com intervalos de 60 e 80 segundos. Como a queda de desempenho é observada também para A_1, tanto na estratégia multi quanto na *single-core*, esta queda deve ser conseqüência de características do sistema.

A estratégia *single-core* para a aplicação A_1, no entanto, apresentou ganho apenas para o intervalo de 10 segundos em A_0. O desempenho pior do que o da estratégia *multi-core* pode ser causado pela repetição do benchmark, o que induz um comportamento temporal devido às barreiras na finalização da aplicação MPI.

A estratégia *multi-core*, por sua vez, apresentou um desempenho virtualmente estável para A_0. Esta estabilidade, com o aumento no tempo de inatividade, implica um aumento no desempenho bruto de escrita. Este aumento não parece impactar A_1 na execução *multi-core*, uma vez que seu desempenho aumenta a partir de intervalos de 40 segundos.

Na Figura 7.20 são apresentados os desempenhos de A_0 e A_1 em suas execuções multi e *single-core*. Neste caso, porém, A_0 utiliza objetos de $8MB$, efetuando a escrita de 64MB por nó do cluster. Devido aos valores de $\beta\,(A_0)$ para este tamanho de objeto, conforme apresentado nas tabelas 7.3 e 7.2, para intervalos de 10 e 20 segundos, o número maior de clientes força a utilização de servidores exclusivos para A_0. Com intervalos a partir de 40, no entanto, a aplicação A_1, por ter potencial de escrever mais dados, acaba por ter seu I/O reconfigurado.

Independente da aplicação selecionada, a estratégia *single-core* apresentou vantagem em comparação à *multi-core* para A_0. Já A_1 apresentou melhor desempenho com a estratégia *single-core* para os intervalos de 10 e 20 segundos. A partir de 40 segundos de inatividade em A_0, a execução *multi-core* apresenta vantagem, sendo capaz de utilizar melhor os períodos de inatividade de A_0. Nestes intervalos, a utilização de AppIOD resultou em desempenho próximo a $52MB/s$ para os três casos.

Por estes resultados, pode-se concluir que, para aplicações com comportamento temporal, a agregação de operações de escrita dos vários processos residentes em um mesmo traz um aumento significativo de desempenho de I/O da aplicação. Aplicações que fazem

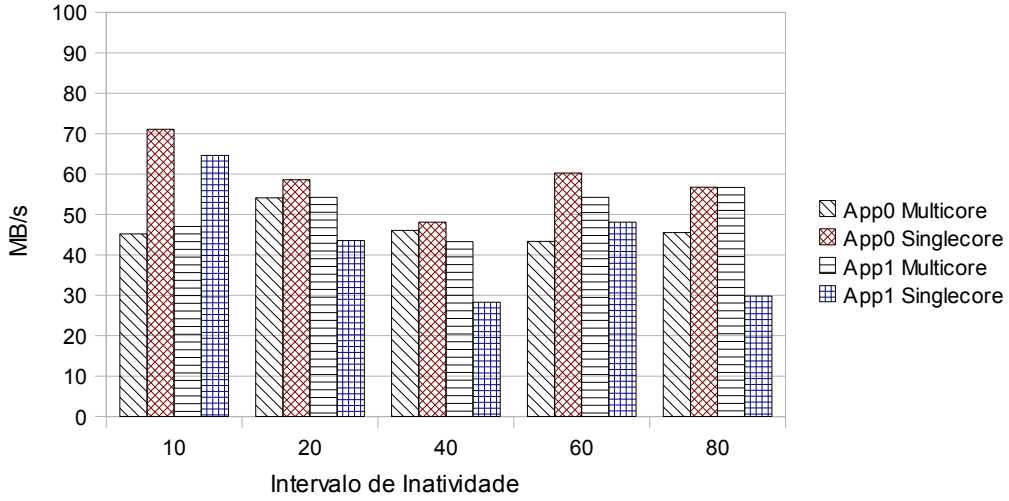


Figura 7.19: Desempenho de I/O para aplicações 0 e 1 com execução *Multi-core* e *Single-core* – 8 objetos de 128MB por cliente

escritas longas e de forma independente, como é o caso das aplicações I/O Bound e End-of-Execution Output, ainda apresentam melhor desempenho em um cenário com muitos acessos concorrentes.

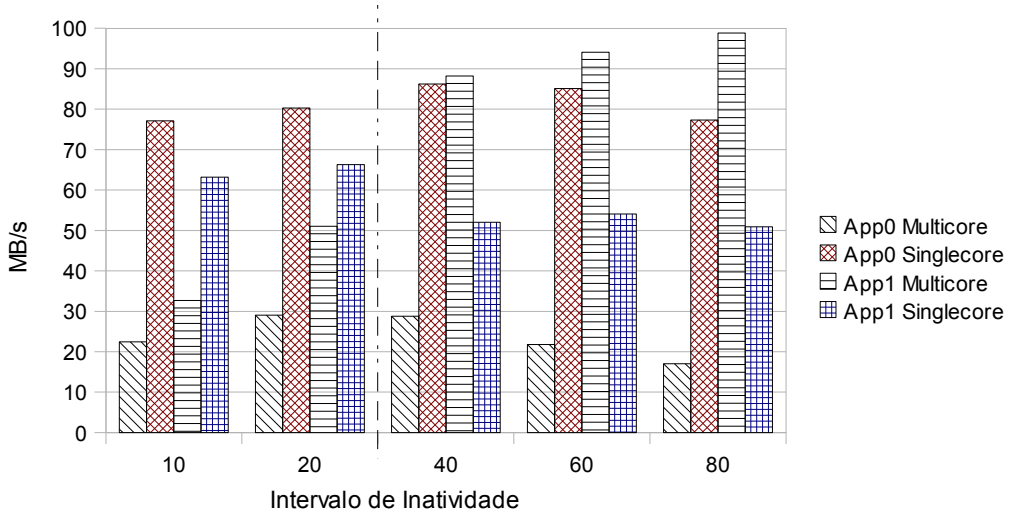


Figura 7.20: Desempenho de I/O para aplicações 0 e 1 com execução *Multi-core* e *Single-core* – 8 objetos de 8MB por cliente

8 CONSIDERAÇÕES FINAIS

Clusters de alto desempenho necessitam de sistemas de armazenamento de dados que possuam alta capacidade e vazão. O dimensionamento deste sistema, no entanto, é bastante dificultado pela variedade de aplicações executadas neste ambiente.

Uma característica bastante importante destas aplicações é o comportamento temporal. Ao longo de suas execuções, diversas aplicações intercalam fases onde poucos acessos são feitos ao sistema de armazenamento com fases que fazem acesso intensivo ao sistema de armazenamento.

Com a execução concorrente de aplicações, a intersecção de fases de I/O intenso de várias aplicações pode levar a uma grande contenção no acesso aos dados, prejudicando todas as aplicações. Uma maneira de evitar isto é a adaptação do sistema de armazenamento, permitindo seu crescimento ou encolhimento conforme a demanda por I/O.

No dNFSp, a utilização de servidores exclusivos por aplicação (AppIOD) permite aumentar o desempenho de I/O das aplicações que efetuam escritas no sistema de arquivos. A estratégia de servidores exclusivos, no entanto, utiliza recursos originalmente destinados ao processamento. Desta forma, é preciso definir uma estratégia que permita a melhoria de desempenho e utilize estes recursos apenas quando necessários.

Neste trabalho, foi apresentado um modelo de reconfiguração que utiliza servidores exclusivos para desafogar os servidores de dados permanentes do dNFSp sempre que o início de fases de I/O de uma aplicação interferir com o desempenho de outras. Foi apresentada uma avaliação do modelo com um cenário observado em um cluster de produção. Para este cenário, foram avaliadas situações com aplicações de diferentes necessidades de I/O e com diferentes estratégias de distribuição de tarefas.

Os resultados mostram que a utilização de AppIOD's no dNFSp pode aumentar o desempenho em até 28% quando o acesso aos servidores é predominantemente de escrita e até 17% quando há fluxos de escrita e leitura sobre o sistema de arquivos.

A execução de aplicações com comportamento temporal mostrou que tal característica causa grande impacto no desempenho de I/O. Os desempenhos observados sem a utilização de AppIOD's foi de no máximo $87MB/s$ para objetos de $128MB$ e $60MB/s$ para objetos de $16MB$. Com a detecção desta queda e a utilização de servidores exclusivos, foi possível melhorar o desempenho do sistema de 15% a 47% (intervalos de 60 e 20 segundos de inatividade na *execução 0*, respectivamente) para objetos de $128MB$ e de 126% 198% (60 e 80 segundos) para objetos de $16MB$.

Também foi avaliado o impacto da reconfiguração quando as aplicações são executadas com diversos (execução *multi-core*) ou um único processo de I/O em cada cliente (execução *single-core*). Foi demonstrado que, para aplicações com comportamento temporal, a utilização de servidores exclusivos obteve um melhor desempenho com a estratégia *single-core*. A execução de um único processo de I/O por nó apresentou desempenho

de 8% a 57% superior ao caso multicore (intervalos de 20 e 10 segundos, respectivamente).

Já para aplicações que fazem escritas sem coordenação de acessos, como o caso das aplicações *I/O Bound*, a estratégia multi-core apresentou melhor desempenho quando executados concorrentemente com aplicações de menor relação β. Nestes casos, observou-se desempenhos até 90% superiores ao caso *single-core* para objetos de 128*MB* e 94% para objetos de 8*MB*.

Do ponto de vista do desempenho total do sistema (banda combinada de I/O das aplicações), no entanto, a estratégia *single-core* apresentou, predominantemente, melhor desempenho sobre a *multi-core*, apesar do maior desempenho para a aplicação *I/O Bound*.

A reconfiguração dinâmica é, portanto, capaz de aumentar o desempenho do sistema de armazenamento mesmo na presença de contenções causadas por comportamentos temporais por parte das aplicações. Porém, a estratégia mostra-se sensível ao comportamento da aplicação com relação à quantidade de processos de I/O presentes em cada nó. Segundo os testes, a utilização de apenas um processo de I/O por nó apresentou melhor desempenho do que a utilização de vários.

Como trabalhos futuros, propõe-se estudar outros cenários de testes, permitindo a execução de maior quantidade de aplicações, diferentes características temporais e quantidade de servidores de I/O. Com a maior quantidade de cenários, será possível avaliar estratégias para o início da sincronização de dados e seu impacto nas aplicações em execução.

Também deverão ser investigadas as origens do baixo desempenho obtido em aplicações *I/O Bound* quando executadas concorrentemente com aplicações de comportamento temporal. Os motivos para o melhor desempenho de aplicações *I/O Bound* na estratégia *multi-core*, contrários ao melhor desempenho da execução *single-core* com aplicações temporais, também deve ser avaliado.

Outro trabalho futuro considerado é a integração do monitoramento do sistema com gerenciamento de aplicações dinâmicas, permitindo o crescimento das aplicações apenas se o subsistema de I/O comportar mais acessos aos dados de entrada ou a possível geração de arquivos de saída.

REFERÊNCIAS

AMIRI, K.; GIBSON, G. A.; GOLDING, R. Highly concurrent shared storage. **Distributed Computing Systems, 2000. Proceedings. 20th International Conference on**, [S.l.], p.298–307, 2000.

Apache Foundation. **The Hadoop File System Architecture**. Disponível e `http://hadoop.apache.org/common/docs/current/hdfs_design.html`. Último acesso: Outubro de 2009.

ÁVILA, R. B. **Uma Proposta de Distribuição do Servidor de Arquivos em Clusters**. 2005. Tese de doutorado — Universidade Federal do Rio Grande do Sul, Brasil.

BAKER, M.; BUYYA, R. Cluster computing: the commodity supercomputer. **Software and Practice and Experience**, [S.l.], v.29, n.6, p.551–576, 1999.

BATSAKIS, A.; BURNS, R. Cluster delegation: high-performance, fault-tolerant data sharing in nfs. In: HPDC-14: PROCEEDINGS OF THE 14TH IEEE INTERNATIONAL SYMPOSIUM ON HIGH PERFORMANCE DISTRIBUTED COMPUTING, 2005, Washington, DC, USA. **Anais...** IEEE Computer Society, 2005. p.100–109.

BOLZE, R. et al. Grid'5000: a large scale and highly reconfigurable experimental grid testbed. **International Journal of High Performance Computing Applications**, [S.l.], v.20, n.4, p.481–494, 2006.

BORRILL, J. et al. Investigation of leading HPC I/O performance using a scientific-application derived benchmark. In: ACM/IEEE CONFERENCE ON SUPERCOMPUTING (SC '07), 2007., 2007, New York, NY, USA. **Proceedings...** ACM, 2007. p.1–12.

BRANDT, S. A. et al. Efficient Metadata Management in Large Distributed Storage Systems. **20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies**, [S.l.], p.290–298, 2003.

BUISSON, J. **Adaptation dynamique de programmes et composants parallèles**. 2006. Tese (Doutorado em Ciência da Computação) — INSA de Rennes.

BYNA, S. et al. Parallel I/O prefetching using MPI file caching and I/O signatures. In: SC '08: PROCEEDINGS OF THE 2008 ACM/IEEE CONFERENCE ON SUPERCOMPUTING, 2008, Piscataway, NJ, USA. **Anais...** IEEE Press, 2008. p.1–12.

CARNS, P. H. et al. PVFS: a parallel file system for linux clusters. In: ANNUAL LINUX SHOWCASE AND CONFERENCE (ALS'00), 4., 2000, Berkeley, CA, USA. **Proceedings...** USENIX Association, 2000. p.28–28.

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

CHAPIN, S. J. et al. Benchmarks and standards for the evaluation of parallel job schedulers. **Lecture notes in computer science**, [S.l.], v.1659, p.67–90, 1999.

CHEN, P. M. et al. RAID: high-performance, reliable secondary storage. **ACM Computing Surveys**, [S.l.], v.26, n.2, p.145–185, 1994.

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

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.

DARLING, A. E.; CAREY, L.; FENG, W.-C. The Design, Implementation, and Evaluation of mpiBLAST. In: CLUSTERWORLD 2003, 2003. **Proceedings...** [S.l.: s.n.], 2003.

DEAN, J.; GHEMAWAT, S. MapReduce: simplified data processing on large clusters. In: OSDI'04: PROCEEDINGS OF THE 6TH CONFERENCE ON SYMPOSIUM ON OPEARTING SYSTEMS DESIGN AND IMPLEMENTATION, 2004, Berkeley, CA, USA. **Anais...** USENIX Association, 2004. p.10–10.

DEVULAPALLI, A.; OHIO, P. W. File Creation Strategies in a Distributed Metadata File System. **Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International**, [S.l.], p.1–10, 2007.

DRBD. **Distributed Replicated Block Device Homepage**. http://www.drbd.org/.

FAN, Z.; XIONG, J.; MA, J. A failure recovery mechanism for distributed metadata servers in DCFS2. **High Performance Computing and Grid in Asia Pacific Region, 2004. Proceedings. Seventh International Conference on**, [S.l.], p.2–8, 2004.

FEITELSON, D. **Parallel Workloads Archive**. Último Acesso em Dezembro de 2009.

FRYXELL, B. et al. FLASH: an adaptive mesh hydrodynamics code for modeling astrophysical thermonuclear flashes. **The Astrophysical Journal Supplement Series**, [S.l.], v.131, n.1, p.273–334, 2000.

GARCIA, 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.

GHEMAWAT, S.; GOBIOFF, H.; LEUNG, S.-T. The Google file system. **SIGOPS Oper. Syst. Rev.**, New York, NY, USA, v.37, n.5, p.29–43, 2003.

GIBSON, G.; CORBETT, P. **pNFS Problem Statement**. Disponível em www.ietf.org/internet-drafts/draft-gibson-pnfs-problem-statement-01.txt.

GIBSON, G.; WELSH, W.; CORBETT, P. **Parallel NFS Requirements and Design**. Internet Draft, http://www.ietf.org/internet-drafts/draft-gibson-pnfs-problem-statement-01.txt.

HERMANN, E. **Dinamismo de Servidores de Dados no Sistema de Arquivos dNFSp**. 2006. Dissertação de Mestrado — Universidade Federal do Rio Grande do Sul, Brasil.

HERMANN, E. et al. Utilização de Recursos Alocados pelo Usuário para Armazenamento de Dados no Sistema de Arquivos dNFSp. In: VII WORKSHOP EM SISTEMAS COMPUTACIONAIS DE ALTO DESEMPENHO (WSCAD 2006), 2006, Ouro Preto - MG. **Anais...** [S.l.: s.n.], 2006.

HILDEBRAND, D.; HONEYMAN, P. Scaling NFSv4 with parallel file systems. In: FIFTH IEEE INTERNATIONAL SYMPOSIUM ON CLUSTER COMPUTING AND THE GRID (CCGRID'05), 2005, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2005. v.2, p.1039–1046.

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 (MSST 05), 22., 2005, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2005. p.18.

HUPFELD, F. et al. The XtreemFS architecture—a case for object-based file systems in Grids. **Concurr. Comput. : Pract. Exper.**, Chichester, UK, v.20, n.17, p.2049–2060, 2008.

ISAILA, F. et al. Integrating Logical and Physical File Models in the MPI-IO Implementation for "Clusterfile". In: SIXTH IEEE INTERNATIONAL SYMPOSIUM ON CLUSTER COMPUTING AND THE GRID (CCGRID'06), 2006, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2006. p.462.

ISAILA, F.; TICHY, W. F. Clusterfile: a flexible physical layout parallel file system. In: IEEE INTERNATIONAL CONFERENCE ON CLUSTER COMPUTING (CLUSTER '01), 3., 2001, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2001. p.37.

JAIN, R. et al. Heuristics for scheduling I/O operations. **Parallel and Distributed Systems, IEEE Transactions on**, [S.l.], v.8, n.3, p.310–320, 1997.

JANSSEN, C. et al. **The Massively Parallel Quantum Chemistry Program (MPQC): version 2.3.** `http://www.mpqc.org`.

KASSICK, R.; BOITO, F.; NAVAUX, P. Interaction of Access Patterns on dNFSp File System. In: MEMóRIAS DE CONFERENCIA LATINOAMERICANA DE COMPUTACIóN DE ALTO RENDIMIENTO, 2009. **Anais...** [S.l.: s.n.], 2009. Cópia disponível em `http://eventos.saber.ula.ve/eventos/documentos/clacr2009/pdf_completo.pdf`.

KASSICK, R. et al. Evaluating the Performance of the dNFSP File System. In: IEEE INTERNATIONAL SYMPOSIUM ON CLUSTER COMPUTING AND THE GRID, CCGRID, 5., 2005, Cardiff, UK. **Proceedings...** Los Alamitos: IEEE Computer Society Press, 2005. CD-ROM Proceedings, ISBN 0-7803-9075-X.

KIM et al. **Bigfoot-NFS**: a parallel file-striping nfs server (extended abstract). 1994.

KOLBECK, B. et al. **D3.4.3 – Design Report for the Advanced XtreemFS and OSS Features**. Takustr. 7, 14195 Berlin, Germany: Zuze Institute of Berlin, 2008. Relatório de Projeto, Disponível em `http://www.xtreemos.eu/publications/ploneearticlemultipage.2008-06-26.0232965573/project-deliverables/d3_4_3-final2.pdf`. Último acesso em Outubro de 2009.

KUHN, M.; KUNKEL, J. M.; LUDWIG, T. Dynamic file system semantics to enable metadata optimizations in PVFS. **Concurrency and Computation: Practice and Experience**, [S.l.], 2009.

KULKARNI, K.; GABRIEL, E. Evaluating Algorithms for Shared File Pointer Operations in MPI I/O. In: INTERNATIONAL CONFERENCE ON COMPUTATIONAL SCIENCE (ICCS '09), 9., 2009, Berlin, Heidelberg. **Proceedings...** Springer-Verlag, 2009. p.280–289.

KUNKEL, J. M. **Towards Automatic Load Balancing of a Parallel File System with Subfile Based Migration**. 2007. Dissertação (Mestrado em Ciência da Computação) — Ruprecht-Karls-Universität Heidelberg –Institut für Informatik.

KUNKEL, J. M.; LUDWIG, T. Bottleneck Detection in Parallel File Systems with Trace-Based Performance Monitoring. **Lecture Notes in Computer Science**, [S.l.], v.5168, p.212–221, 2008.

LANG, S. et al. I/O Performance Challenges at Leadership Scale. In: SUPERCOMPUTING 2009, 2009. **Proceedings...** [S.l.: s.n.], 2009. Accepted to Supercomputing 09.

LEBRE, A. et al. I/O Scheduling Service for Multi-Application Clusters. In: IEEE INTERNATIONAL CONFERENCE ON CLUSTER COMPUTING, 2006., 2006. **Proceedings...** [S.l.: s.n.], 2006. p.10.

LISKOV, B. et al. Replication in the harp file system. In: SOSP '91: PROCEEDINGS OF THE THIRTEENTH ACM SYMPOSIUM ON OPERATING SYSTEMS PRINCIPLES, 1991, New York, NY, USA. **Anais...** ACM, 1991. p.226–238.

LIU, W. et al. An Effective File Migration Algorithm in Cluster File Systems. **Parallel Processing Workshops, International Conference on**, Los Alamitos, CA, USA, v.0, p.329, 2000.

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...** IEEE Computer Society, 2002. p.352.

MA, Y.; ZHAO, L.; LIU, D. An Asynchronous Parallelized and Scalable Image Resampling Algorithm with Parallel I/O. In: INTERNATIONAL CONVERENCE ON COMPUTATIONAL SCIENCE (ICCS 2009), 2009. **Proceedings...** Springer, 2009. p.357–366. (LNCS, v.5545).

MADHYASTHA, T. M.; REED, D. A. Intelligent, adaptive file system policy selection. **Frontiers of Massively Parallel Computing, 1996. Proceedings 'Frontiers '96'., Sixth Symposium on the**, [S.l.], p.172–179, Oct 1996.

MARTIN, R. P.; CULLER, D. E. NFS sensitivity to high performance networks. In: ACM SIGMETRICS INTERNATIONAL CONFERENCE ON MEASUREMENT AND MODELING OF COMPUTER SYSTEMS (SIGMETRICS '99), 1999., 1999, New York, NY, USA. **Proceedings...** ACM Press, 1999. p.71–82.

MICROSYSTEMS, S. **Lustre File System**. Available in `http://www.sun.com/software/products/lustre/docs/lustrefilesystem_wp.pdf`, White Paper.

MICROSYSTEMS, S. **Pools of Targets – Lustre File System**. Available in `http://arch.lustre.org/index.php?title=Pools_of_targets`.

NIEUWEJAAR, N. et al. File-Access Characteristics of Parallel Scientific Workloads. **IEEE Trans. Parallel Distrib. Syst.**, Piscataway, NJ, USA, v.7, n.10, p.1075–1089, 1996.

OLDFIELD, R. **Parallel I/O Examples and Benchmark Codes**. http://www.cs.dartmouth.edu/pario/examples.html, Internet Website.

PARALLEL I/O Benchmarking Consortium. http://www-unix.mcs.anl.gov/pio-benchmark/.

PAWLOWSKI, B. et al. **The NFS version 4 protocol**. 2000.

PEREZ, J. M. et al. Data allocation and load balancing for heterogeneous cluster storage systems. **Cluster Computing and the Grid, 2003. Proceedings. CCGrid 2003. 3rd IEEE/ACM International Symposium on**, [S.l.], p.718–723, 2003.

QIN, X. et al. Boosting performance of I/O-intensive workload by preemptive job migrations in a cluster system. In: COMPUTER ARCHITECTURE AND HIGH PERFORMANCE COMPUTING, 2003. PROCEEDINGS. 15TH SYMPOSIUM ON, 2003. **Anais...** [S.l.: s.n.], 2003. p.235–243.

RIBLER, Y. L.; SIMITCI, H.; REED, D. A. The Autopilot performance-directed adaptive control system. **Future Generation Computer Systems**, Amsterdam, The Netherlands, The Netherlands, v.18, n.1, p.175–187, 2001.

ROSELLI, D.; LORCH, J. R.; ANDERSON, T. E. A comparison of file system workloads. In: USENIX ANNUAL TECHNICAL CONFERENCE (ATEC '00), 2000, Berkeley, CA, USA. **Proceedings...** USENIX Association, 2000. p.4–4.

SEELAM, S. et al. Early experiences in application level I/O tracing on blue gene systems. **Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on**, [S.l.], p.1–8, April 2008.

SMIRNI, E.; REED, D. A. Workload Characterization of Input/Output Intensive Parallel Applications. In: INTERNATIONAL CONFERENCE ON COMPUTER PERFORMANCE EVALUATION: MODELLING TECHNIQUES AND TOOLS, 9., 1997, London, UK. **Proceedings...** Springer-Verlag, 1997. p.169–180.

STENDER, J.; KOLBECK, B. **D3.4.5 – XtreemFS and OSS Developer Guide**. Takustr. 7, 14195 Berlin, Germany: Zuze Institute of Berlin, 2009. Manual do Desenvolvedor, Disponível em `http://www.xtreemos.eu/publications/project-deliverables/d3_4_5.pdf`. Último Acesso: Outubro de 2009.

SUN MICROSYSTEMS, I. **RFC 1094**: nfs: network file system protocol specification. 1989.

TRAN, N.; REED, D. A. ARIMA time series modeling and forecasting for adaptive I/O prefetching. In: SUPERCOMPUTING (ICS '01), 15., 2001, New York, NY, USA. **Proceedings...** ACM, 2001. p.473–485.

WANG, F. et al. File system workload analysis for large scale scientific computing applications. In: IEEE / 12TH NASA GODDARD CONFERENCE ON MASS STORAGE SYSTEMS AND TECHNOLOGIES, 21., 2004. **Proceedings...** [S.l.: s.n.], 2004. p.139–152.

WEIL, S. A. **CEPH**: reliable, scalable, and high-performance distributed storage. 2007. Tese (Doutorado em Ciência da Computação) — University of California, Santa Cruz.

WEIL, S. A. et al. Ceph: a scalable, high-performance distributed file system. In: SYMPOSIUM ON OPERATING SYSTEMS DESIGN AND IMPLEMENTATION (OSDI), 7., 2006. **Proceedings...** [S.l.: s.n.], 2006. p.307–320.

WEIL, S. A. et al. CRUSH: controlled, scalable, decentralized placement of replicated data. In: SC '06: PROCEEDINGS OF THE 2006 ACM/IEEE CONFERENCE ON SUPERCOMPUTING, 2006, New York, NY, USA. **Anais...** ACM, 2006. p.122.

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

YU, W. et al. Benefits of high speed interconnects to cluster file systems: a case study with lustre. In: INTERNATIONAL PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM, 2006 (IPDPS ´06), 20., 2006. **Proceedings...** [S.l.: s.n.], 2006. p.8.

ZHOU, X.; WEI, T. A greedy I/O scheduling method in the storage system of clusters. **Cluster Computing and the Grid, 2003. Proceedings. CCGrid 2003. 3rd IEEE/ACM International Symposium on**, [S.l.], p.712–717, 2003.

ZHU, Y. et al. 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.