PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

AVALIANDO O SISTEMA DE ARQUIVOS LUSTRE COM USO DE CARGAS DE TRABALHO DE APLICAÇÕES PARALELAS

EVANDRO MIGUEL KUSZERA

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Ciência da Computação na Pontifícia Universidade Católica do Rio Grande do Sul.

Orientador: Prof. Dr. César Augusto Fonticielha De Rose

Porto Alegre
2010

Dados Internacionais de Catalogação na Publicação (CIP)

K97a Kuszera, Evandro Miguel
Avaliando o sistema de arquivos Lustre com uso de cargas de trabalho de aplicações paralelas / Evandro Miguel Kuszera. – Porto Alegre, 2010.
90 f.

Diss. (Mestrado) – Fac. de Informática, PUCRS.
Orientador: Prof. Dr. César Augusto Fonticielha De Rose.

1. Informática. 2. Sistemas Distribuídos.
3. Banco de Dados Distribuídos. 4. Programação Paralela.
I. De Rose, César Augusto Fonticielha. II. Título.

CDD 004.36

Ficha Catalográfica elaborada pelo
Setor de Tratamento da Informação da BC-PUCRS

Pontifícia Universidade Católica do Rio Grande do Sul
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada "Avaliando o Sistema de Arquivos Lustre com uso de Cargas de Trabalhos de Aplicações Paralelas", apresentada por Evandro Miguel Kuszera, como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, Processamento Paralelo e Distribuído, aprovada em 25/03/10 pela Comissão Examinadora:

Prof. Dr. César Augusto Fonticielha De Rose – Orientador — PPGCC/PUCRS

Prof. Dr. Luiz Gustavo Leão Fernandes – — PPGCC/PUCRS

Prof. Dr. Nicolas Maillard – — UFRGS

Homologada em 01/06/2010, conforme Ata No. 09/2010 pela Comissão Coordenadora.

Prof. Dr. Fernando Gehm Moraes
Coordenador.

PUCRS
Campus Central
Av. Ipiranga, 6681 – P32– sala 507 – CEP: 90619-900
Fone: (51) 3320-3611 – Fax (51) 3320-3621
E-mail: ppgcc@pucrs.br
www.pucrs.br/facin/pos

DEDICATÓRIA

Aos meus pais, José e Helena Kuszera, e
amada esposa Sheila Redivo Kuszera.

AGRADECIMENTOS

Ao meu orientador professor Dr. César Augusto Fonticielha De Rose pela orientação, apoio e paciência durante toda a realização deste trabalho.

Aos meus pais, José e Helena Kuszera, pelo amor, educação, conselhos e apoio em todos os passos da minha vida.

A minha amada esposa Sheila Kuszera, pelo apoio, compreensão, incentivo, paciência pelas horas de ausência e, acima de tudo, por ser essa pessoa dedicada e carinhosa, sempre com palavras de força e afeto, que tanto ajudaram durante os momentos mais difíceis desta caminhada.

Aos colegas de trabalho do IFSUL - Campus Passo Fundo pelo apoio e ajuda no cumprimento das atividades profissionais durante esses dois anos. E por fim, aos colegas, professores e funcionários do mestrado em ciência da computação da PUCRS.

AVALIANDO O SISTEMA DE ARQUIVOS LUSTRE COM USO DE CARGAS DE TRABALHO DE APLICAÇÕES PARALELAS

RESUMO

O crescente avanço na capacidade de processamento dos computadores fornece meios para projetar e executar aplicações com demandas cada vez maiores. Contudo, dependendo da aplicação, há também a necessidade de acessar e armazenar grandes porções de dados de forma eficiente. Aplicações voltadas à ciência, engenharia, mineração de dados e simulações de eventos naturais são alguns exemplos de aplicações que requerem alta vazão de dados. *Clusters* Linux e sistemas de arquivos distribuídos, geralmente são utilizados nestes cenários. Entretanto, sistemas de arquivos distribuídos ditos tradicionais, como NFS, não são adequados para aplicações intensivas em dados. A arquitetura centralizada limita o desempenho e escala da aplicação.

Com base nisso, vários sistemas de arquivos paralelos foram concebidos com o objetivo de amenizar o gargalo criado no acesso aos dados. Dentre esses sistemas, destaca-se o Lustre, sistema de arquivos paralelos amplamente utilizado pela comunidade de alto desempenho. Neste trabalho, realiza-se uma avaliação do Lustre sobre um *cluster* Linux de pequena escala. A avaliação tem por objetivo identificar quais fatores afetam o desempenho do sistema de arquivos, e como o mesmo se comporta sob cargas de trabalho típicas de aplicações paralelas.

Os resultados obtidos mostraram que o Lustre é um sistema de arquivos adequado para todas as classes de aplicações avaliadas. Entretanto, para se obter bom desempenho é importante tornar os acessos, realizados pelos processos, contíguos dentro do arquivo. Dessa forma, é possível aproveitar os recursos fornecidos pelo Lustre, como *cache* cliente e *read-ahead*.

Palavras-chave: Sistema de Arquivos Distribuído, Sistema de Arquivos Paralelos; Armazenamento.

EVALUATING THE LUSTRE FILE SYSTEM WITH USE OF PARALLEL APPLICATIONS WORKLOADS

ABSTRACT

The growing advance in computers processing power provides the means to design and run high performance applications. But depending on the application, there is also the need to efficiently store and access large amounts of data. Applications focused on science, engineering, data mining and simulation of natural events are some examples that require high I/O throughput. Linux clusters and distributed file systems are generally used in these scenarios. However, traditional distributed file systems, like NFS, are not suitable for data-intensive applications. The centralized architecture of such systems limits the performance and scalability of the application.

Based on this, several parallel file systems were designed with the purpose of alleviating the bottleneck created in data access. Among these systems is Lustre, a parallel file system widely used by the high performance community. In this work, an evaluation of Lustre on a small scale Linux cluster is carried out. The evaluation aims to identify which factors affect the performance of the parallel file system, and how it behaves under typical parallel applications workloads.

The results showed that Lustre is a file system suitable for all the evaluated application classes. However, to achieve good performance processes should try to maximize contiguous access to files. In that way, is possible to take advantage of the optimizations provided by Lustre, like the client cache and read-ahead mechanism.

Keywords: Distributed File System; Parallel File System, Storage.

LISTA DE FIGURAS

Figura 2.1 Arquitetura genérica de *clusters*. 28
Figura 2.2 Componentes da arquitetura do NFS. 33
Figura 2.3 Arquivo paralelo. 34
Figura 2.4 Arquitetura de sistema de arquivos paralelos. 35
Figura 2.5 Arquitetura do GlusterFS [18]. 37
Figura 2.6 Arquitetura do PVFS. 38
Figura 2.7 Arquitetura do pNFS. 39
Figura 2.8 Padrão de acesso desempenhado pelo *benchmark* IOR. 42

Figura 3.1 Arquitetura do Lustre. 44
Figura 3.2 Representação de um *inode* no Lustre. 45
Figura 3.3 Dois arquivos distribuídos com *stripecount* 2 e 3, e *stripesizes* distintos. . . . 46
Figura 3.4 Sistema de arquivos global usando LNET. 47
Figura 3.5 Representação gráfica do ambiente de teste. 51

Figura 4.1 Modelo de entrada/saída de aplicações paralelas. 54
Figura 4.2 Padrão de acesso *file-per-proc*. 55
Figura 4.3 Padrão de acesso *segment access*. 56
Figura 4.4 Padrão de acesso *simple strided*. 56
Figura 4.5 Padrão de acesso *random access*. 57
Figura 4.6 Vazão média e desvio padrão na escrita ao variar o tamanho do arquivo. . . . 61
Figura 4.7 Uso do *cache* cliente. 62
Figura 4.8 Vazão na leitura ao variar o tamanho do arquivo. 62
Figura 4.9 Efeito do *read-ahead*. 63
Figura 4.10 Variando o tamanho da unidade de transferência. 64
Figura 4.11 Variando o número de processos. 65
Figura 4.12 Escrita variando o número de servidores de armazenamento. 66
Figura 4.13 Leitura variando o número de servidores de armazenamento. 67
Figura 4.14 Formas de distribuição de arquivos adotadas no experimento. 68
Figura 4.15 Vazão alcançada com *P=3* e *P=6*. 70
Figura 4.16 Vazão alcançada com *P=4* e *P=8*. 70

Figura 4.17 Acesso não alinhado ao tamanho do *stripe*. 71
Figura 4.18 Acesso alinhado e não alinhado ao tamanho do *stripe*. 73
Figura 4.19 Padrão de acesso *file-per-proc*. 75
Figura 4.20 Padrão de acesso *segmented-access*. 76
Figura 4.21 Acesso a regiões não contíguas do arquivo (*simple-strided*). 78
Figura 4.22 Vazão na escrita com padrão de acesso *random-strided*. 79
Figura 4.23 Vazão na leitura com padrão de acesso *random-strided*. 80

LISTA DE TABELAS

Tabela 3.1 Configuração das máquinas. 50
Tabela 3.2 Relação de *softwares* do cenário de estudo. 51

Tabela 4.1 Descrição das configurações usadas no experimento. 60
Tabela 4.2 Limites empíricos dos componentes do ambiente de teste. 65
Tabela 4.3 Descrição das configurações usadas no experimento. 69
Tabela 4.4 Lista dos testes realizados no experimento. 69
Tabela 4.5 Descrição das configurações usadas no experimento. 71
Tabela 4.6 Lista dos testes realizados no experimento. 72
Tabela 4.7 Descrição das configurações usadas no experimento. 75
Tabela 4.8 Descrição das configurações usadas no experimento. 76
Tabela 4.9 Descrição das configurações usadas no experimento. 77
Tabela 4.10 Descrição das configurações usadas no experimento. 79
Tabela 4.11 Resumo das melhores taxas de transferência (MB/s) obtidas com o Lustre. . . 81

LISTA DE SIGLAS

ACL	Access Control List
AFS	Andrew File System
API	Application Programming Interface
E/S	Entrada/Saída
FUSE	Filesystem in Userspace
GPL	General Public License
IETF	Internet Engineering Task Force
IOR	Interleaved or Random
LNET	Lustre Networking
MDS	Metadata Server
MDT	Metadata Target
MPI	Message Passing Interface
MPP	Massive Parallel Processing
NFS	Network File System
NPB	NAS Parallel Benchmarks
OBD	Object-Based Disks
OSC	Object Storage Client
OSS	Object Storage Servers
OST	Object Storage Target
PC	Personal Computer
pNFS	Parallel NFS
POSIX	Portable Operating System Interface
PVFS	Parallel Virtual File System
RAID	Redundant Arrays of Inexpensive Disks
RDMA	Remote Direct Memory Access
RPC	Remote Procedure Call
SAN	Storage Area Network
SMP	Symmetric Multiprocessing
SNMP	Simple Network Management Protocol
SPFS	Scotch Parallel File System

TCP	Transmission Control Protocol
TI	Tecnologia da Informação
VFS	Virtual File System
XDR	eXternal Data Representation

SUMÁRIO

1 INTRODUÇÃO 23

1.1 Motivação 24
1.2 Objetivos 24
1.3 Organização do trabalho 25

2 FUNDAMENTAÇÃO TEÓRICA 27

2.1 Computação de Alto Desempenho 27
2.1.1 *Cluster* de Computadores 27
2.1.2 Interface de Passagem de Mensagens 28
2.2 Armazenamento de Dados 29
2.3 Sistemas de Arquivos Distribuídos 30
2.3.1 Características desejáveis para sistema de arquivos distribuído 30
2.3.2 Arquitetura do NFS 32
2.4 Sistemas de Arquivos Paralelos 33
2.4.1 Distribuição dos Dados e Primitivas de Acesso 33
2.4.2 Arquitetura de Sistemas de Arquivos Paralelos 34
2.4.3 Alguns Sistemas de Arquivos Paralelos 35
2.5 Uso de Benchmarks para Avaliar Sistemas de Arquivos 40
2.5.1 Benchmark IOR 41

3 SISTEMA DE ARQUIVOS LUSTRE 43

3.1 Arquitetura 43
3.1.1 Armazenamento de Arquivos 44
3.1.2 *Striping* de Arquivos 45
3.1.3 Rede 46
3.2 Características Gerais 47
3.2.1 Interfaces de Acesso 47
3.2.2 Limites do Lustre 48
3.2.3 Administração do Lustre 48

3.3 Cenário de Estudo 50
3.3.1 Ambiente de Teste 50
3.3.2 Configuração do Lustre 51
3.3.3 Configuração do NFS 52

4 AVALIAÇÃO E RESULTADOS OBTIDOS 53
4.1 Carga de Trabalho 53
4.1.1 Entrada e Saída 53
4.1.2 Padrões de Acesso 55
4.2 Metodologia de Avaliação 57
4.2.1 Métricas de Avaliação 57
4.2.2 Projeto de Experimentos 58
4.3 Avaliação com Foco no Ambiente de Teste 59
4.3.1 Tamanho do Arquivo 60
4.3.2 Tamanho da Unidade de Transferência 63
4.3.3 Número de Clientes 64
4.3.4 Número de Servidores de Armazenamento 66
4.3.5 Distribuição de Carga entre Servidores de Armazenamento 67
4.3.6 Acesso Não Alinhado ao Tamanho do *Stripe* 70
4.3.7 Resumo dos Fatores de Impacto 73
4.4 Avaliação com Foco nos Padrões de Acesso 74
4.4.1 *File-per-proc* 74
4.4.2 *Segmented-access* 75
4.4.3 *Simple-strided* 77
4.4.4 *Random-strided* 78
4.4.5 Comparando os padrões de acesso 80

5 CONCLUSÃO 83

BIBLIOGRAFIA 87

1. INTRODUÇÃO

O crescente avanço na capacidade de processamento dos computadores fornece meios para projetar e executar aplicações com demandas cada vez maiores. Na área científica há muitas aplicações que realizam computação intensiva sobre grandes massas de dados, relacionadas à modelagem climática, dinâmica de fluídos, física de altas energias, biologia, entre outras. Outros exemplos de aplicações são as relacionadas à engenharia para modelar fenômenos físicos, essenciais para indústrias, incluindo indústria petrolífera, automotiva, aeronáutica, farmacêutica, entre outras. Além dessas, aplicações comerciais também necessitam de grande vazão de dados, como por exemplo, aplicações de mineração de dados e grandes servidores de e-mail.

O desafio dos cientistas e engenheiros é produzir mecanismos eficientes para processar, armazenar e recuperar extensa quantidade de dados. Arquiteturas paralelas tipicamente são utilizadas para a execução dessas aplicações, sendo os *clusters* de computadores as máquinas mais utilizadas. Entretanto, a evolução dos dispositivos de armazenamento, neste caso, discos magnéticos, não acompanhou a velocidade que a tecnologia de construção de processadores e memórias evoluiu. Atualmente há processadores operando na casa dos sub-nanossegundos e memórias com taxas de transferência de 10 GB/s, enquanto que o tempo de acesso e vazão de dados de discos rígidos atuais ainda são da ordem de milissegundos e 300 MB/s, respectivamente. Para amenizar essas diferenças de velocidades são necessários mecanismos adequados para diminuir o gargalo no acesso aos dados.

Sistemas de arquivos distribuídos como NFS [19], [33] e AFS [26], [15], baseados no modelo cliente-servidor são largamente utilizados em *clusters* Linux. Em *clusters* de baixa escala, com poucos nós, NFS e AFS mostram-se eficientes, provendo um sistema de arquivos de rede compartilhado e com espaço de nomes único. Entretanto, a arquitetura baseada no modelo cliente-servidor, sob um grande número de clientes executando operações de entrada e saída (E/S) em paralelo, cria um gargalo no acesso aos dados, que limita a escala e desempenho do sistema.

Em cenários com dezenas de milhares de nós são necessários sistemas de arquivos que forneçam alta vazão de dados. Surgiram então os sistemas de arquivos paralelos. Nesta abordagem, alguns nós do *cluster* são responsáveis por armazenar os dados. Os arquivos são fracionados e distribuídos entre os nós de armazenamento. Dessa forma, é possível acessar em paralelo diferentes partes de um mesmo arquivo, aumentando a vazão de dados do sistema. De forma sucinta, os sistemas de arquivos paralelos devem prover espaço de nomes único, boa escalabilidade e capacidade de distribuir grandes arquivos através de múltiplos nós.

Neste trabalho será realizada uma investigação do Lustre, um dos sistemas de arquivos paralelos com grande visibilidade e aceitação na comunidade de alto desempenho. A seguir, as motivações que levaram a elaboração deste estudo e objetivos que se deseja atingir.

1.1 Motivação

Sistemas de arquivos distribuídos ditos tradicionais, como NFS, não são capazes de fornecer os níveis de escalabilidade e *throughput* adequados. O gargalo no acesso aos dados ocasionado pela arquitetura do sistema de arquivos limita o desempenho de aplicações paralelas intensivas em dados. Para amenizar esses problemas surgiram os sistemas de arquivos paralelos. Entretanto, a arquitetura mais complexa e grau de dificuldade encontrado na instalação, configuração e gerenciamento desses sistemas, muitas vezes tornam árduo o uso em ambientes de produção. Por isso a importância em investigar tais sistemas, visando fornecer maiores informações de como utilizá-los de forma eficiente.

Dentre outros sistemas de arquivos paralelos, o que motiva o estudo do Lustre, se deve principalmente pela sua grande utilização entre os pesquisadores e centros de alto desempenho. Entre os dez primeiros supercomputadores da lista TOP500 [24], seis utilizam o Lustre como sistema de arquivos.

Vários trabalhos avaliaram o Lustre sobre *clusters* de grande escala [47], [46], [1], [37], entretanto, o presente trabalho está voltado para *clusters* de pequena escala, que de certa forma representam a realidade de muitos centros de pesquisas nacionais. Além disso, ainda faltam muitas informações para os profissionais de TI determinarem qual sistema de arquivos é mais adequado para a sua classe de aplicações.

1.2 Objetivos

Esta dissertação tem como objetivo principal investigar e avaliar o Lustre sob cargas de trabalho de aplicações paralelas. Para tanto, os seguintes objetivos específicos foram traçados:

- Realizar um estudo dos sistemas de arquivos distribuídos e paralelos, visando compreender melhor a arquitetura e funcionalidades fornecidas por esses sistemas. Em um segundo momento será realizado um estudo aprofundado do Lustre, discutindo detalhes da arquitetura, forma de armazenamento de arquivos, funcionalidades, características gerais e limitações.

- Avaliar o Lustre através de *benchmarks* e pequenas aplicações paralelas a fim de verificar quais fatores tem influência no desempenho.

- Avaliar o Lustre sob alguns padrões de acesso típicos de aplicações paralelas reais. O objetivo é verificar se o Lustre é um sistema de arquivos adequado para os padrões de acesso utilizados neste trabalho.

- Por último, as avaliações realizadas neste trabalho visam fornecer maiores informações aos profissionais de TI que pretendem usar o Lustre. Essas informações poderão ser utilizadas para auxiliar na escolha do sistema de arquivos mais adequado para a classe de aplicações alvo.

1.3 Organização do trabalho

O restante deste trabalho está organizado da seguinte forma: no próximo Capítulo será apresentado o embasamento teórico relativo à computação de alto desempenho e armazenamento de dados. Serão abordados conceitos sobre sistemas de arquivos centralizados, distribuídos e paralelos. No Capítulo 3 o Lustre é discutido em maiores detalhes. Além disso, a infraestrutura do ambiente de testes e configurações do Lustre é apresentada. A elaboração dos testes e resultados obtidos são apresentados no Capítulo 4. Por fim, no Capítulo 5 as conclusões.

2. FUNDAMENTAÇÃO TEÓRICA

Este Capítulo tem como objetivo servir de embasamento teórico para o restante do trabalho. O mesmo está dividido em cinco seções, onde a primeira discute alguns aspectos relacionados à computação de alto desempenho. A seção 2.2 define conceitos básicos relativos ao armazenamento de arquivos. A seção 2.3 introduz conceitos de sistemas de arquivos distribuídos, características desejáveis e apresenta a arquitetura do NFS. A seção 2.4 conceitua sistemas de arquivos paralelos, comenta os tipos de arquiteturas utilizadas, forma de distribuição de dados e primitivas de acesso. Além disso, apresenta alguns sistemas de arquivos paralelos existentes. Por fim, na seção 2.5 serão apresentados conceitos sobre *benchmarks*, além de apresentar alguns *benchmarks* voltados para avaliar sistemas de armazenamento de dados.

2.1 Computação de Alto Desempenho

Aplicações de diversas áreas do conhecimento, incluindo física, química, biologia, astronomia e engenharias, entre outras, possuem fortes requisitos computacionais. Essas aplicações necessitam de máquinas paralelas capazes de resolver problemas de grande escala ou simular fenômenos naturais que não podem ser resolvidos em tempo hábil por um único computador.

Geralmente, supercomputadores eram utilizados para executar essa classe de aplicações. Entretanto, na década de 90, o conceito de processamento paralelo começou a mudar este cenário. Em lugar dos supercomputadores (máquinas vetoriais e MPPs), com *hardware* proprietário e de alto custo, surgiram os *clusters* de computadores, que utilizam computadores tradicionais (PCs, *Workstations*, SMPs) interconectados em rede.

Os *clusters* são computadores baseados em *hardware* de prateleira, de baixo custo. Contudo, a grande evolução da tecnologia de construção de processadores e memórias tornou essas máquinas uma alternativa interessante, que já se popularizaram como plataformas para computação de alto desempenho. Maiores detalhes sobre esse tipo de máquina paralela serão apresentados a seguir.

2.1.1 *Cluster* de Computadores

De acordo com Buyya[3], *cluster* é um tipo de sistema de processamento distribuído ou paralelo, que consiste de uma coleção de computadores interconectados, trabalhando de forma cooperativa como um único recurso de computação integrado. Um *cluster* geralmente se refere a dois ou mais computadores conectados através de uma rede. A Figura 2.1 ilustra uma arquitetura genérica de *cluster*.

Como mostra a Figura 2.1, um *cluster* é composto de vários computadores ou nós, compostos por um ou mais processadores, memória principal, sistema de E/S e sistema operacional, interconectados por uma rede de comunicação de baixa latência. O *Middleware* é uma camada de *software*

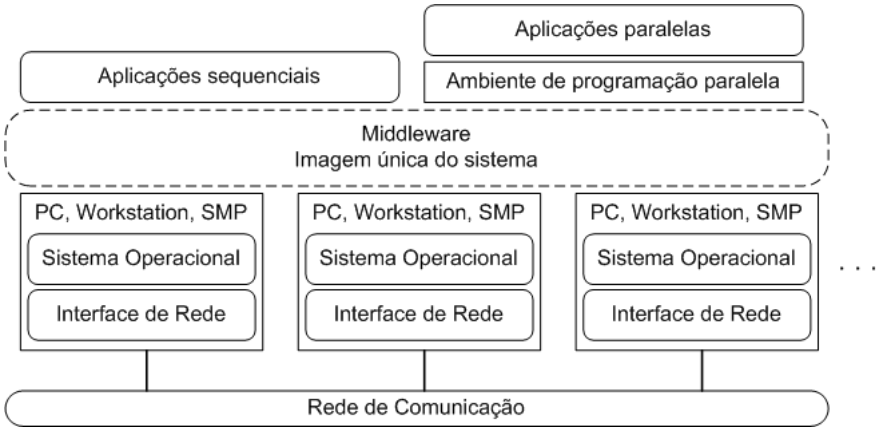


Figura 2.1: Arquitetura genérica de *clusters*.

responsável por fornecer uma imagem única do sistema para usuários e aplicações. Os programas paralelos se comunicam através de bibliotecas de passagem de mensagens (geralmente MPI), fornecidas pelo ambiente de programação disponível. Mesmo sendo uma máquina paralela, é importante ressaltar que *clusters* podem ser utilizados para executar aplicações sequenciais ou paralelas.

2.1.2 Interface de Passagem de Mensagens

Em máquinas paralelas os processos cooperam através de troca de mensagens. As mensagens podem ser dados necessários à computação ou pontos de sincronismos. MPI (*Message Passing Interface*) [8] é a principal interface de troca de mensagens. Foi padronizado por um grupo de pesquisadores com intuito de se tornar um padrão para programas que utilizam troca de mensagem para se comunicar. A principal vantagem oferecida é a portabilidade sob máquinas distintas, ou seja, programas escritos com MPI são capazes de executar sem modificações em plataformas distintas desde que esteja disponível uma versão da biblioteca. Outro aspecto é a capacidade de executar de modo transparente em máquinas heterogêneas, com processadores de arquiteturas diversas.

MPI oferece um conjunto de rotinas para envio e recebimento de mensagens de dados e sincronismo. As tarefas MPI podem executar em um mesmo processador ou concorrentemente sobre diversos processadores.

Na segunda versão da biblioteca foi introduzido o suporte a E/S paralela, também conhecida como MPI-IO [12]. Essa extensão aproveita a analogia do envio e recebimento de mensagens para escrever e ler dados, respectivamente. O propósito da MPI-IO é melhorar o desempenho sobre a API Unix de acesso ao sistema de arquivos. A MPI-IO fornece suporte a acessos não contíguos em memória e arquivo, operações de E/S coletivas, uso explícito de *offsets* para evitar *seeks* individuais, suporte a ponteiros de arquivos exclusivos e compartilhados, E/S não bloqueante, possibilidade de criação de representações de dados portáveis e, por fim, suporte a *hints* (meios de fornecer informações sobre a aplicação, por exemplo, a forma como a aplicação irá interagir com o sistema de arquivos).

2.2 Armazenamento de Dados

Arquivos são abstrações utilizadas para armazenar dados sobre dispositivos que tipicamente utilizam meio magnético para manter as informações. Além disso, os arquivos têm um nome associado, frequentemente relacionado ao tipo de conteúdo armazenado. Os dois principais objetivos que levam ao uso de arquivos consistem em armazenar dados de forma persistente e permitir posterior compartilhamento entre usuários [39].Os arquivos são compostos por dados e atributos. Os dados consistem em uma sequência de *bytes*, acessíveis através de operações de leitura e escrita. Os atributos mantêm informações como tamanho do arquivo, indicações de tempo, tipo de arquivo, proprietário e listas de controle de acesso. Tipicamente são organizados como um registro [7].

O sistema de arquivos é um subsistema do sistema operacional, que tem por objetivo fornecer serviços relacionados ao gerenciamento de arquivos. É responsável pela organização, armazenamento, recuperação, atribuição de nomes, compartilhamento e proteção de arquivos. Grande parte dos sistemas de arquivos utiliza o conceito de diretórios para organizar os arquivos. Um diretório consiste em um arquivo especial, responsável por armazenar informações sobre os arquivos que contém, por exemplo, o mapeamento dos nomes textuais de arquivos para referências internas de arquivos [7]. Além disso, questões relacionadas ao controle de acesso sobre cada arquivo, como permissões dos usuários e o tipo de acesso requerido (leitura, escrita e execução), são de responsabilidade do sistema de arquivos.

Todas as informações adicionais utilizadas para o gerenciamento dos arquivos, como atributos, diretórios e demais informações persistentes utilizadas pelo sistema de arquivos são referenciadas pelo termo metadados [7].

O acesso aos arquivos é realizado através de um conjunto de primitivas fornecidas pelo sistema de arquivos. Geralmente são implementadas como chamadas de sistema, localizadas no *kernel* do sistema operacional. A seguir as principais operações que podem ser executadas sobre arquivos [7]:

- **Abrir:** abre um arquivo existente de acordo com o nome fornecido por parâmetro.
- **Criar:** cria um novo arquivo de acordo com o nome fornecido por parâmetro.
- **Fechar:** fecha o arquivo aberto.
- **Ler:** transfere *n* blocos de dados para local especificado na memória principal.
- **Escrever:** transfere *n* blocos de dados para o arquivo em disco.
- **Reposicionar:** desloca o ponteiro de leitura e escrita para determina região do arquivo (operação conhecida como *seek*).
- **Apagar:** remove o arquivo da estrutura de diretórios, de acordo com o nome fornecido por parâmetro.

- **Truncar:** apaga somente o conteúdo do arquivo, porém não altera os dados dos atributos.

Considerando os conceitos básicos apresentados nesta seção sobre sistemas de arquivos centralizados, a seguir serão apresentados os sistemas de arquivos distribuídos, que têm por função prover serviços de arquivo compartilhado pela rede comunicações.

2.3 Sistemas de Arquivos Distribuídos

Um sistema de arquivos distribuído tem por função compartilhar serviços de arquivo através de uma rede de comunicação, permitindo acesso a arquivos remotos de forma transparente. Para o usuário, o acesso a um arquivo remoto é realizado da mesma forma como são realizados os acessos em um sistema de arquivos centralizado. Sistemas de arquivos distribuídos tipicamente fornecem três tipos de serviços [39]:

- **Serviço de armazenamento:** trata do gerenciamento e alocação de espaço sobre dispositivos de armazenamento. Fornece uma camada de abstração para armazenar e recuperar dados do dispositivo.

- **Serviço de arquivos:** trata de operações relacionadas a arquivos individuais, como criação, alteração e remoção. Para executar essas primitivas são necessários mecanismos de acesso a arquivos, compartilhamento, *cache*, replicação, controle de concorrência, consistência de dados e controle de acesso.

- **Serviço de nomes:** fornece o mapeamento entre nomes de arquivos e identificadores de arquivos. Esse mecanismo facilita a localização do arquivo por parte do usuário.

A separação entre serviço de armazenamento e serviço de arquivos permite a utilização de vários tipos de dispositivos (discos, fitas, etc.) para armazenar arquivos.

2.3.1 Características desejáveis para sistema de arquivos distribuído

A implementação de um sistema de arquivos distribuído deve levar em consideração as seguintes características apresentadas a seguir [7]:

- **Transparência:** as seguintes formas de transparências são parcialmente, ou totalmente, tratadas pelos serviços de arquivos atuais:

  - Transparência do acesso: os programas clientes não devem conhecer a distribuição de arquivos. Um único conjunto de operações é fornecido para acesso a arquivos locais e remotos. Os programas escritos para operar sobre arquivos locais também estão aptos a operar arquivos remotos sem modificações.

  - Transparência de localização: os programas clientes devem ver um espaço de nomes de arquivos uniforme. Os arquivos, ou grupos de arquivos, podem ser deslocados de um servidor a outro sem alteração de seus nomes de caminho, e os programas de usuário devem ver o mesmo espaço de nomes onde quer que sejam executados.
  - Transparência de mobilidade: nem os programas clientes, nem as tabelas de administração de sistema nos nós clientes precisam ser alterados quando os arquivos são movidos. Isso permite a mobilidade do arquivo.
  - Transparência de desempenho: os programas clientes devem continuar funcionando satisfatoriamente, enquanto a carga sobre o serviço varia dentro de um intervalo especificado.
  - Transparência de mudança de escala: o serviço pode ser expandido de forma a tratar com uma ampla variedade de cargas e tamanhos de rede.

- **Atualizações concorrentes de arquivos:** as alterações feitas em um arquivo por um único cliente não devem interferir na operação de outros clientes que estejam acessando, ou alterando, o mesmo arquivo simultaneamente. A maior parte dos serviços de arquivo atuais segue os padrões Unix modernos, fornecendo travamentos (*locking*) em nível de arquivo ou em nível de registro.

- **Replicação de arquivos:** em um serviço de arquivos que suporta replicação, um arquivo pode ser representado por várias cópias de seu conteúdo em diferentes locais. Isso tem duas vantagens, permitir que vários servidores compartilhem a carga do fornecimento de um serviço para clientes que acessam o mesmo conjunto de arquivos, melhorando a escalabilidade do serviço, e melhora a tolerância a falhas, permitindo que, em casos de falhas, os clientes localizem outro servidor que contenha uma cópia do arquivo. Poucos serviços de arquivos suportam replicação completa, mas a maioria suporta o armazenamento de arquivos, ou de porções de arquivos, em *caches* locais, que é uma forma limitada de replicação.

- **Heterogeneidade do *hardware* e do sistema operacional:** as interfaces de serviço devem ser definidas de modo que o *software* cliente e servidor possam ser implementados para diferentes sistemas operacionais e computadores.

- **Tolerância a falhas:** por ser parte essencial nos sistemas distribuídos, é essencial que o serviço de arquivos distribuído continue a funcionar diante de falhas de clientes e servidores. Felizmente, um projeto moderadamente tolerante a falhas é fácil para servidores simples. Os servidores podem ser sem estado (*stateless*), para que após uma falha o serviço seja reiniciado e restaurado sem necessidade de recuperar o estado anterior.

- **Segurança:** nos sistemas de arquivos distribuídos, há necessidade de autenticar as requisições dos clientes para que o controle de acesso no servidor seja baseado nas identidades corretas de

usuários e para proteger o conteúdo das mensagens de requisição e resposta com assinaturas digitais e criptografia de dados secretos.

- **Eficiência:** um serviço de arquivo distribuído deve oferecer recursos que tenham pelo menos o mesmo poder e generalidade daqueles encontrados nos sistemas de arquivos convencionais, e deve obter um nível de desempenho comparável.

Dentre os sistemas de arquivos distribuídos, o NFS da Sun Microsystem é um dos mais amplamente utilizados para sistemas baseados em *clusters* Linux. A arquitetura do NFS define a maioria dos serviços que um sistema de arquivos distribuído deve prover. A seguir, a arquitetura do NFS será apresentada.

2.3.2 Arquitetura do NFS

O papel do NFS [19] é fornecer acesso remoto e transparente a arquivos compartilhados em rede. Foi projetado para ser independente de plataforma, arquitetura de rede e protocolos de comunicação. Isto se deve ao uso de mecanismos como RPC (*Remote Procedure Call*) e XDR (*eXternal Data Representation*). O protocolo RPC permite a execução de chamadas remotas de forma transparente, da mesma forma como chamadas locais são efetuadas. A XDR define a representação de dados utilizada na troca de mensagens entre diferentes arquiteturas e sistemas operacionais.

Um cliente acessa o sistema de arquivos usando as chamadas de sistema fornecidas pelo sistema operacional local (Figura 2.2). Essas chamadas são repassadas para a camada de sistema de arquivo virtual (*Virtual File System* - VFS), que tem por função mascarar o sistema de arquivos adjacente. Dessa forma é possível alterar o tipo do sistema de arquivos utilizado sem ter que reimplementar várias porções de código do sistema operacional. A ideia do VFS é ocultar as diferenças entre vários sistemas de arquivos e fornecer uma interface de acesso padrão. Com NFS, as operações na interface VFS são passadas para um sistema de arquivos local ou para um componente separado conhecido como cliente NFS, que se encarrega de manipular o acesso a arquivos localizado em um servidor remoto [40].

No lado do servidor podemos ver uma organização semelhante. As requisições chegam ao apêndice RPC que desmonta a mensagem e envia ao servidor NFS. O servidor NFS é responsável por converter as mensagens em operações comuns de arquivos VFS que, na seqüência, são passadas para a camada VFS. Mas uma vez, VFS transforma as operações em chamadas compreensíveis ao sistema de arquivo local [40].

A grande vantagem do uso do VFS é o fato de tornar o NFS independente do sistema local de arquivos. Isso permite exportar sistemas de arquivos que estão sob sistemas operacionais baseados em Windows e Unix, por exemplo.

Outro ponto importante do NFS é o fato do protocolo não manter informações de estado. O servidor não retém informações a respeito dos acessos realizados pelos clientes. Na ocorrência de

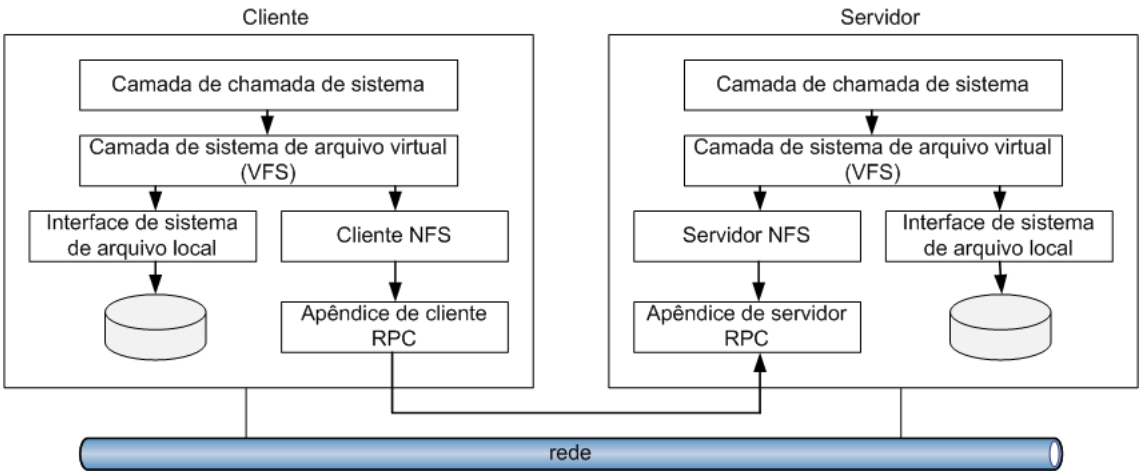


Figura 2.2: Componentes da arquitetura do NFS.

falhas, a tarefa do cliente resume-se em reencaminhar as requisições ao servidor até que estas sejam respondidas. Entretanto, em servidores com informações de estado, os clientes necessitam de mecanismos para detectar e restaurar as informações ao ponto anterior a falha, caso contrário o cliente pode tornar-se falho. Esses mecanismos adicionam maior complexidade a implementação do protocolo.

2.4 Sistemas de Arquivos Paralelos

O contínuo avanço na tecnologia de desenvolvimento de processadores permite a implementação de *clusters* com milhares de núcleos de processamento. Na medida em que o número de processadores aumenta, a demanda por E/S também cresce. Nesses cenários, o modelo cliente-servidor do NFS não alcança a escalabilidade necessária. Em *clusters* com maiores dimensões, o acesso concorrente ao servidor de arquivos gera um gargalo no acesso aos dados. Além disso, a arquitetura centralizada torna o servidor o ponto central de falhas.

Os sistemas de arquivos paralelos foram projetados para fornecer maior vazão de dados para aplicações paralelas. Da mesma forma que são utilizados vários processadores em conjunto para se obter maior poder computacional, os sistemas de arquivos paralelos utilizam vários discos, de forma a agregar a banda individual de cada dispositivo, fornecendo alta vazão de dados. O princípio básico desses sistemas é fracionar o arquivo e distribuir os blocos de dados resultantes entre nós de armazenamento conectados através de uma rede. Esse fracionamento possibilita o acesso em paralelo a diferentes partes do arquivo pelos processos de uma aplicação paralela. Além do fator desempenho, sistemas de arquivos paralelos podem fornecer grande capacidade de armazenamento, atendendo as crescentes demandas das aplicações científicas.

2.4.1 Distribuição dos Dados e Primitivas de Acesso

Os usuários visualizam um arquivo paralelo como sendo um arquivo lógico único. Entretanto, o arquivo paralelo é composto por vários blocos de dados disjuntos fisicamente, também referenciados

como *stripes*. Os blocos de dados são distribuídos entre os servidores de armazenamento seguindo um padrão pré-estabelecido. Um padrão comum de distribuição é o chamado *striping*. Nesse padrão, os blocos de dados são distribuídos de forma circular (*round robin*) entre os servidores. Tipicamente, todos os blocos de dados têm o mesmo tamanho, visando facilitar a manipulação do arquivo. O tamanho do bloco pode ser definido pelo usuário no momento da criação do arquivo. A Figura 2.3 exibe a representação gráfica de um arquivo paralelo. Nesse caso, os blocos do arquivo paralelo estão distribuídos entre quatro discos.

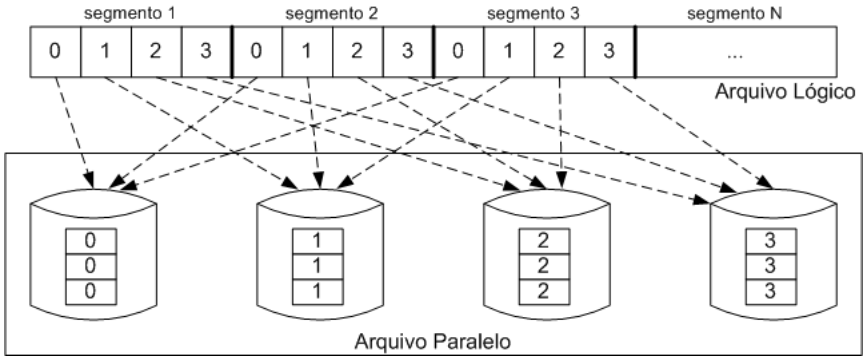


Figura 2.3: Arquivo paralelo.

As primitivas de acesso fornecidas devem ser transparentes aos usuários. O usuário deve acessar um sistema de arquivos paralelos da mesma forma como são acessados sistemas de arquivos locais e centralizados. Primitivas como *open*, *close*, *seek*, *rename* e *unlink* devem ser fornecidas por esses sistemas. As operações de E/S são executadas através das primitivas *read* e *write*.

A transparência no acesso tem por objetivo facilitar a execução de aplicações que utilizam bibliotecas de acesso padronizadas sem a necessidade de recompilação. Entretanto, o programador pode requerer maior controle sobre a forma de fracionamento e distribuição dos dados. Dessa forma é possível otimizar a aplicação para tirar proveito das características do sistema de arquivos alvo.

2.4.2 Arquitetura de Sistemas de Arquivos Paralelos

A maioria dos sistemas de arquivos paralelos existentes compartilha diversas características estruturais. Basicamente esses sistemas podem ser divididos em dois tipos genéricos de arquitetura, centralizada ou descentralizada, conforme tratam as operações de gerenciamento de metadados. A Figura 2.4 exibe uma representação gráfica dos dois tipos de arquiteturas. Para a arquitetura descentralizada não considerar o componente coordenador de serviços, em destaque.

A arquitetura mais simples é a descentralizada, onde existe apenas dois tipos de processos, clientes e servidores. Os processos servidores armazenam os blocos de dados de arquivos paralelos sobre um sistema de arquivos local. Enquanto que os processos clientes solicitam arquivos através de primitivas Unix para manipulação de arquivos. Em conjunto, ambos devem ser capazes de prover as funcionalidades de sistemas de arquivos paralelos.

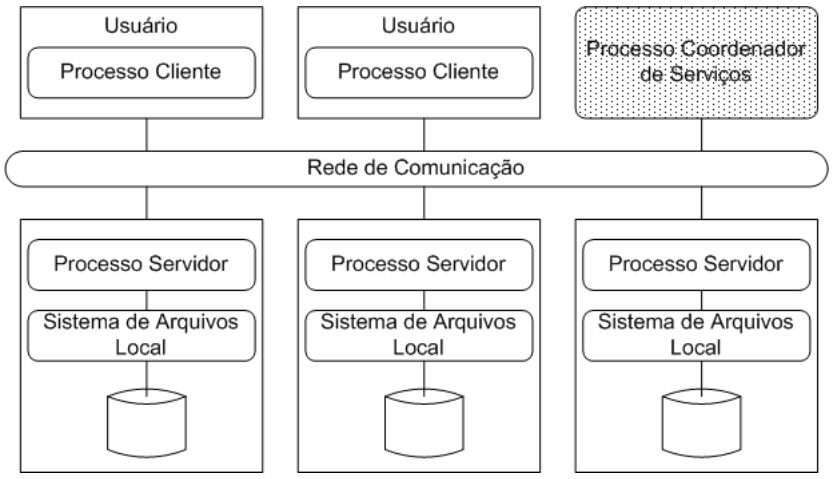


Figura 2.4: Arquitetura de sistema de arquivos paralelos.

Na arquitetura centralizada há um processo adicional responsável por gerenciar as operações de metadados do sistema de arquivos, chamado processo coordenador de serviços. Os metadados do arquivo incluem informações como, padrão de distribuição do arquivo entre os servidores, tamanho da unidade de distribuição, localização das unidades, entre outras. Operações como abertura, fechamento, remoção e renomeação são realizadas com intermédio do processo coordenador. Por exemplo, o cliente contata o processo coordenador de serviços, recebe as informações de metadados do arquivo, para então, acessar de forma direta os servidores de armazenamento.

Essas duas arquiteturas são comumente utilizadas no projeto e desenvolvimento de sistemas de arquivos paralelos. Entretanto, novos processos podem ser adicionados de acordo com as características da aplicação alvo ou sistema em uso.

2.4.3 Alguns Sistemas de Arquivos Paralelos

Nesta seção são listados alguns sistemas de arquivos paralelos. Esses foram escolhidos para exemplificar os dois tipos de arquiteturas discutidas anteriormente, centralizada e descentralizada. Todavia, há vários sistemas de arquivos paralelos implementados e em produção. Para maiores informações, nos trabalhos [13] e [20] foram apresentados alguns sistemas de arquivos paralelos existentes.

Expand

O Expand[10] utiliza um conjunto de servidores NFS para armazenar o arquivo paralelo. Todas as funcionalidades são implementadas nos clientes. Os metadados do arquivo são armazenados em um dos servidores NFS, denominado nó mestre. O cliente localiza o nó mestre através de uma função *hash* que recebe o nome do arquivo e retorna um inteiro. Esse inteiro define qual servidor mantém os metadados. Esse mecanismo aumenta a confiabilidade do sistema de arquivos ao descentralizar os metadados, porém cria alguns problemas. Por exemplo, ao renomear o arquivo o valor retornado pela função *hash* pode não ser o mesmo. Dessa forma, as informações de metadados devem ser migradas

de servidor, alterando a localização do nó mestre.

SPFS

Scotch Parallel File System (SPFS) [11] compartilha algumas características do Expand, como a distribuição das informações de metadados entre os servidores de armazenamento. Como diferencial, o SPFS utiliza de forma agressiva mecanismos de busca antecipada e escrita adiada de dados. Esses mecanismos são implementados através de *buffers* locais nos clientes.

Galley

Galley [27] é outro exemplo de sistema de arquivos paralelos semelhante ao Expand e SPFS. Em adição, o sistema provê uma interface simples e genérica que possibilita aos processos clientes controlar de forma explícita o paralelismo no acesso aos arquivos.

GlusterFS

GlusterFS [18] é um sistema de arquivos que agrega várias unidades de armazenamento interconectadas por redes *Infiniband* RDMA ou TCP/IP. Seu objetivo é criar um sistema de arquivos paralelos com grande capacidade de armazenamento e vazão de dados. É um sistema de arquivos que executa no espaço do usuário, não sendo necessárias alterações no *kernel* do sistema operacional. Isso permite instalar o GlusterFS sobre várias distribuições Linux (Debian, Solaris, BSD, Fedora, etc).

A arquitetura do GlusterFS é formada por dois tipos de componentes: servidor e cliente. Não há um servidor de metadados dedicado. Para acessar os arquivos, os clientes executam algoritmos nativos do GlusterFS para obter os metadados sob demanda. Essa característica elimina o gargalo no acesso aos metadados em servidor centralizado, da mesma forma que elimina o ponto central de falhas do sistema. A arquitetura do GlusterFS pode ser visualizada na Figura 2.5.

No lado servidor as unidades de armazenamento são chamadas de *bricks*. Tipicamente, um *brick* é um nó do *cluster* que compartilha partições de dados ou discos inteiros. Um *brick* pode conter um ou mais volumes, apontando para partições ou discos formatados com um sistema de arquivos local. No lado cliente, utiliza-se a biblioteca FUSE para acessar os volumes compartilhados pelos servidores de armazenamento. A configuração do cliente é feita através de arquivos descritores, chamados volfile, que informam quais volumes, forma de acesso e protocolo de comunicação serão utilizados para acessar os dados.

O modo como o cliente acessa o sistema de arquivos é definido por componentes chamados *translators*. Os *translators* são objetos que implementam todas as operações de sistema de arquivos. Quando GlusterFS recebe uma chamada de sistema de arquivos, está é encaminhada para o respectivo *translator* ou grupo de *translators*. Os *translators* podem ser agrupados de modo a definir o comportamento do sistema de arquivos. Por exemplo, é possível definir a forma de distribuição de arquivos (*striping*), ativar mecanismos para melhorar o desempenho de escrita e leitura (*write-behind*,

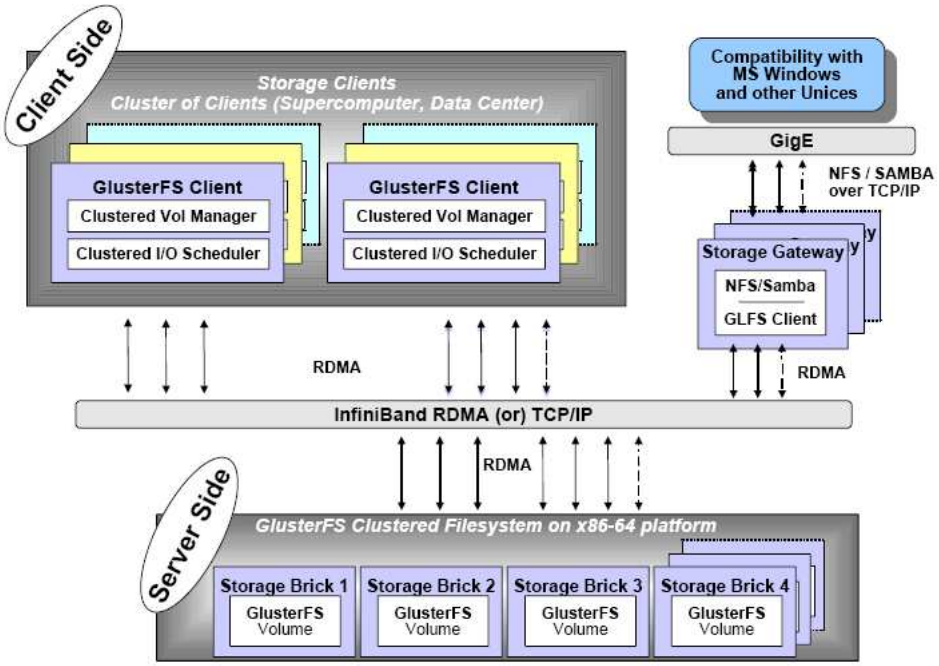


Figura 2.5: Arquitetura do GlusterFS [18].

read-ahead, *io-cache*, etc), habilitar ou não redundância de arquivos, definir tipo de protocolo de comunicação, etc. O usuário deve especificar os *translators* desejados nos arquivos de configuração de servidores e clientes (volfile). Além do mais, usuários podem adicionar novas funcionalidades ao GlusterFS através da implementação de novos *translators*.

Os pontos fortes do GlusterFS são a ausência de servidor de metadados centralizado, facilidade de instalação e manutenção, esta devida ao baixo nível de intrusão sobre o sistema operacional (FUSE). Entretanto, todos os clientes precisam ser configurados (volfile) para ter acesso às unidades de armazenamento. A adição de um novo servidor de armazenamento acarreta na reconfiguração dos clientes, não sendo uma operação transparente.

PVFS

Parallel Virtual File System (PVFS) [4] é um sistema de arquivos paralelos para *clusters* Linux. Tem como objetivo fornecer alta vazão de dados para aplicações paralelas. O PVFS é composto por: servidores de E/S, servidor de metadados e processos clientes. Todos os componentes podem estar localizados na mesma máquina. Entretanto, a distribuição destes em máquinas distintas trás benefícios como maior desempenho. A Figura 2.6 apresenta a arquitetura do PVFS.

A aplicação cliente quando necessita ler ou escrever dados consulta o servidor de metadados. A resposta informa em quais servidores de E/S os dados estão armazenados. De posse dessas informações, o acesso se dá de forma direta entre cliente e servidores de E/S, não envolvendo o servidor de metadados.

Os arquivos armazenados no PVFS são divididos através de um conjunto de servidores de E/S

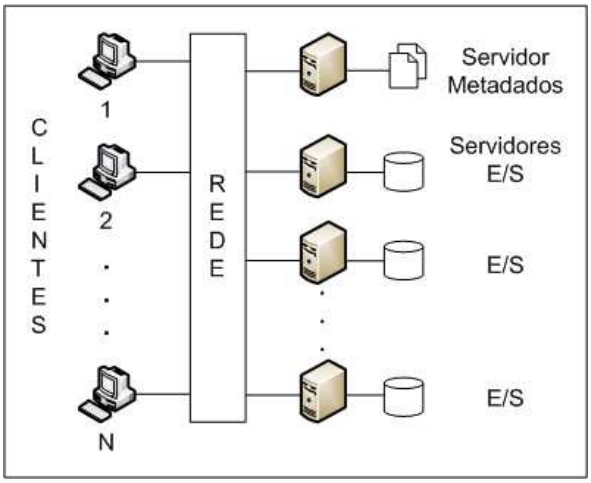


Figura 2.6: Arquitetura do PVFS.

para facilitar o acesso paralelo. A especificação a respeito da distribuição de determinado arquivo é descrita por três parâmetros de metadados: número do servidor de E/S base, número de servidores de E/S e tamanho da unidade de armazenamento em *kilobytes*. Dessa forma é possível determinar qual servidor de E/S armazena a primeira unidade de dados, por quantos servidores de E/S o arquivo estará distribuído e qual o tamanho de cada unidade de armazenamento, respectivamente.

O PVFS utiliza o sistema de arquivos local da máquina, não é necessária a instalação de um sistema de arquivos específico. Ambos, servidores de E/S e metadados, executam um *daemon* do PVFS. Resumidamente, o *daemon* escuta requisições das aplicações cliente e lê ou escreve dados nos dispositivos de armazenamento existentes. Quando um cliente executa uma operação de leitura ou escrita sobre o ponto de montagem do PVFS, as chamadas de sistema são interceptadas e redirecionadas ao *daemon*, impedindo a execução sobre o sistema de arquivos local. Esse mecanismo de interceptação permite que aplicações façam uso do PVFS sem a necessidade de recompilar o código fonte.

O PVFS fornece uma API (*Application Programming Interface*) nativa que permite acesso aos arquivos. Entretanto, também existe suporte para as APIs Unix/POSIX [16] e MPI-IO [12][8]. A API Unix/POSIX oferece suporte às funções *read*, *write* e comandos *shell* Unix como *ls*, *cp* e *rm*.

O PVFS pode ser utilizado em conjunto com um sistema de replicação de dados para prover tolerância a falhas. Este mecanismo evita a perda de dados na ocorrência de problemas físicos em servidores. Porém, esse sistema puro de replicação não provê alta disponibilidade para o sistema de arquivos. Por exemplo, quando um servidor para de responder por problemas físicos ou problemas na rede, nenhum outro servidor toma o seu lugar para dar continuidade aos serviços de arquivo.

pNFS

pNFS [14] é uma extensão do protocolo NFSv4. O seu desenvolvimento está sob responsabilidade do IETF (*Internet Engineering Task Force*). O principal objetivo é prover maior escalabilidade e desempenho em relação ao NFS. A principal mudança implementada foi a separação entre os fluxos

de controle e fluxos de dados.

Dessa forma, um cliente pode acessar em paralelo vários servidores de E/S. O pNFS permite a exportação de toda a hierarquia de diretórios ou parte dela, assim como no NFS. A arquitetura do pNFS é composta pelos componentes *driver* de *layout*, *driver* de E/S e interface de recuperação de *layouts* [14]. A Figura 2.7 ilustra a arquitetura do pNFS.

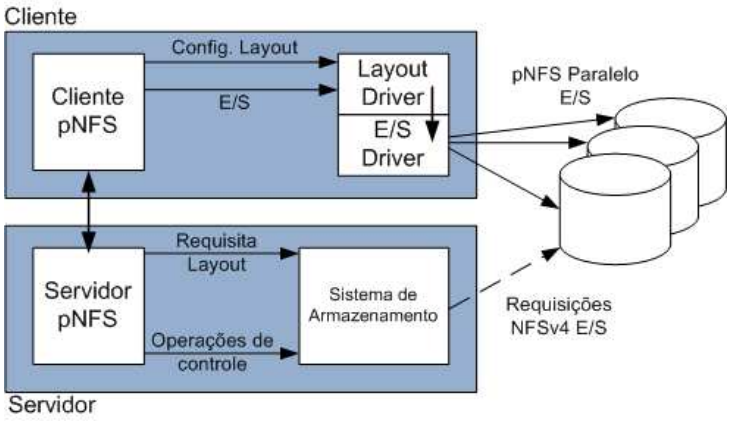


Figura 2.7: Arquitetura do pNFS.

Para realizar o acesso ao servidor pNFS, o cliente inicialmente solicita o *layout* do arquivo. Um *layout* consiste em toda informação necessária para acessar qualquer faixa de bytes dentro do arquivo. Por exemplo, informações sobre tamanho do bloco de dados, deslocamento do primeiro bloco sobre cada dispositivo de armazenamento e número de blocos. De posse do *layout* do arquivo, o componente *driver* de *layout* traduz as requisições de leitura e escrita do cliente pNFS em operações de E/S interpretáveis pelos dispositivos de armazenamento. O componente *driver* de E/S possui suporte a operações de E/S sobre redes *Myrinet*, *Infiniband* e TCP/IP.

Como mencionado anteriormente, o pNFS é uma extensão do protocolo NFSv4. A semântica de acesso a arquivos é a mesma fornecida pelo NFSv4. No momento da escrita deste trabalho, o pNFS não permite replicar o servidor de metadados. Dessa forma, tolerância a falhas e escalabilidade está limitada pelo servidor de metadados único.

Lustre

O Lustre [34] tem arquitetura semelhante ao PVFS e pNFS. É composto por: servidor de metadados (MDS), servidores de armazenamento (OSS) e processos clientes (OSC). Como PVFS e pNFS, os fluxos de dados e controle foram separados, permitindo o acesso em paralelo a partes distintas do arquivo. No próximo Capítulo, serão apresentadas maiores informações sobre a arquitetura e funcionamento do Lustre.

2.5 Uso de Benchmarks para Avaliar Sistemas de Arquivos

Para fornecer uma ideia do quão rápido pode ser um sistema computacional, geralmente são utilizados *benchmarks*. Estes são capazes de imitar o comportamento de aplicações reais, fornecendo subsídios para medir o desempenho de *software* e *hardware*. De maneira ideal, para analisar o desempenho de um sistema deveria-se usar aplicações reais, com cargas de trabalhos reais, mas isso às vezes é impraticável, por consumir muito tempo. Isso implica no aprendizado da nova aplicação, envolvendo acerto de configurações, migração de dados para o novo ambiente, além de ser necessário tratar possíveis erros existentes no código. Por isso, *benchmarks* são ferramentas geralmente empregadas na avaliação de desempenho de sistemas. Entretanto, é preciso tomar cuidado na escolha de quais *benchmarks* serão utilizados. É importante compreender o funcionamento dessas ferramentas e como interpretar os resultados gerados, a fim de selecionar *benchmarks* que representem da melhor forma possível o comportamento de aplicações reais.

Existem diversos *benchmarks* direcionados para analisar o comportamento e desempenho de sistemas computacionais. Em [43] os *benchmarks* são classificados em três categorias:

- *Macrobenchmarks*: avaliação de desempenho através de cargas de trabalho reais, tipicamente se utiliza a carga de trabalho de uma aplicação real.
- *Trace Replays*: a carga de trabalho de uma aplicação real é gravada para posterior execução.
- *Microbenchmarks*: são projetados para medir o desempenho de partes específicas de um sistema, a fim de identificar possíveis gargalos.

Neste trabalho, o Lustre será avaliado sob cargas de trabalho de aplicações paralelas reais. Para realizar essa tarefa é necessário selecionar um *benchmark* configurável e escalável, que possa avaliar diversos fatores relacionados as operações de E/S, e que possa também, ser utilizado em ambientes com poucos ou muitos processadores/nós.

A seguir, são listados alguns *benchmarks* que visam simular o comportamento de E/S de aplicações paralelas científicas:

- FLASH I/O [9] é um *benchmark* baseado em uma aplicação real voltada para área de astrofísica termonuclear. Em intervalos regulares de tempo, todos os processos transferem dados ao sistema de arquivos, a fim de armazenar o estado da aplicação (*checkpoint*). Este *benchmark* executa principalmente operações de escrita, usando a biblioteca paralela HDF5.
- MADbench2 [2] é derivado de uma aplicação paralela real, que analisa conjuntos de dados CMB (*Cosmic Microware Background*) provenientes de vários satélites. Essa análise sobre grandes massas de dados gera intensa atividade de E/S. MADbench2 é um *benchmark* parametrizável, possibilita o uso de POSIX ou MPI-IO, acesso síncrono ou assíncrono, uso de arquivos exclusivos ou compartilhados entre os processos, além dos parâmetros relacionados à aplicação de análise de dados CMB.

- O *benchmark* BTIO [45] é baseado em um código para resolução de equações de *Navier-Stokes*, através de um método de blocos tri-diagonais (BT). Cada nó processa uma grade de 5 x 5 blocos e armazena os resultados em disco. Isso gera um número significante de operações de E/S. O BTIO faz parte do NAS *Parallel Benchmarks* (NPB) , um conjunto de programas utilizados para avaliar o desempenho de máquinas paralelas, derivado de aplicações relacionadas à dinâmica de fluídos.

- MPI Tile I/O [32] avalia o desempenho da biblioteca MPI-IO sob uma carga de trabalho com acessos não contíguos. Neste *benchmark*, os dados são divididos em uma estrutura bi-dimensional, formando uma região ladrilhada (*tiled*), semelhante a um tabuleiro de xadrez. Essa carga de trabalho representa aplicações de visualização, onde a cena é dividida em vários *tiles*, que são processados em paralelo.

- *Effective I/O benchmark* (b_eff_io) [31] faz uso de MPI para executar uma série de configurações pré-definidas, a fim de derivar um número que representa o desempenho de E/S alcançado pelo sistema. Usa três métodos de acesso para medir o desempenho do sistema: escrita inicial, reescrita e leitura. Simula os padrões de acesso *strided* e *segmented*, com ou sem operações coletivas, para aplicações que acessam um arquivo compartilhado. E, simula acesso não coletivo para aplicações onde cada processo MPI acessa um arquivo exclusivo. Apesar de simular cinco tipos de padrões de acesso, não é um *benchmark* que fornece muitas opções de configuração. Alguns parâmetros não podem ser alterados, limitando o ajuste fino da carga de trabalho.

- O IOR (*Interleaved or Random*) [23] é um *microbenchmark* altamente configurável capaz de simular a maioria dos padrões de acesso de aplicações paralelas científicas. Permite simular aplicações que acessam arquivos compartilhados ou exclusivos, usando interfaces como POSIX, MPI-IO, HDF5 e parallelNetCDF para realizar E/S. Este *benchmark* será apresentando em maiores detalhes a seguir.

Os *benchmarks* FLASH I/O, MADbench2, BTIO e MPI Tile I/O são derivados de aplicações paralelas reais. Alguns deles são parametrizáveis, porém dentro do escopo do padrão de acesso da aplicação alvo. O b_eff_io é capaz de simular cinco padrões de acesso, mas não oferece muitas opções para ajuste fino da carga de trabalho. No entanto, este trabalho visa avaliar o Lustre sob cargas de trabalho de aplicações paralelas reais, sendo necessário selecionar um *benchmark* flexível e altamente parametrizável. Entre as opções de *benchmarks* investigadas, selecionou-se o IOR (*Interleaved or Random*), por ser considerada a opção mais adequada. Maiores detalhes do IOR serão apresentados a seguir.

2.5.1 Benchmark IOR

O IOR (*Interleaved or Random*) [23] possibilita a execução de operações de escrita e leitura em paralelo sobre arquivos de tamanhos diversos. Faz uso da biblioteca de passagem de mensagens MPI

[8] para sincronizar as operações entre processos. Possibilita definir o número de arquivos que cada processo acessa, ou se um arquivo compartilhado será acessado por todos os processos. Também fornece suporte as interfaces POSIX, MPI-IO, HDF5 e parallelNetCDF.

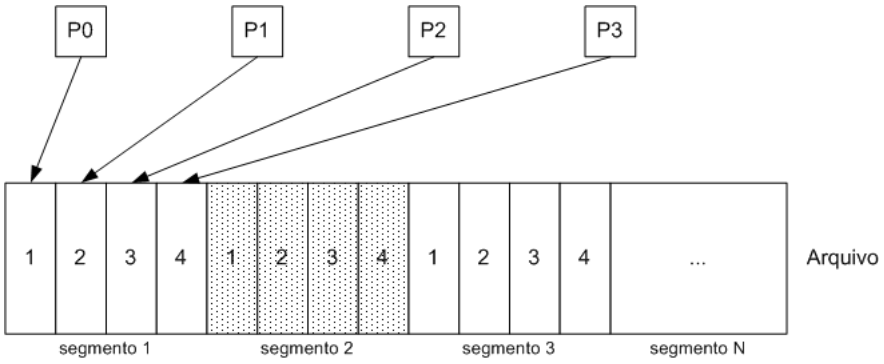


Figura 2.8: Padrão de acesso desempenhado pelo *benchmark* IOR.

A Figura 2.8 ilustra um exemplo de padrão de acesso que pode ser representado pelo IOR. Nesse caso todos os processos acessam um arquivo compartilhado. O arquivo é dividido em segmentos e cada segmento é dividido em blocos. O número de blocos por segmento está relacionado ao número de processos envolvidos na execução. Os segmentos podem representar passos de uma simulação ou uma variável de dados da aplicação paralela. Por exemplo, na Figura 2.8 os segmentos possuem quatro blocos de dados. Cada processo faz acesso ao seu respectivo bloco. Somente após todos os processos concluírem o acesso ao segmento 1 os blocos do segmento 2 serão acessados. O IOR fornece parâmetros para definir o número de segmentos e tamanho dos blocos de dados, a combinação dos mesmos determina o tamanho do arquivo. Além disso, o IOR permite definir o tamanho das transferências de dados entre memória e arquivo. Tipicamente são necessárias várias transferências para escrever/ler um bloco inteiro de dados.

O exemplo anterior representa aplicações onde todos os processos acessam um arquivo compartilhado. Para aplicações onde existe um arquivo por processo, a forma de acesso é semelhante, porém cada processo escreve ou lê blocos de dados em um arquivo exclusivo.

O IOR é capaz de representar grande parte das operações de E/S executadas por aplicações paralelas científicas. Além de ser uma ferramenta parametrizável, possibilita a automatização de testes, facilitando a comparação entre sistemas. Dessa forma, não será necessário desenvolver um novo *benchmark*, o IOR será utilizado na avaliação do Lustre.

3. SISTEMA DE ARQUIVOS LUSTRE

Este Capítulo apresenta o sistema de arquivos paralelos Lustre. Por se tratar do objeto de estudo deste trabalho, aqui são apresentadas maiores informações sobre sua arquitetura, forma de armazenamento de arquivos e *striping* de arquivos, tipos de redes suportadas, interfaces de acesso e demais características gerais.

A escolha do Lustre como objeto de estudo, se deve principalmente pela sua grande utilização entre os pesquisadores e centros de alto desempenho. Entre os dez primeiros supercomputadores da lista TOP500 [24], seis utilizam o Lustre. Existem outros sistemas de arquivos paralelos, que inclusive, compartilham algumas características em termos de arquitetura com o Lustre. Porém, o Lustre é o que possui maior visibilidade, motivo que o torna um sistema de arquivos alvo de vários estudos.

Lustre é um sistema de arquivos paralelos *open source* para *clusters* Linux com objetivo de prover alta escalabilidade, eficiência e redundância [34]. Surgiu em 1999 como um projeto de pesquisa da Universidade de *Carnegie Mellon*, USA. Entretanto, somente em 2003 a primeira versão foi liberada para *clusters* de produção. Atualmente, a Sun Microsystems é responsável pelo desenvolvimento e suporte do Lustre, disponibilizando o mesmo sobre os termos e condições da GNU *General Public License* (GPL).

O Lustre visa atender as necessidades de aplicações intensivas em dados, permitindo criar sistemas de arquivos com grande capacidade de armazenamento e vazão de dados. Por ser um sistema escalável, tipicamente é utilizado como sistema de arquivos global para todos os *clusters* do site. Isso facilita as tarefas de gerenciamento, evitando a manutenção de várias cópias de dados distribuídas entre vários sistemas de arquivos.

Este Capítulo está organizado da seguinte forma: a seção 3.1 apresenta a arquitetura do Lustre. A seção 3.2 comenta as principais características e limites do sistema de arquivos. Por fim, a seção 3.3 apresenta o cenário de estudo, onde serão realizados os testes sobre o Lustre.

3.1 Arquitetura

A arquitetura do Lustre é baseada em quatro componentes [17]:

- *Metadata Server* (MDS): servidor responsável por gerenciar todas as operações relacionadas aos metadados do sistema de arquivos. Os metadados incluem informações sobre a estrutura de diretórios, tamanho de arquivos, atributos e permissões. Isso inclui também as informações sobre a localização dos objetos de dados, distribuídos entre os servidores de armazenamento do sistema de arquivos. Em adição, o MDS tem associado um componente MDT (*Metadata Target*), dispositivo responsável por persistir os metadados.

- *Object Storage Servers* (OSS): servidores responsáveis por fornecer serviços de arquivos. Isso inclui serviço de E/S e manipulação das requisições sobre os OST locais.
- *Object Storage Target* (OST): dispositivos responsáveis por armazenar os objetos de dados. Tipicamente são utilizados discos magnéticos. Um servidor OSS pode gerenciar vários OSTs (geralmente entre dois e oito OSTs).
- *Object Storage Client* (OSC): biblioteca de *software* que permite acesso remoto ao sistema de arquivos Lustre por parte dos clientes. Em um sistema de arquivos Lustre, os clientes tipicamente são nós de computação, visualização ou estações de trabalho. Todos os clientes que montam o sistema de arquivos visualizam sempre um espaço de nomes único, coerente e sincronizado.

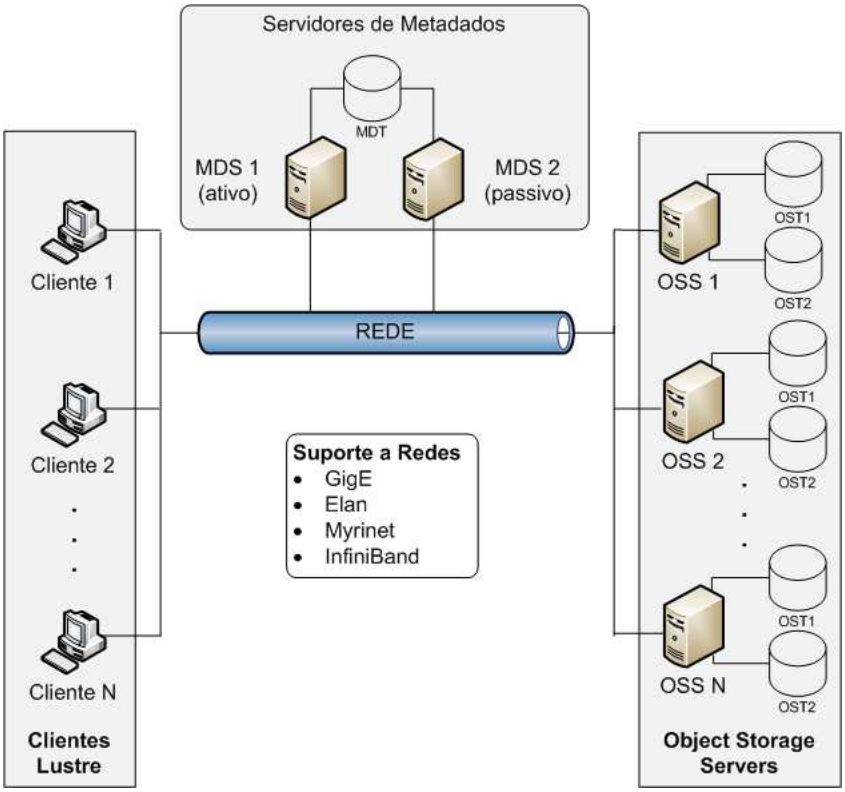


Figura 3.1: Arquitetura do Lustre.

A Figura 3.1 ilustra a arquitetura do Lustre. Todos os componentes são interconectados por uma rede de comunicação.

3.1.1 Armazenamento de Arquivos

No Lustre todos os arquivos são tratados como objetos. Dependendo do tamanho do arquivo e configurações de fracionamento e distribuição, um arquivo pode estar contido em um ou mais objetos. Os objetos são responsáveis por armazenar os dados do arquivo e são distribuídos entre os servidores

OSS. O Lustre utiliza o sistema de arquivos adjacente dos OSTs para armazenar os objetos, ou seja, os objetos na realidade são arquivos armazenados nos OSTs. Através do servidor MDS, os clientes são capazes de localizar todos os objetos do arquivo e, com posse dessas informações, podem contatar de forma direta os servidores OSS para executar operações de E/S.

Em sistemas de arquivos Unix são utilizados *inodes* para representar cada arquivo armazenado. Um *inode* é uma estrutura de dados que armazena informações sobre um arquivo, como permissões, atributos e localização física. No Lustre, o *inode* é utilizado de forma semelhante. Entretanto, em vez de apontar para blocos de dados, o *inode* aponta para os objetos que compõem o arquivo (Figura 3.2).

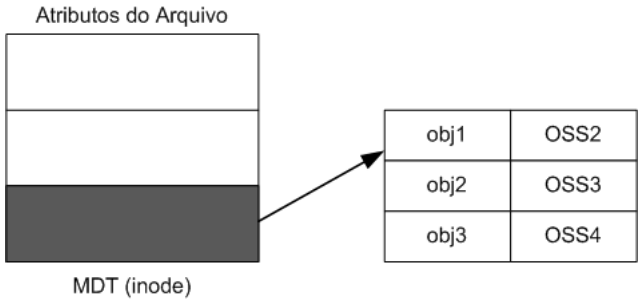


Figura 3.2: Representação de um *inode* no Lustre.

A criação de arquivo causa a alocação e configuração de um *inode*. No Lustre, a responsabilidade de criar um novo arquivo é atribuída ao servidor MDS. O cliente envia a requisição para o MDS, que faz a criação do *inode* local e contata os servidores OSS ordenando a criação dos objetos do arquivo. Ao concluir a operação, o MDS retorna as informações aos clientes, que a partir deste ponto, enviam os dados diretamente aos servidores OSS.

A escrita e leitura de arquivos ocorrem de maneira semelhante. O cliente acessa o servidor MDS e recebe um *inode*. Este *inode* tem uma lista de objetos e servidores OSS responsáveis pelo armazenamento do arquivo. De posse destas informações, o cliente passa a acessar diretamente os objetos. Caso o arquivo seja composto por apenas um objeto, significa que este objeto contém todos os dados do arquivo. Por outro lado, a existência de mais de um objeto significa que o arquivo está distribuído (*striped*) por vários OSS.

3.1.2 *Striping* de Arquivos

O *striping* permite que partes do arquivo possam ser armazenadas em diversos OSTs. A Figura 3.3 ilustra a distribuição de arquivos sobre um certo número de objetos. O número de objetos é chamado de *stripecount*. Cada objeto contém blocos de dados. O tamanho do bloco é chamado de *stripesize*. Quando o tamanho dos dados excede o *stripesize*, o restante é armazenado no próximo OST. A Figura 3.3 ilustra o armazenamento de dois arquivos com *stripecount* 2 e 3, e *stripesize* diferente entre os arquivos.

O *striping* de arquivos trás alguns benefícios. Por exemplo, o tamanho máximo do arquivo não está limitado ao tamanho de um único dispositivo de armazenamento. No caso do Lustre, cada OSS

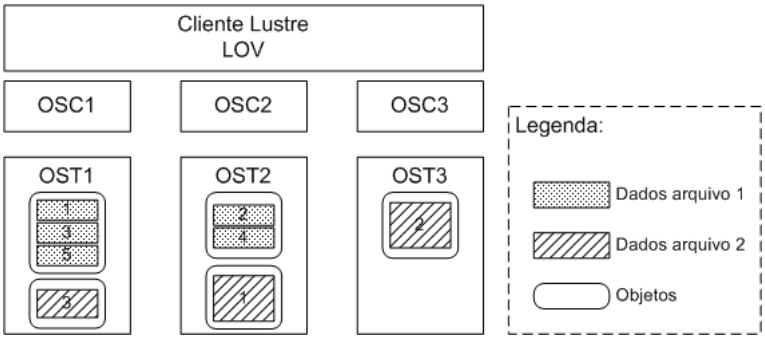


Figura 3.3: Dois arquivos distribuídos com *stripecount* 2 e 3, e *stripesizes* distintos.

pode armazenar até 8 TB. Outro benefício é o ganho com a largura de banda agregada. Como o arquivo está distribuído entre vários dispositivos, a largura de banda de cada dispositivo pode ser agregada, melhorando o desempenho. Essa característica permite o acesso em paralelo a diferentes partes do arquivo.

Os servidores OSS manipulam as interações entre as requisições de clientes e os dispositivos físicos de armazenamento. Os dispositivos de armazenamento são referenciados como *Object-Based Disks* (OBDs), mas os dispositivos não estão limitados somente a discos rígidos. Na realidade, os OBDs são representados por *drivers* de dispositivos. Desta forma é possível adicionar novos dispositivos de armazenamento facilmente ao Lustre.

3.1.3 Rede

Toda a infraestrutura de comunicação necessária para conectar clientes e servidores de sistemas de arquivos Lustre é fornecida pela API LNET (*Lustre Networking*). Tipicamente, os dispositivos de armazenamento, discos magnéticos, por exemplo, são conectados aos servidores de MDS e OSS através da tecnologia SAN (*Storage Area Network*) tradicional. A LNET atua somente na conexão entre clientes e servidores, fornecendo interoperabilidade entre diversos tipos de redes, incluindo:

- InfiniBand: OpenFabrics 1.0 e 1.2, Mellanox Gold, Cisco, Voltaire e Silverstorm.
- TCP: qualquer tecnologia com suporte a TCP, como GigE, 10GigE e IPoIB.
- Quadrics: Elan3 e Elan4.
- Myricom: GM e MX.
- Cray: Seastar, RapidArray.

O suporte aos vários tipos de redes permite o uso do Lustre como sistema de arquivos global em grandes instalações. A Figura 3.4 ilustra o uso do Lustre como sistema de arquivos global para dois *clusters* com tecnologias de redes distintas.

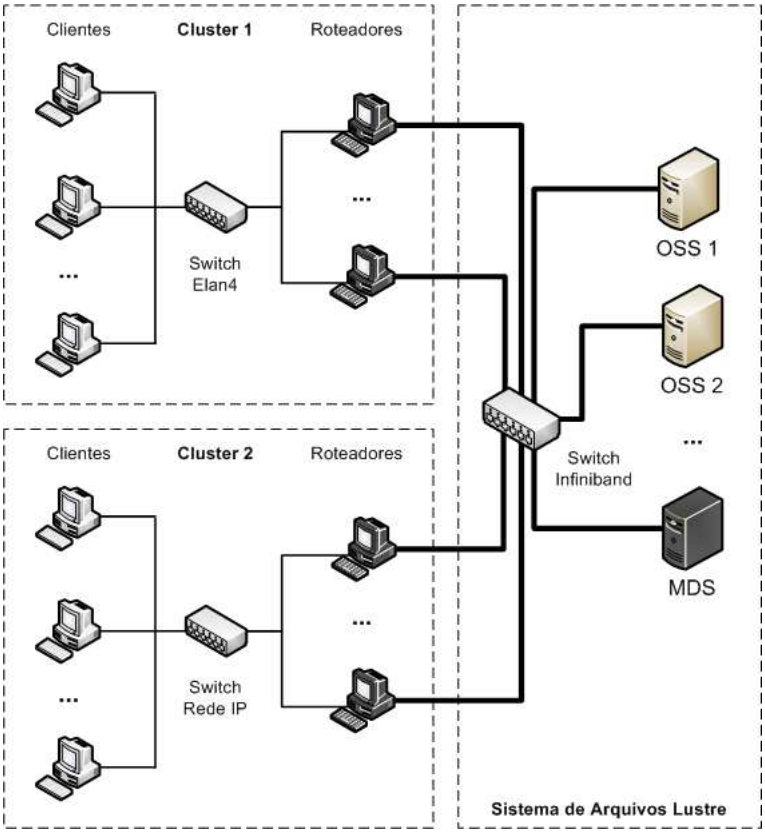


Figura 3.4: Sistema de arquivos global usando LNET.

Neste caso, alguns nós atuam como roteadores. É possível configurar um servidor OSS para atuar também como roteador. A possibilidade de usar o Lustre como um sistema de arquivos global trás vantagens no gerenciamento de arquivos, quotas de usuários e permissões de acesso. Além disso, a LNET fornece suporte a *channel bonding* e balanceamento de carga. Toda a flexibilidade oferecida pela LNET tem por objetivo minimizar gargalos na comunicação entre *clusters* e sistemas de arquivos Lustre.

3.2 Características Gerais

3.2.1 Interfaces de Acesso

Lustre segue as normas POSIX [16] para manipular arquivos. POSIX (*Portable Operating System Interface*) é um conjunto de padrões e interfaces para acesso ao sistema operacional. No contexto de sistemas de arquivos, POSIX define que a maioria das operações devem ser atômicas, garantindo semântica no acesso aos dados. Sendo assim, o usuário tem a visão de estar operando um sistema de arquivos local [17].

A forma de montar sistemas de arquivos Lustre é semelhante ao NFS. Após montar o sistema, o usuário visualiza um espaço de nomes único. A manipulação de arquivos pode ser realizada através de comandos *shell* Unix como *ls*, *cp*, *rm*, etc. Através das ACL (*Access Control List*) POSIX é possível

definir as permissões de acesso aos arquivos e diretórios. O controle é semelhante ao utilizado em sistemas Unix, com três classes de usuários (proprietário, grupo e outros) e permissões para leitura (r), escrita (w) e execução (x).

3.2.2 Limites do Lustre

A documentação do Lustre trás as seguintes informações sobre os limites de arquivos e sistema de arquivos:

- Número máximo de *stripes*: um arquivo pode ser fracionado e distribuído por até 160 OST.
- Tamanho do *stripe*: para máquinas 64 bits, o tamanho mínimo do *stripe* é definido pelo tamanho da página de memória, o mínimo padrão é de 64kB. O tamanho máximo do *stripe* é de 2 TB. Para máquinas de 32 bits, o resultado da expressão ($stripe\_size * stripe\_count$) deve ser menor que 4 GB.
- Tamanho do arquivo: o tamanho máximo de um arquivo em máquinas 32 bits é de aproximadamente 16 TB. Em plataformas de 64 bits o tamanho do arquivo está limitado a 2^{16} *bytes*. Porém, existe a limitação do tamanho do *stripe* a 2 TB e o número de *stripes* em 160, mesmo assim, isso permite arquivos de 320 TB.
- Tamanho máximo do sistema de arquivos: sistemas operacionais Linux com kernel 2.6 limitam os dispositos de bloco a 8 TB, ou seja, cada OST pode ter um sistema de arquivos de 8 TB (limite imposto pelo *ext3*). Sendo que o Lustre tem um limite definido no código fonte de 8150 OST por sistema de arquivos, é possível ter um sistema de arquivos teórico de 64 PB.
- Número máximo de clientes: atualmente o número máximo de clientes Lustre é de 131072.

Alguns desses limites são definidos no código fonte do Lustre e são possíveis alterações, desde que o Lustre seja recompilado. Essas informações indicam que o Lustre é um sistema escalável, limitado ao *hardware* disponível.

3.2.3 Administração do Lustre

O Lustre provê utilitários via linha de comando para gerenciar o comportamento do sistema de arquivo e forma de distribuição de *stripes* entre os servidores de armazenamento. Dois utilitários utilizados com frequência são *lctl* e *lfs*.

Utilitários de gerenciamento

Através do utilitário *lctl* é possível executar tarefas administrativas, como configurar parâmetros do sistema de arquivos e depurar de erros. Por exemplo, o *lctl* provê meios de desativar determinado OSS para troca de *hardware* ou execução de *backup* e, ativar novamente após manutenção.

Configurações de rede, *timeouts* e recuperação de informações sobre os componentes do Lustre são tarefas geralmente realizadas através dessa ferramenta.

O utilitário *lfs* permite criar um novo arquivo ou diretório com um padrão específico de distribuição. Da mesma forma é possível consultar informações sobre os objetos do arquivo e sua localização entre os OSS. Também é possível definir o padrão de distribuição de diretórios, onde todos os subdiretórios e arquivos contidos adotarão o mesmo padrão. Por exemplo, no momento da criação do arquivo é possível definir os parâmetros *stripesize*, *stripecount* e *index*, que representam respectivamente, o tamanho dos *stripes* em que o arquivo será fracionado, o número de OSS em que serão distribuídos os *stripes* e, por fim, o índice do primeiro OSS a receber o primeiro *stripe*.

Adição de OSS

O Lustre permite a adição de novos OSS e OST ao sistema de arquivos sem interromper os serviços de arquivos aos clientes. O procedimento de adição de novos OSS ou OST é semelhante ao executado durante a criação do sistema de arquivos. Novos OST tornam a distribuição de espaço livre desbalanceada. Tarefas simples como copiar os arquivos antigos para nova localização dentro do mesmo sistema de arquivos redistribuem os *stripes* sobre os novos OSTs. Esse tipo de tarefa pode facilmente ser realizada com auxílio de *scripts*.

Gerenciamento do espaço livre

O servidor de MDS usa duas formas para alocar os *stripes* entre os OSS: *(i)* forma circular ou *(ii)* ponderada de acordo com o espaço livre. A forma circular (*round-robin*) tem melhor desempenho por maximizar o uso da banda de rede agregada. Entretanto, a forma ponderada é utilizada quando a diferença de espaço de armazenamento entre servidores OSS é superior a 20%. O administrador do sistema pode alterar esse comportamento via utilitário *lctl*.

Quotas

O controle de quotas permite limitar a quantidade de disco que o usuário ou grupo pode usar em um diretório. As quotas são configuradas pelo usuário *root*. Além da quota para espaço em disco existe o controle similar para o número de arquivos permitidos. As quotas no Lustre diferem do Linux em várias formas:

- A administração das quotas é configurada através do comando *lfs*, após a montagem do diretório.
- As quotas são distribuídas entre os componentes do Lustre (OSSs).
- Quando uma quota é habilitada, automaticamente todos os clientes do sistema de arquivos estão sobre seu efeito.

Tabela 3.1: Configuração das máquinas.

Clientes	Servidores
8 nós	4 nós
Dual Pentium III 1GHz	Intel Pentium 4 2.8GHz
DIMM PC133 256MB	2,5GB
9.1GB ULTRA3 10K SCSI	40GB SAMSUNG SP0411N
Gigabit Ethernet	Gigabit Ethernet

SNMP

É possível usar o protocolo SNMP (*Simple Network Management Protocol*) [22] para reportar informações sobre os componentes do Lustre e estado atual do sistema. Para tanto é necessário carregar um módulo do Lustre que permite o uso do SNMP e pacotes de monitoramento associados.

3.3 Cenário de Estudo

Vários trabalhos avaliaram o Lustre sobre *clusters* de grande escala [47], [46], [1], [37]. Neste trabalho, o foco está voltado para *clusters* de pequena escala, que de certa forma representam a realidade de muitos centros de pesquisas nacionais. A infraestrutura para a realização deste trabalho foi cedida pelo Laboratório de Alto Desempenho da PUCRS (LAD-PUCRS) [29], que tem como objetivo desenvolver pesquisas na área de computação de alto desempenho.

Nesta seção serão apresentadas as características físicas do ambiente de teste e configurações do Lustre.

3.3.1 Ambiente de Teste

O ambiente de teste disponibilizado tem doze máquinas divididas em duas classes, de acordo com a configuração dos equipamentos. A primeira classe possui quatro máquinas com maior capacidade de processamento e armazenamento. Estas serão responsáveis por hospedar os sistemas de arquivos. A segunda classe conta com oito máquinas biprocessadas Pentium III de 1GHz. Essas máquinas atuaram como clientes dos sistemas de arquivos. A configuração dos equipamentos está listada na Tabela 3.1. Todas as máquinas estão interconectadas através de um *switch Gigabit Ethernet*. O ambiente de teste pode ser visualizado na Figura 3.5.

As máquinas do experimento foram disponibilizadas para uso exclusivo dos testes, ou seja, não havia outros usuários compartilhando os mesmo recursos. Apenas os *softwares* necessários para simular a carga de aplicações paralelas foi instalado e configurado. Para fins comparativos, o NFS foi instalado em máquina da mesma classe de configuração do Lustre. Os *softwares* utilizados estão listados na Tabela 3.2.

Tabela 3.2: Relação de *softwares* do cenário de estudo.

Softwares	Versões
Sistema operacional	Linux Ubuntu 8.10 Server
Versão do *Kernel*	2.6.22.14
Sistemas de arquivos	Lustre 1.6.6 e NFSv3
MPI	MPICH2 versão 1.0.8p1
Benchmarks	IOR 2.10.1

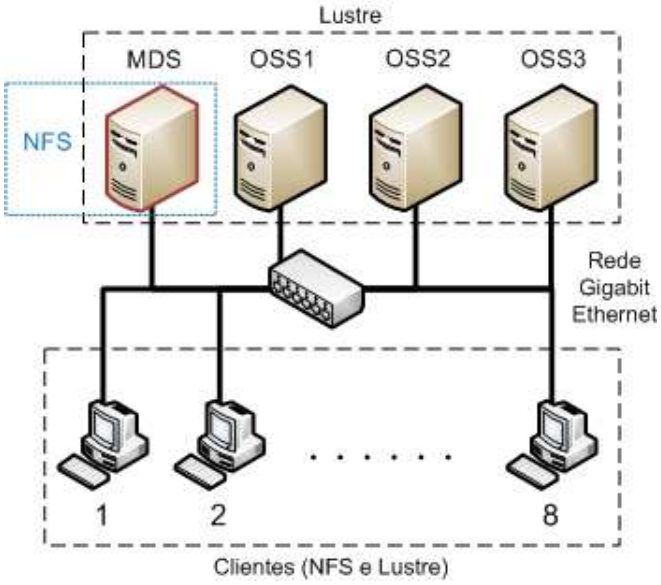


Figura 3.5: Representação gráfica do ambiente de teste.

3.3.2 Configuração do Lustre

Assumindo que o Lustre se encontra instalado em todas as máquinas do ambiente, esta seção discute alguns detalhes de como a configuração do sistema de arquivos foi realizada. Informações mais detalhadas sobre o processo de instalação são encontradas no manual do Lustre.

Um sistema de arquivos Lustre é composto por três componentes principais: MDS, OSS e clientes. Para configurar o sistema, esses três componentes foram distribuídos entre as máquinas da seguinte forma (Figura 3.5):

- Um servidor de metadados (MDS).
- Três servidores de armazenamento (OSS).
- Oito clientes Lustre.

É importante ressaltar que as máquinas servidoras (MDS e OSS) têm apenas um disco rígido. Dessa forma, o sistema operacional e sistema de arquivos Lustre compartilharão o mesmo disco. No momento da instalação do Linux foram reservados aproximadamente 23GB de espaço não par-

ticionado. O sistema de arquivos Lustre resultante tem aproximadamente 70GB de capacidade de armazenamento ($3 * 23GB$).

O processo de criação do sistema de arquivos é realizado através da formatação dos dispositivos ou partições dos servidores MDS e OSS. Obrigatoriamente é preciso formatar o MDS primeiro. No momento da formatação de cada OSS é necessário informar qual o servidor MDS associado. Essa formatação é executada via comando *mkfs* do Linux. Após, os clientes Lustre podem montar o sistema de arquivos de forma semelhante à utilizada pelo NFS.

Nenhum mecanismo de tolerância a falhas pertencente ao Lustre foi utilizado no ambiente de teste. A avaliação experimental não tem como objetivo verificar o comportamento do Lustre na ocorrência de falhas.

As configurações referentes ao fracionamento e distribuição dos arquivos foram definidas através do utilitário *lfs* no momento da criação dos diretórios de teste. Na descrição dos experimentos serão apresentadas maiores informações sobre a forma de fracionamento e distribuição de arquivos.

3.3.3 Configuração do NFS

O uso do NFS visa permitir a comparação com o Lustre em alguns experimentos da avaliação. O servidor NFS foi configurado na mesma máquina onde reside o serviço de MDS do Lustre (Figura 3.5). O objetivo é comparar ambos em termos de desempenho, a capacidade de armazenamento não será considerada.

O NFS permite tornar as operações de acesso assíncronas, melhorando o desempenho, mas não garantindo consistência no acesso aos dados. Visto que o Lustre faz uso de operações assíncronas, o NFS será configurado da mesma forma (uso do parâmetro *async* na instrução para exportar o sistema de arquivos).

4. AVALIAÇÃO E RESULTADOS OBTIDOS

Conhecer as características e limitações da plataforma disponível têm grande importância no projeto e desenvolvimento de aplicações paralelas. Muitos desenvolvedores não conhecem as características e limites das máquinas utilizadas no dia-a-dia, e muitas vezes, detalhes importantes como padrão de acesso e frequência que operações de E/S são executadas pelas aplicações. A falta de conhecimento sobre esses aspectos resulta em implementações ineficientes e de baixo desempenho.

Um dos aspectos que tem forte impacto no desempenho de aplicações paralelas é o acesso ao sistema de armazenamento. Esses sistemas são complexos, compostos por vários componentes, o que inclusive, torna a avaliação uma tarefa complicada. Há diversos fatores que influenciam no desempenho, como tipo de mídia de armazenamento utilizada (discos magnéticos, memória RAM volátil, memória RAM *flash*, unidades óticas, etc.), tipo do ambiente de armazenamento (LVM, RAID, etc.), rede de comunicação e demais componentes dos sistemas operacionais.

Por isso a importância de avaliar esses sistemas, de forma a verificar como aplicações podem ser desenvolvidas para obter maior desempenho, ou mesmo, descobrir se o sistema de armazenamento é adequado para a classe de aplicações alvo.

A fim de contribuir com esse processo, este Capítulo apresenta uma avaliação do sistema de arquivos paralelos Lustre sob cargas de trabalho de aplicações paralelas. O restante deste Capítulo está organizado da seguinte forma: a próxima seção discute a carga de trabalho de aplicações paralelas e padrões de acesso geralmente empregados. A seção 4.2 apresenta a metodologia utilizada na avaliação do Lustre. Por fim, nas seções 4.3 e 4.4 são apresentados os experimentos e resultados obtidos.

4.1 Carga de Trabalho

A avaliação realizada neste trabalho, tem foco no comportamento de E/S de aplicações paralelas intensivas em dados. Para reproduzir esse comportamento foram investigados alguns trabalhos que caracterizaram as operações de E/S de aplicações paralelas reais. A carga de trabalho da avaliação é baseada nos resultados das caracterizações apresentadas nesta seção.

4.1.1 Entrada e Saída

Aplicações paralelas geralmente acessam um ou mais arquivos durante a execução. A Figura 4.1 mostra um modelo de entrada/saída típico [35]. Uma aplicação paralela pode ser dividida em várias fases, onde em determinada fase, um processo pode acessar um ou mais arquivos. Os arquivos podem ser acessados de forma exclusiva ou compartilhada. Quando vários processos acessam um arquivo é preciso definir o nível de compartilhamento, ou seja, se os processos acessam regiões sobrepostas ou não. Todas essas características desempenham papel importante na forma como a aplicação interage com o sistema de arquivos, influenciando no desempenho final.

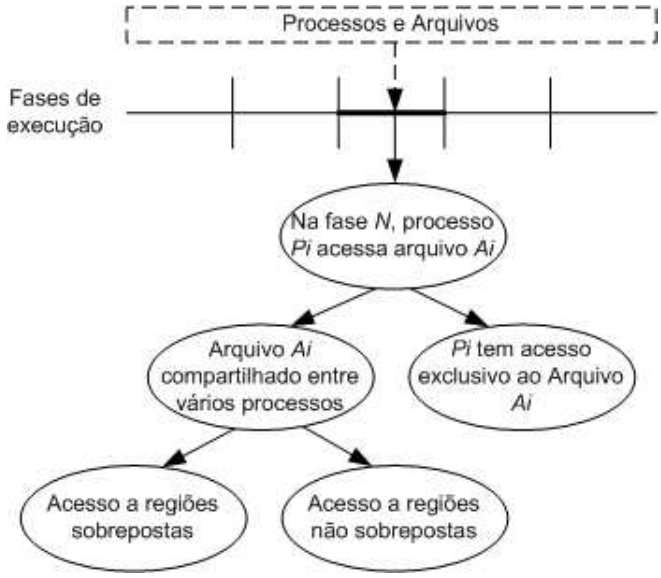


Figura 4.1: Modelo de entrada/saída de aplicações paralelas.

Em Miller e Katz [25], as operações de E/S de aplicações de alto desempenho foram divididas em três categorias: *required*, *data staging* e *checkpoint*. Em *Required*, as operações de E/S consistem na leitura inicial dos dados para processamento e escrita dos resultados obtidos. *Data staging* fornece suporte a aplicações onde os dados não cabem na memória principal do nó, sendo necessário manter os dados em disco. Esse processo pode ser realizado automaticamente pelo sistema operacional ou explicitamente pelo programador. *Checkpoint* consiste na escrita de resultados intermediários para prevenir perda de dados ou possibilitar o reinício, em casos de falhas no sistema ou aplicação. Algumas aplicações científicas demoram semanas ou até meses para completar a execução. *Checkpoint* nessas circunstâncias é de extrema importância. Entretanto, a frequência do *checkpoint* tem influência direta na confiabilidade e desempenho final, sendo o ajuste dependente da aplicação e ambiente de execução.

Alguns estudos caracterizaram a carga de trabalho de aplicações paralelas. Na década de 90, foram caracterizadas as operações de E/S de máquinas de memória compartilhada executando aplicações científicas [21], [30] e [28]. O projeto foi chamado de CHARISMA. Os resultados obtidos permitiram concluir que o tamanho das requisições aos arquivos é relativamente constante em cada aplicação. A maior parte das requisições de E/S são de tamanho pequeno, entretanto, 90% dos dados são transferidos através de requisições grandes. As operações de escrita são dominantes em relação à leitura, sendo infrequente escritas a dados compartilhados entre processos. Além disso, foram abordados os padrões de acessos desempenhados pelas aplicações avaliadas. Esses padrões de acesso serão utilizados na avaliação do Lustre e serão apresentados com maiores detalhes na próxima seção. Em 2004, um grupo de pesquisa da Universidade da Califórnia caracterizou a carga de trabalho de um *cluster* Linux de grande escala [44]. Os resultados obtidos foram semelhantes aos apresentados em CHARISMA. Em adição, os resultados apontaram o uso comum de um nó mestre para receber to-

dos os dados de nós escravos, para posterior armazenamento no sistema de arquivos. A caracterização também destacou, que o uso de um arquivo compartilhado apresenta desempenho cinco vezes inferior ao uso de um arquivo por processo ou nó.

4.1.2 Padrões de Acesso

Os padrões de acesso apresentados a seguir serão utilizados na avaliação do Lustre. A nomenclatura adotada aqui, será utilizada no restante do trabalho para referenciar os padrões de acesso.

File-per-proc

Neste padrão, cada processo (P) acessa um arquivo exclusivo, possibilitando a transferência de dados em paralelo. Como vantagem, essa abordagem possibilita realizar E/S em paralelo através de bibliotecas sequenciais. A principal desvantagem é a criação de vários arquivos ao invés de um. Isso pode dificultar a junção dos arquivos para servir de entrada a outra aplicação, pode dificultar tarefas como cópia, movimentação ou envio dos arquivos pela rede. Além disso, o reinício após falhas no sistema ou aplicação requer número de processos igual ao número de arquivos gerados. Em *clusters* compartilhados isso pode ser um problema.

Esse padrão de acesso é amplamente utilizado por aplicações paralelas. *Checkpoint* e escrita de resultados são exemplos comuns de uso. A Figura 4.2 mostra uma representação gráfica do padrão.

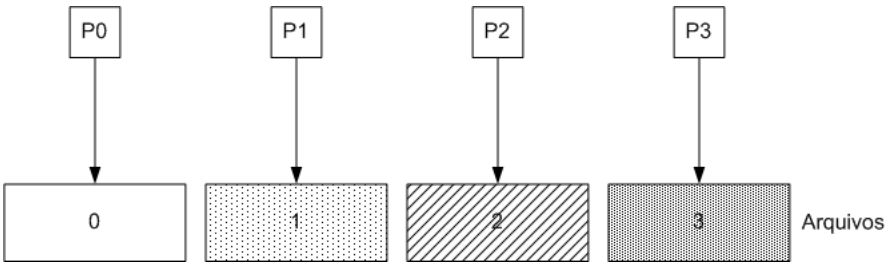


Figura 4.2: Padrão de acesso *file-per-proc*.

Single-shared-file

Todos os processos acessam um arquivo compartilhado. As duas vantagens principais são a criação de apenas um arquivo e a manutenção do acesso em paralelo. Além disso, esse modelo não prejudica a escalabilidade e desempenho da aplicação. Em *clusters* de grandes dimensões, certas aplicações podem criar milhares de arquivos durante a execução, sobrecarregando o sistema de arquivos. Essa estratégia geralmente é implementada com uso de bibliotecas de acesso paralelo. MPI-IO geralmente é empregada.

Single-shared-file pode ser subdividido de acordo com o padrão de acesso realizado pelos processos da aplicação. Os acessos podem ser a regiões contíguas ou não contíguas do arquivo, os blocos de

dados podem ser grandes ou pequenos, de tamanho fixo ou variável. Desta forma, *single-shared-file* será subdividido nos três padrões de acesso a seguir:

- *Segmented access*: o arquivo é dividido logicamente em segmentos de tamanho fixo. Cada processo (P) acessa um segmento do início ao fim. Por exemplo, um arquivo de 4096 *bytes* acessado por quatro processos. O processo 0 acessa o intervalo 0-1023, processo 1 o intervalo de 1024-2047, processo três o intervalo 2048-3071 e o processo quatro o intervalo de 3072-4096 *bytes*. Esse padrão assemelha-se ao *file-per-proc*, entretanto há apenas um arquivo compartilhado sendo acessado. A Figura 4.3 mostra uma representação gráfica do padrão.

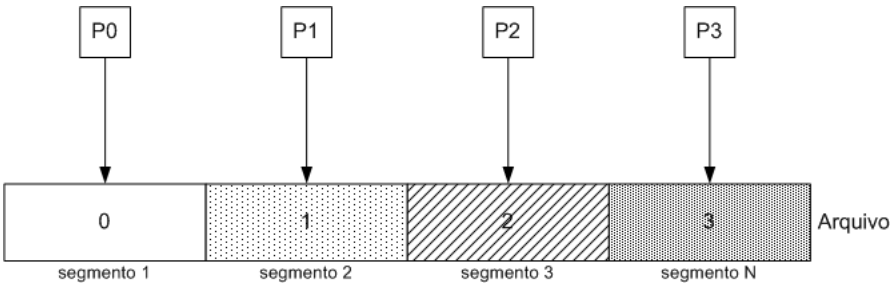


Figura 4.3: Padrão de acesso *segment access*.

- *Simple strided*: esse padrão divide o arquivo logicamente em segmentos, onde cada processo (P) acessa um intervalo de *bytes* do segmento, de acordo com um deslocamento definido. O deslocamento tipicamente é calculado em função do valor da variável *rank* do processo. Cada segmento do arquivo ocorre regularmente de acordo com um fator multiplicador. A soma de todas as regiões de dados dentro do segmento é chamada de "*stride distance*". Esse padrão é utilizado por várias aplicações paralelas. Como exemplo, pode-se citar a aplicação FLASH [9], voltada para área de astrofísica termonuclear. Em intervalos regulares de tempo todos os processos gravam em disco um arquivo de *checkpoint*. A Figura 4.4 mostra uma representação gráfica do padrão.

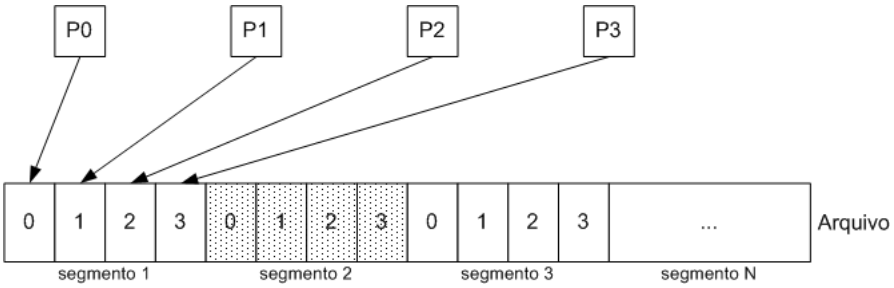


Figura 4.4: Padrão de acesso *simple strided*.

- *Random strided*: nesse padrão cada processo acessa um número de *bytes* aleatório ordenado de forma circular (*round robin*). Devido à forma aleatória, a cada ciclo o tamanho do segmento pode variar. Esse padrão pode ser utilizado por aplicações de codificação de mídia, onde o

tamanho do quadro resultante pode ser variável. Se cada processo gera um quadro de tamanho variável e então escreve logo após o quadro anterior, esse padrão pode ser aplicado. A Figura 4.5 mostra uma representação gráfica do padrão.

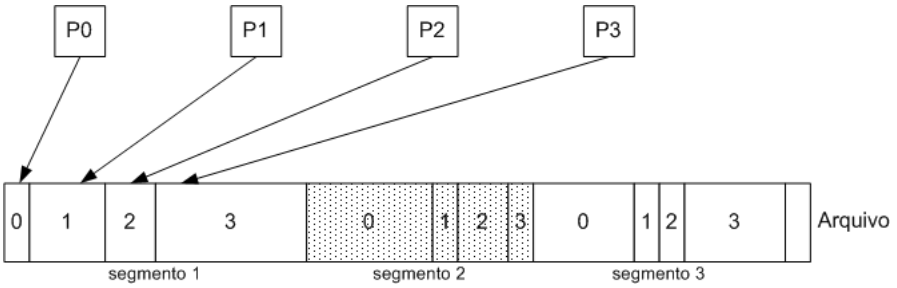


Figura 4.5: Padrão de acesso *random access.*

Para delimitar o escopo do presente trabalho, a avaliação do Lustre estará limitada aos quatro padrões de acesso citados anteriormente. A seleção dos mesmos se deve ao amplo uso em aplicações paralelas científicas (projeto CHARISMA). Esses padrões foram evidenciados em trabalhos que caracterizaram as operações de E/S de aplicações paralelas reais.

Todavia, existem outros padrões de acesso empregados em aplicações paralelas. Como exemplo de outros padrões, pode-se citar *nested strided* e *tiled* [38]. *Nested strided* é um padrão de acesso complexo que combina múltiplos acessos do padrão *simple strided.* Em cada bloco de dados do segmento tem associado um ou mais blocos de dados *simple strided.* Esse padrão pode ser utilizado no acesso a elementos de uma estrutura tridimensional, como um cubo armazenando em arquivo. No padrão *tiled*, o arquivo é dividido logicamente na forma de um ladrilho, onde cada processo é responsável por processar determinada região da área ladrilhada. Esse padrão, geralmente é utilizado por aplicações de visualização, podendo ser decomposto nos padrões *segmented access* ou *simple strided.*

4.2 Metodologia de Avaliação

Para avaliar sistemas computacionais é preciso definir uma metodologia de trabalho. É preciso definir quais serão os procedimentos, fatores e métricas que serão utilizados. Todo esse processo deve ser documentado, permitindo desta forma a reprodução dos experimentos e avaliações, fornecendo meios para que outros possam comparar os resultados obtidos. Assim, nesta seção serão apresentadas as métricas, procedimentos e também, como foram projetados e organizados os experimentos para avaliar o Lustre.

4.2.1 Métricas de Avaliação

A escolha de métricas depende dos objetivos que se deseja atingir. A seguir são apresentadas algumas métricas que podem ser usadas para avaliar sistemas de arquivos [5]:

- *Throughput*: medida de velocidade ou, a taxa na qual o sistema de arquivos entrega dados. O *throughput* pode ser medido de duas formas: (i) acessos/segundo, quando o tamanho das requisições são pequenas, ou (ii) *bytes*/segundo para requisições maiores.

- Tempo de resposta: tempo que o sistema demora a entregar dados. Pode ser medido através de vários pontos de vista, por exemplo, através da perspectiva do usuário, sistema operacional ou mesmo, através da perspectiva da controladora do disco.

- Capacidade: a capacidade de armazenamento também pode ser considerada como métrica para comparar sistemas de arquivos. Por exemplo, é possível comparar sistemas através da facilidade e limites de expansão da capacidade de armazenamento.

- Confiabilidade: outra métrica importante na avaliação de sistemas de armazenamento em geral. Há grande preocupação por parte dos pesquisadores em aprimorar a confiabilidade dos sistemas de armazenamento.

- Custo: o valor gasto para adquirir, instalar, configurar e manter sistemas de armazenamento é fundamental no momento de comparar soluções, muitas vezes determinante na escolha da solução de armazenamento.

Throughput e tempo de resposta são métricas comuns usadas na avaliação de sistemas de arquivos. Como o objetivo principal da avaliação é medir o desempenho do Lustre e apontar possíveis gargalos, o *throughput* será a principal métrica utilizada.

4.2.2 Projeto de Experimentos

Os experimentos foram divididos em duas classes de testes, com foco no ambiente de teste e foco nos padrões de acesso de aplicações paralelas.

Foco no Ambiente de Teste

A avaliação sob a perspectiva do ambiente de teste visa investigar o comportamento do Lustre ao variar fatores como tamanho do arquivo, tamanho da unidade de transferência, número de clientes, número de servidores de armazenamento, distribuição da carga entre os servidores de armazenamento e acesso não alinhado ao tamanho do *stripe*. Os resultados são úteis para determinar os gargalos e limites do *hardware* e *software* do ambiente de teste.

Foco nos Padrões de Acesso de Aplicações Paralelas

O foco neste caso é o comportamento do Lustre sob alguns padrões de acesso utilizados por aplicações paralelas reais. Para maiores detalhes, na Seção 4.1.2 foram apresentados alguns padrões geralmente utilizados. Nesta classe de testes os padrões de acesso *file-per-proc* e *single-shared-file*

serão avaliados. Considerando acessos contíguos ou não contíguos por parte dos processos ao arquivo, *single-shared-file* será subdividido em *segmented-access*, *simple-strided* e *random-strided*.

Metodologia Usada na Execução dos Experimentos

Na maior parte dos experimentos utilizou-se o *benchmark* IOR para simular a carga de trabalho de aplicações paralelas. Em alguns casos foi necessário implementar pequenos programas MPI para representar determinado padrão de acesso ou compreender melhor o funcionamento do Lustre. Além disso, foram elaborados *scripts* para automatizar as tarefas e reduzir a ocorrência de erros durante as avaliações. Para fins estatísticos, os resultados apresentados são valores médios obtidos pela execução dos experimentos por no mínimo 20 vezes. O desvio padrão foi calculado e será exibido em conjunto com os resultados somente nos casos onde o mesmo foi superior ou inferior a 10% do valor médio. Nos demais casos deve-se considerar que não houve grandes variações nos resultados obtidos.

Todas as máquinas do *cluster* foram utilizadas de modo exclusivo. Somente os processos básicos foram mantidos em execução, ou seja, após a instalação do sistema operacional e Lustre, somente os processos que por padrão são executados durante o *boot* foram mantidos. Da mesma forma, a rede de comunicação não estava sendo compartilhada por outros usuários ou processos. Devido às complexidades dos componentes envolvidos em um sistema de arquivos distribuído (disco, rede e componentes do sistema operacional), foi necessário preparar o ambiente a cada rodada de testes. Por exemplo, para eliminar o efeito do *cache* e garantir que o sistema de arquivos fosse exercitado, antes da execução das operações de escrita ou leitura, esvaziava-se o *buffer cache* das máquinas do *cluster*.

Para monitorar o comportamento do sistema operacional e Lustre utilizou-se a ferramenta COLLECTL [36]. Essa ferramenta permite monitorar o uso de CPU, disco, memória, rede e algumas informações fornecidas pelos serviços do Lustre. Dentre essas informações, pode-se citar a vazão de rede e disco de servidores de armazenamento e uso do *cache* e *read-ahead* nos clientes. A ferramenta COLLECTL auxiliou no entendimento da arquitetura do Lustre e análise dos dados obtidos nos experimentos. Entretanto, para não influenciar nos resultados a ferramenta não estava em execução durante as avaliações de desempenho.

Em alguns experimentos o Lustre é comparado ao NFS. O motivo é pelo simples fato que muitas instituições e grupos de pesquisa utilizam o NFS para manter os dados do *cluster*. Dessa forma, em alguns experimentos, a mesma carga de trabalho será aplicada em ambos os sistemas para fins comparativos.

A próxima seção apresenta os experimentos, como foram executados e resultados obtidos.

4.3 Avaliação com Foco no Ambiente de Teste

Para verificar os gargalos e limites do ambiente de teste foram propostos alguns experimentos com o objetivo de exercitar todos os componentes do sistema de arquivos. Alguns experimentos serviram

de base para os demais, ou seja, à medida que os resultados foram obtidos novos experimentos foram modelados. Os experimentos realizados e os resultados obtidos são apresentados a seguir.

4.3.1 Tamanho do Arquivo

Durante a execução de aplicações paralelas vários arquivos são criados, acessados e removidos do sistema de arquivos. O tamanho dos arquivos pode variar de acordo com o tempo de execução, tamanho do problema e nível de detalhamento dos resultados gerados. Todos esses aspectos influenciam no desempenho final da aplicação.

Para avaliar o impacto do tamanho do arquivo serão executadas operações de escrita e leitura em paralelo sobre o Lustre. A Tabela 4.1 lista os parâmetros e valores utilizados no experimento. Cada processo escreve/lê em arquivo exclusivo usando transferências de 64kB e biblioteca de acesso POSIX. Arquivos de 1MB a 256MB serão utilizados na avaliação. O Lustre está configurado para distribuir os arquivos entre todos os servidores de armazenamento em *stripes* de 1MB. Todos os testes serão executados 20 vezes e os resultados representam a vazão média alcançada.

Para fins comparativos o NFS será avaliado sob a mesma carga de trabalho.

Resultados da Escrita

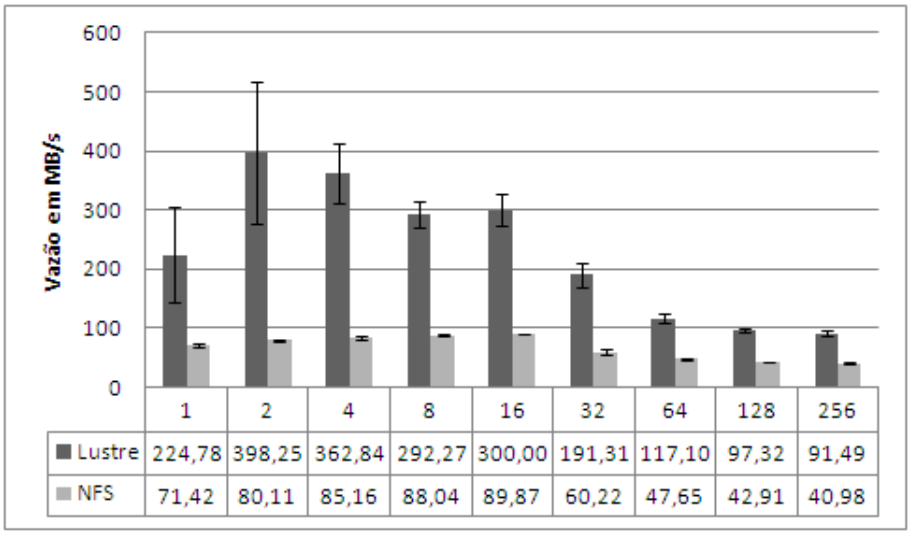
A Figura 4.6 apresenta a vazão média obtida na escrita de arquivos de tamanhos diversos no Lustre e NFS. As barras representam o desvio padrão. Com base nos resultados é possível destacar três pontos discutidos a seguir.

Primeiro, o Lustre apresentou desempenho superior ao NFS para todos os tamanhos de arquivos. Isso se deve a escrita de arquivos em paralelo. Mesmo sendo um *cluster* de pequena escala, com poucos processadores, a carga de trabalho simulada favorece o uso de um sistema de arquivos paralelos. O uso do NFS neste cenário limita o desempenho e escalabilidade da aplicação.

Segundo, o Lustre apresentou taxa de transferência elevada para arquivos pequenos. Isto está relacionado à natureza assíncrona das operações de escrita. Como o Linux utiliza parte da memória

Tabela 4.1: Descrição das configurações usadas no experimento.

Parâmetros	Valores
Benchmark	IOR
Número de Execuções	20
Número de Processos	8
Tamanho do Arquivo	1, 2, 4, 8, 16, 32, 64, 128, 256MB
Transferência (kB)	64kB
Padrão de Acesso	*file-per-proc*
Biblioteca de acesso	POSIX
Stripecount	3
Stripesize	1 MB



	1	2	4	8	16	32	64	128	256
Lustre	224,78	398,25	362,84	292,27	300,00	191,31	117,10	97,32	91,49
NFS	71,42	80,11	85,16	88,04	89,87	60,22	47,65	42,91	40,98

Figura 4.6: Vazão média e desvio padrão na escrita ao variar o tamanho do arquivo.

RAM como *buffer* de E/S e sabendo que as máquinas do *cluster* têm aproximadamente 128MB de espaço livre, para arquivos pequenos, menores que 128MB, a vazão atingiu taxas elevadas. Contudo, ao esgotar o espaço livre para *buffer*, os dados devem ser transferidos imediatamente para o sistema de arquivos, explicando a queda na vazão de dados para arquivos a partir de 128MB. O mesmo efeito ocorre com o NFS à medida que o tamanho do arquivo aumenta.

Reforçando a questão do *buffer cache*, no Lustre o parâmetro *max_cached_mb* define o tamanho da área de memória RAM livre da máquina que pode ser utilizada como *cache* cliente. Por padrão são destinados 75% da memória livre da máquina. Para visualizar o uso do *buffer cache* foram capturadas informações de uma das máquinas do experimento. As informações foram capturas através da ferramenta COLLECTL durante o teste de escrita de arquivo de 128MB. O gráfico da Figura 4.7 exibe o uso da *cache* cliente. Como se pode observar, a memória para *cache* rapidamente é utilizada durante toda a operação de escrita do arquivo, nesse caso aproximadamente 140MB.

Terceiro, para exercitar o sistema de arquivos é necessário utilizar arquivos com tamanho suficiente para ocupar toda a memória RAM livre do sistema. Caso contrário, o *throughput* aferido pode não representar o desempenho do sistema de arquivos, mas sim a velocidade de acesso a memória RAM. Em adição, o desvio padrão elevado para arquivos pequenos está relacionado ao estado atual do nó, onde o espaço livre em memória pode variar e afetar as medições. Observando a Figura 4.6 é possível verificar que a taxa de transferência sofre uma queda e estabiliza para arquivos maiores que 64MB. Da mesma forma, o desvio padrão também diminui para arquivos maiores que 64MB. Para evitar o efeito do *cache* cliente, nos próximos experimentos cada processo acessará no mínimo 256 MB de dados.

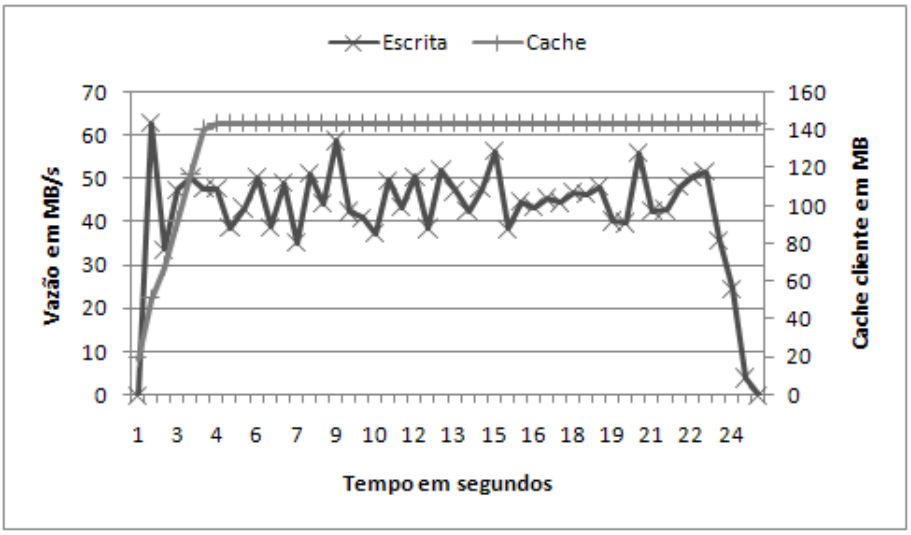


Figura 4.7: Uso do *cache* cliente.

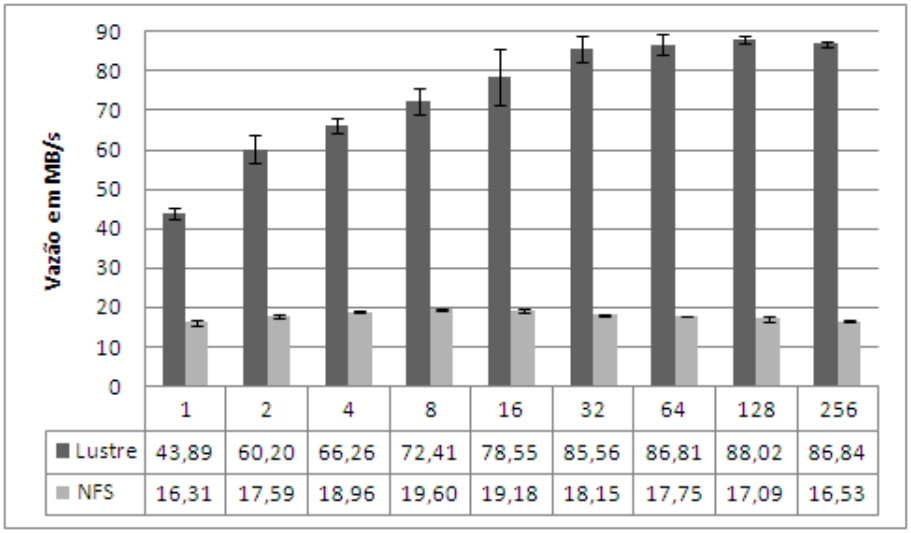


Figura 4.8: Vazão na leitura ao variar o tamanho do arquivo.

Resultados da Leitura

A Figura 4.8 exibe a vazão média e desvio padrão na leitura ao variar o tamanho do arquivo. Como se pode observar, o NFS apresentou desempenho inferior ao Lustre para todos os tamanhos de arquivos, a arquitetura centralizada limita a vazão de dados que ficou abaixo dos 20 MB/s. No Lustre a taxa de transferência agregada chegou próximo aos 90 MB/s. Nas medições, os maiores valores para desvio padrão obtidos foram de 7,25 e 0,70 para Lustre e NFS, respectivamente.

Para NFS e Lustre a vazão na leitura mostrou desempenho inferior em relação às operações de escrita. Enquanto que a escrita pode ser assíncrona, a leitura envolve o envio da requisição ao servidor remoto, acesso aos dados em disco e envio dos dados pela rede. Além disso, no Lustre foram obtidas

taxas de transferências maiores à medida que o tamanho do arquivo aumentava. Esse efeito está relacionado ao mecanismo de *read-ahead* implementado pelo Lustre. O *read-ahead* armazena blocos contíguos de dados no *cache* cliente.

Para visualizar o uso do *read-ahead* foram capturadas, com auxílio da ferramenta COLLECTL, informações como número de acertos na *cache* do Lustre, vazão em operações de leitura e rede de uma máquina do experimento. Devido a natureza sequencial das operações executadas, a taxa de acertos na *cache* ficou entre 98% e 100% (Figura 4.9).

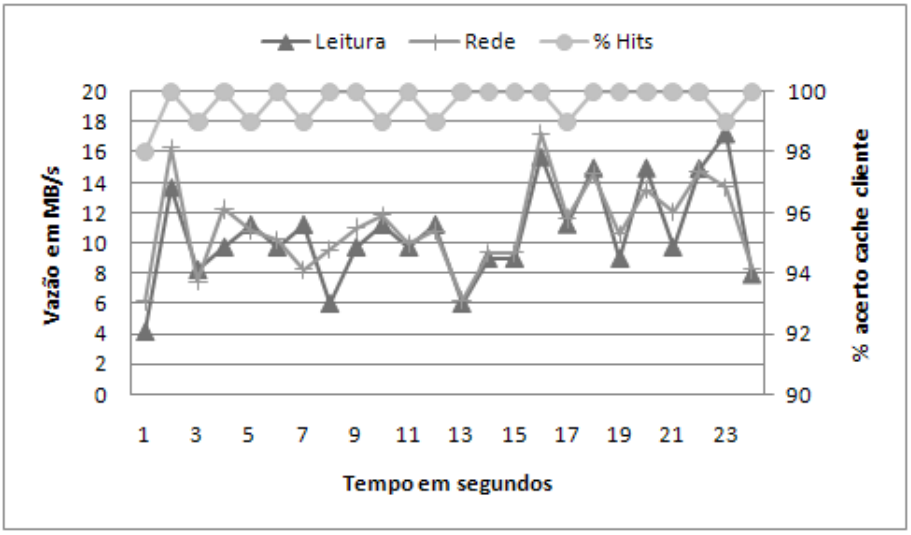


Figura 4.9: Efeito do *read-ahead*.

4.3.2 Tamanho da Unidade de Transferência

Este experimento investiga o impacto ao variar o tamanho da unidade de transferência. A unidade de transferência define o tamanho das rajadas de dados entre memória e arquivo.

Os mesmos parâmetros do experimento anterior serão utilizados, ou seja, o mesmo número de processos e configurações do Lustre. A Tabela 4.1 pode ser consultada. Entretanto, o tamanho do arquivo será fixado em 256MB. A definição do tamanho do arquivo tem por objetivo isolar o efeito do *cache* discutido anteriormente. Assim a vazão de dados obtida representará o desempenho do subsistema de E/S.

Resultados

A Figura 4.10 apresenta os resultados obtidos para escrita e leitura. Como se pode observar, variar o tamanho da unidade de transferência não impactou no desempenho do Lustre. O uso do *cache* cliente em operações de escrita, e na leitura o uso do *read-ahead*, suprimiu qualquer efeito relacionado ao tamanho da unidade de transferência.

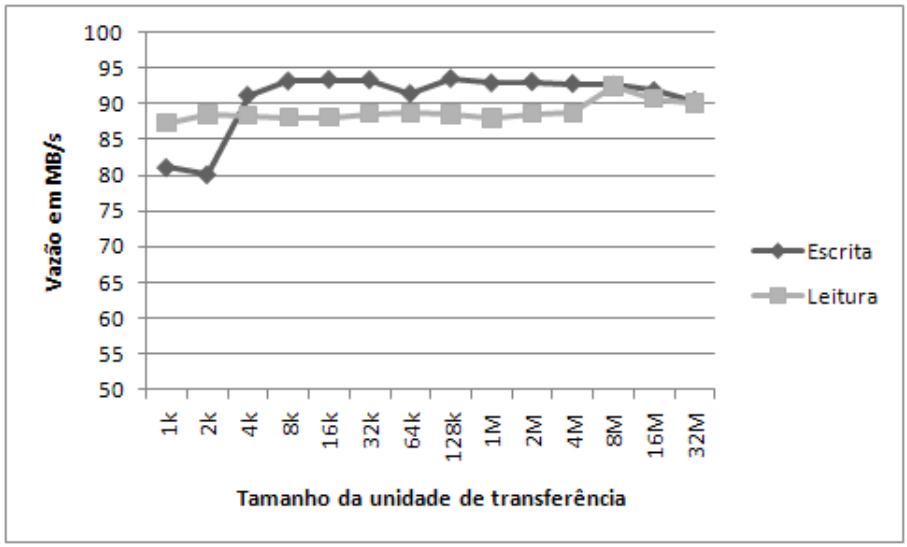


Figura 4.10: Variando o tamanho da unidade de transferência.

Somente em operações de escrita com transferências de 1kB e 2kB o desempenho sofreu queda, em torno de 12,7% inferior aos demais casos. A leitura não sofreu variações significativas. Com base nos resultados obtidos e visando reduzir o número de parâmetros avaliados, nos próximos experimentos o tamanho da unidade de transferência será fixado em 64kB.

4.3.3 Número de Clientes

Neste experimento o objetivo é verificar como o Lustre se comporta ao variar o número de processos clientes. Os testes serão realizados com número de processos entre 1 e 8. Cada processo estará alocado em uma máquina e executará operações de escrita e leitura sobre um arquivo de 256MB. Para tratar o efeito do *cache*, antes da execução das operações de escrita e leitura o *buffer cache* de todas as máquinas do *cluster* será esvaziado. Além disso, os processos utilizarão transferências de 64kB e biblioteca de acesso POSIX. A Tabela 4.1 pode ser consultada.

Resultados

Para analisar os resultados considere os dados da Tabela 4.2. Esses dados representam a vazão média dos componentes do ambiente de teste. A vazão da rede entre servidores e clientes foi aferida através do *benchmark* Netperf [6]. Para o disco rígido foi utilizado o comando *dd* do Linux. A coluna vazão agregada representa a vazão que teoricamente pode ser atingida considerando os três servidores de dados do Lustre. O cálculo não considera acesso concorrente aos discos e *overheads* inerentes na comunicação entre clientes e servidores.

A Figura 4.11 apresenta os resultados obtidos ao variar o número de processos participantes do experimento. Comparando o gráfico da Figura 4.11 com os dados da Tabela 4.2 é possível destacar os seguintes aspectos:

Tabela 4.2: Limites empíricos dos componentes do ambiente de teste.

	Vazão Um Servidor	Vazão Agregada Três Servidores
Rede Gigabit	76 MB/s	228 MB/s
Disco (Escrita)	61 MB/s	183 MB/s
Disco (Leitura)	48 MB/s	146 MB/s

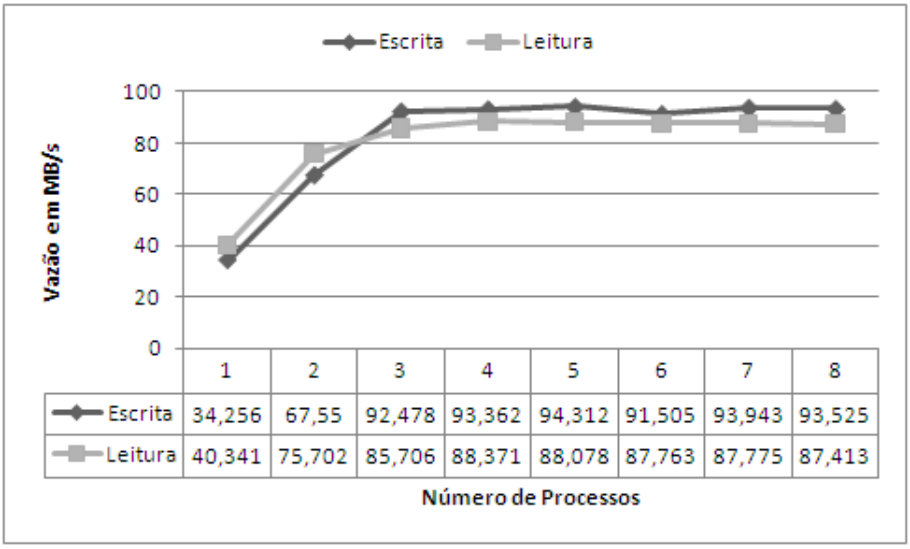


Figura 4.11: Variando o número de processos.

- A vazão alcançada no experimento é inferior a vazão agregada apresentada na Tabela 4.2. A diferença está relacionada ao *overhead* existente no acesso ao Lustre. O acesso paralelo aos discos e custo de comunicação influencia na vazão final fornecida. É importante ressaltar que a vazão agregada (Tabela 4.2) é resultado da soma da vazão de cada disco, sem considerar acesso concorrente e custo de comunicação.

- A rede também é um fator que afeta o desempenho do Lustre, porém, neste caso não está relacionada ao gargalo criado no acesso aos dados. De acordo com as informações da Tabela 4.2, a rede é aproximadamente 19% mais rápida que os discos.

- O número de nós clientes disponível para o teste não chegou a saturar os servidores de armazenamento. A vazão média estabilizou a partir de 3 processos e não sofreu queda para escrita e leitura.

- Outro fator que pode ser visualizado na Figura 4.11 é o baixo desempenho das máquinas clientes. No teste com um cliente, como o arquivo está distribuído entre todos os servidores de armazenamento, a vazão atingida exibe os limites do cliente e não do sistema de arquivos. O cliente não chega a sobrecarregar a rede *Gigabit Ethernet*. Esse comportamento era esperado devido ao *hardware* antigo dos clientes.

Analisando esses aspectos é possível identificar quais componentes influenciam no desempenho do Lustre. Conhecer as limitações dos componentes possibilita, por exemplo, calcular a relação entre servidores de armazenamento e clientes necessários para se ter um sistema de arquivos balanceado. Neste experimento os resultados sugerem duas modificações que podem melhorar o desempenho do Lustre. A primeira baseia-se na adição de novos servidores de armazenamento aumentando dessa forma a vazão de dados agregada. Baseado no fato da rede não ser o gargalo, a segunda é aumentar o desempenho do subsistema de E/S de cada servidor. Isso pode ser feito através do uso de matrizes de discos (ex. RAID 0, 5, 6, 10).

4.3.4 Número de Servidores de Armazenamento

De acordo com as considerações apontadas no experimento anterior, foram adicionados três servidores de armazenamento no sistema de arquivos Lustre. É importante ressaltar que as máquinas disponibilizadas são idênticas as já utilizadas no ambiente de teste como servidores de armazenamento. O objetivo é verificar se a adição de mais servidores aumenta a vazão do sistema. Para tanto, o experimento anterior foi executado novamente, porém, sobre um sistema de arquivos Lustre com maior número de servidores de armazenamento.

Resultados

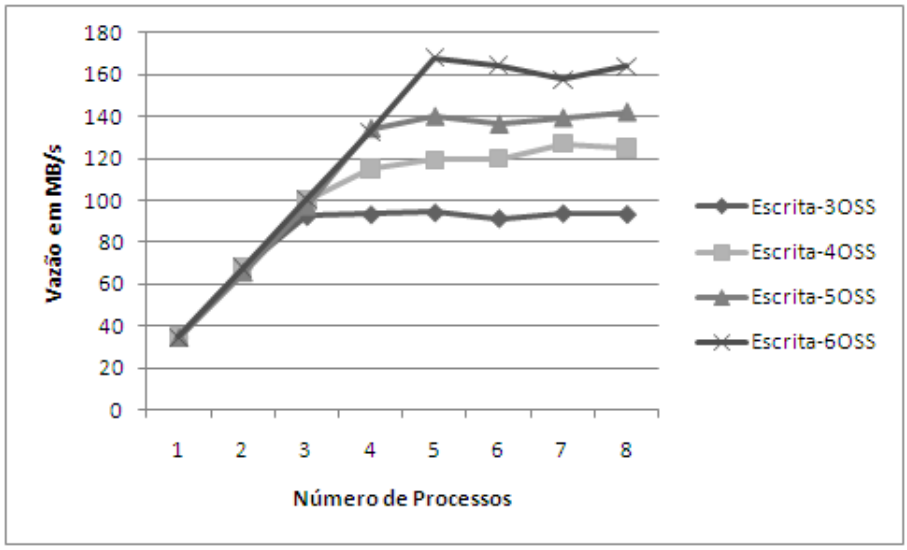


Figura 4.12: Escrita variando o número de servidores de armazenamento.

As Figuras 4.12 e 4.13 mostram a vazão de dados ao adicionar novos servidores de armazenamento (3, 4, 5 e 6 OSS). Como se pode observar, os resultados mostram que o desempenho pode ser melhorado ao adicionar novos servidores de armazenamento. Nos melhores casos observados, ao dobrar o número de OSS (de 3 para 6) se teve um ganho de aproximadamente 75% na escrita e 81% na leitura com 8 processos.

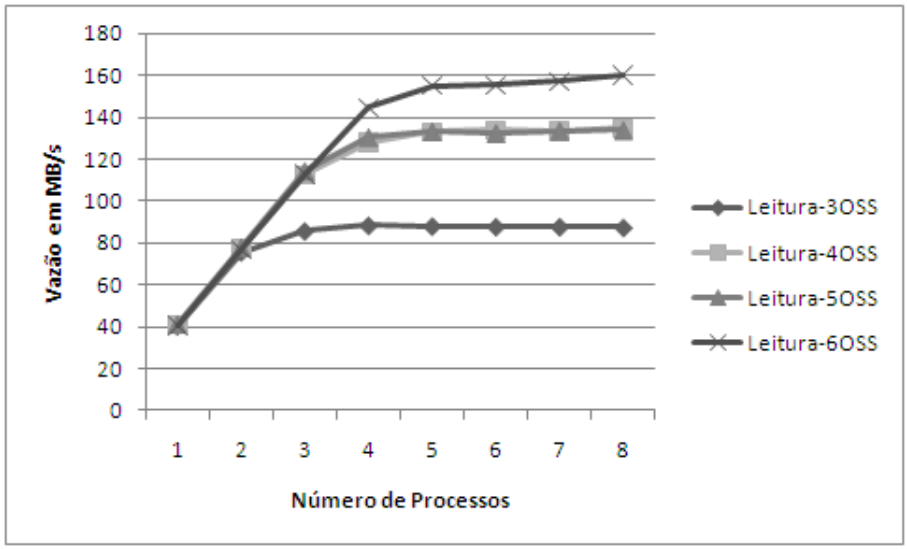


Figura 4.13: Leitura variando o número de servidores de armazenamento.

Da mesma forma como no experimento anterior, o número de nós clientes não chegou a saturar os servidores de armazenamento.

Um ponto interessante é a vazão média semelhante na leitura com 4 e 5 OSS. Em momento futuro é necessário investigar o motivo pelo qual não houve melhora ao adicionar o quinto OSS.

4.3.5 Distribuição de Carga entre Servidores de Armazenamento

O objetivo deste experimento é analisar o impacto da distribuição de carga entre os servidores de armazenamento do Lustre. Levando em consideração o ambiente de teste apresentado na Seção 3.3.1, há três formas de distribuir os dados:

- *stripecount=1*: armazenamento do arquivo em um servidor;
- *stripecount=2*: armazenamento do arquivo distribuído entre dois servidores;
- *stripecount=3*: armazenamento do arquivo distribuído entre três servidores.

Para exemplificar o experimento considere duas aplicações paralelas que escrevem dados no Lustre. A primeira com três processos e a segunda com quatro processos. Ambas as aplicações estão prontas para escrever dados no sistema de arquivos, cada processo escreve em um arquivo exclusivo, não existindo acesso concorrente. A Figura 4.14 ilustra as formas possíveis de distribuição de dados entre processos e servidores de armazenamento.

Quando o número de processos é múltiplo do número de servidores de armazenamento a distribuição dos dados pode ser facilmente balanceada, Figura 4.14(a-c). Entretanto, quando não existe essa relação de multiplicidade a distribuição pode não ser equilibrada, Figuras 4.14(d) e 4.14(e).

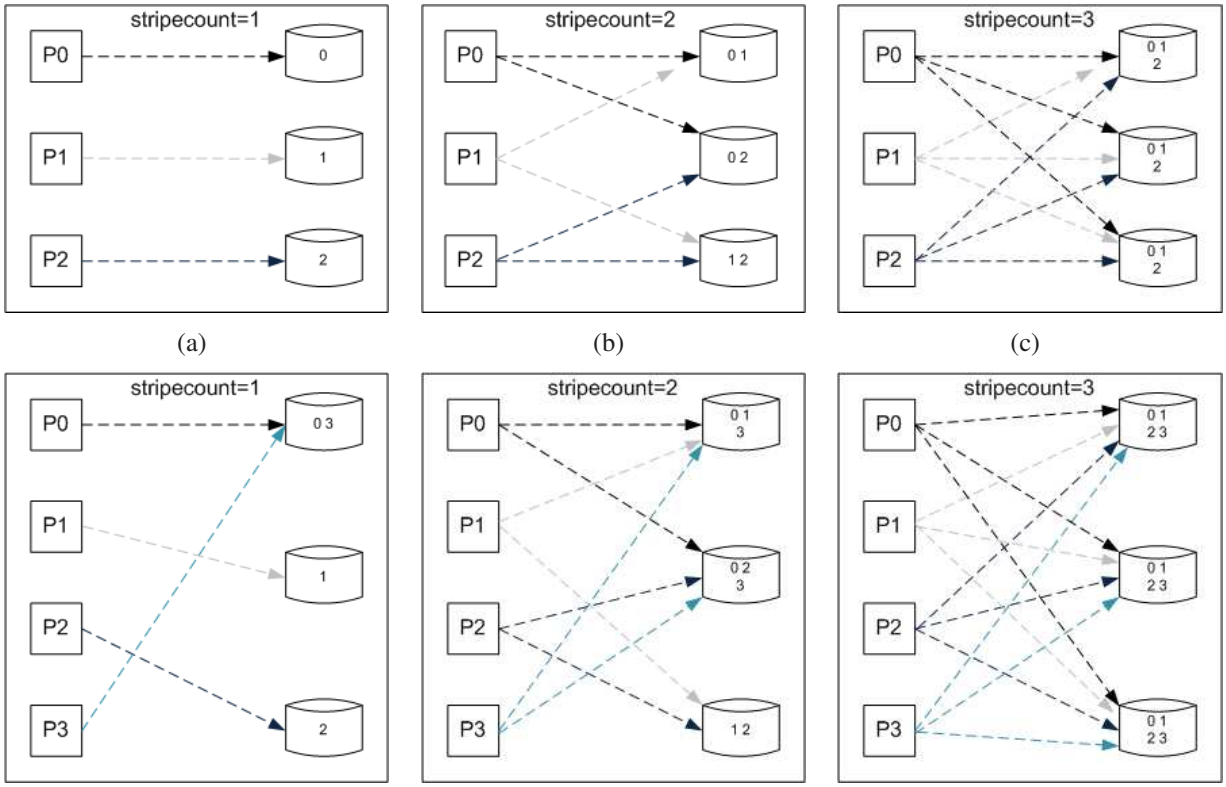


Figura 4.14: Formas de distribuição de arquivos adotadas no experimento.

Para mostrar o impacto da distribuição de carga serão utilizados números de processos múltiplos (*P=3* e *P=6*) e não múltiplos (*P=4* e *P=8*) do número de servidores de armazenamento. O Lustre possibilita, no momento da criação de diretórios ou arquivos, a definição de como os dados serão distribuídos entre os servidores de armazenamento. Neste caso foram criados três diretórios configurados para representar as formas de distribuição descritas anteriormente. O *benchmark* IOR será utilizado para simular as operações de escrita. A Tabela 4.3 apresenta um resumo dos parâmetros utilizados neste experimento.

Resultados

Três formas de distribuir os dados foram avaliadas. Neste experimento foram criados diretórios configurados de forma que os arquivos fossem armazenados em um, dois ou três servidores. Outro fator importante é o número de processos envolvidos no experimento (*P=3*, *P=4*, *P=6* e *P=8*). A definição do número de processos permite avaliar dois casos: (*i*) distribuição equilibrada (número de processos múltiplo do número de servidores de armazenamento) e (*ii*) distribuição não equilibrada (números não múltiplos) dos dados entre os servidores de armazenamento.

A Tabela 4.4 lista os testes realizados no experimento. Nesta tabela estão listados, a identificação do teste, parâmetros *stripecount* e *stripesize* do Lustre, número de processos, tamanho do arquivo

Tabela 4.3: Descrição das configurações usadas no experimento.

Parâmetros	Valores
Benchmark	IOR
Número de Execuções	20
Operações	escrita
Número de Processos	3, 4, 6 e 8
Tamanho do Arquivo	256MB por processo
Transferência (kB)	64kB
Padrão de Acesso	*file-per-proc*
Biblioteca de acesso	POSIX
Stripecount	1, 2 e 3
Stripesize	1, 2 e 4 MB

Tabela 4.4: Lista dos testes realizados no experimento.

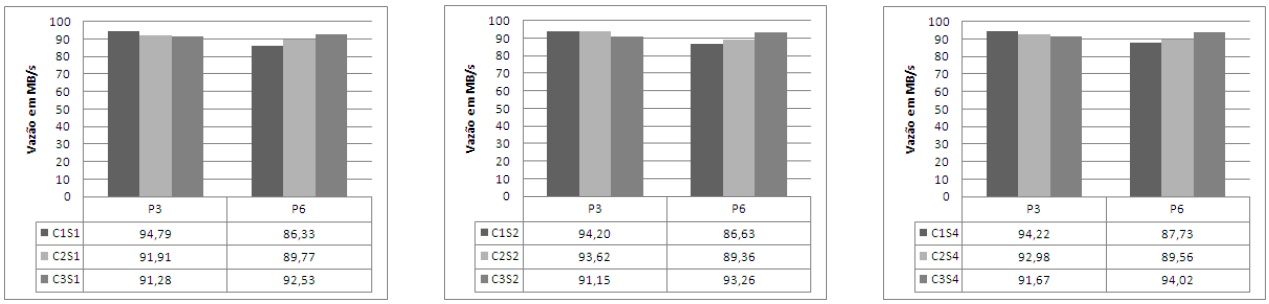
	Stripecount	*Stripesize* (MB)	Núm. Processos	Arquivo	Bloco
C1S1	1	1	3, 4, 6 e 8	256MB/proc	64kB
C1S2	1	2	3, 4, 6 e 8	256MB/proc	64kB
C1S4	1	4	3, 4, 6 e 8	256MB/proc	64kB
C2S1	2	1	3, 4, 6 e 8	256MB/proc	64kB
C2S2	2	2	3, 4, 6 e 8	256MB/proc	64kB
C2S4	2	4	3, 4, 6 e 8	256MB/proc	64kB
C3S1	3	1	3, 4, 6 e 8	256MB/proc	64kB
C3S2	3	2	3, 4, 6 e 8	256MB/proc	64kB
C3S4	3	4	3, 4, 6 e 8	256MB/proc	64kB

por processo e tamanho do bloco de transferência. Os parâmetros *stripesize* e *stripecount* definem a forma como os arquivos serão fracionados e distribuídos entre os servidores de armazenamento. Para exemplificar, a entrada C2S4 significa *stripecount=2* e *stripesize=4*. As Figuras 4.15 e 4.16 apresentam os resultados obtidos no experimento.

Na Figura 4.15 os gráficos mostram a vazão de dados obtida com *P=3* e *P=6*. Como o número de processos é múltiplo do número de servidores, a carga de trabalho pode ser distribuída de forma equilibrada. Pode-se observar que todas as formas de distribuição de dados (*C=1*, *C=2* e *C=3*) apresentam desempenho semelhante.

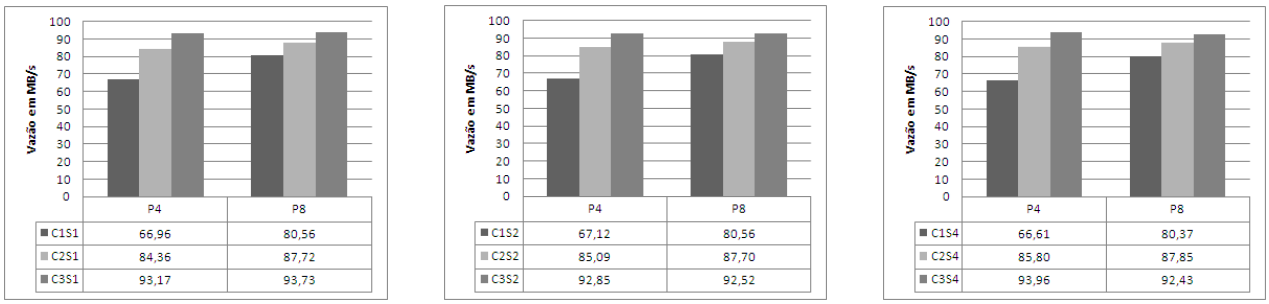
Para *P=4* e *P=8*, onde o número de processos não é múltiplo do número de servidores de armazenamento, a carga pode não ser distribuída de forma equilibrada nas situações *C=1* e *C=2*. Na Figura 4.16 os gráficos apresentam a vazão obtida. Para *C=1* cada arquivo é armazenado inteiramente em um servidor, e como pode-se observar a vazão atingida é inferior ao demais casos. Para *C=2* a vazão apresenta melhora, entretanto a maior vazão obtida é quando os arquivos são distribuídos entre os três servidores (*C=3*).

Para explicar o motivo da vazão inferior para *C=1*, considere o cenário exibido na Figura 4.14(d). Como há quatro processos e apenas três servidores de dados a distribuição de carga resultante não é equilibrada, acarretando em desempenho inferior. A mesma situação ocorre para *C=2*. Para *C=3* a



(a) (b) (c)

Figura 4.15: Vazão alcançada com *P=3* e *P=6*.



(a) (b) (c)

Figura 4.16: Vazão alcançada com *P=4* e *P=8*.

carga é distribuída de forma equilibrada, melhorando o desempenho.

Os resultados obtidos neste experimento indicam que casos onde os números de processos e servidores não são múltiplos, a melhor abordagem é distribuir os dados entre todos os servidores de armazenamento. A verificação da distribuição dos dados entre os servidores de armazenamento foi realizada com auxílio da ferramenta COLLECTL e ferramentas do próprio Lustre.

4.3.6 Acesso Não Alinhado ao Tamanho do *Stripe*

O Lustre fornece semântica de acesso POSIX, garantindo consistência no acesso aos arquivos. Através do sistema de travas distribuídas cada servidor de armazenamento é responsável por gerenciar as travas dos *stripes* de arquivos que mantém. Esse mecanismo permite ao Lustre boa escalabilidade, permitindo acesso em paralelo ao arquivo. Neste experimento será investigado o impacto ocasionado quando o tamanho do bloco de dados, que um processo lê ou escreve, não está alinhado aos limites dos *stripes* do arquivo.

Para melhorar a compreensão do experimento considere uma aplicação paralela que lê dados para a fase de processamento. A aplicação consiste em três processos lendo um arquivo compartilhado de 256MB. O arquivo é dividido em diversos segmentos (*S1,S2...Sn*), e cada segmento tem três blocos de dados. Cada processo lê um bloco de dados contíguo. A leitura do próximo segmento somente

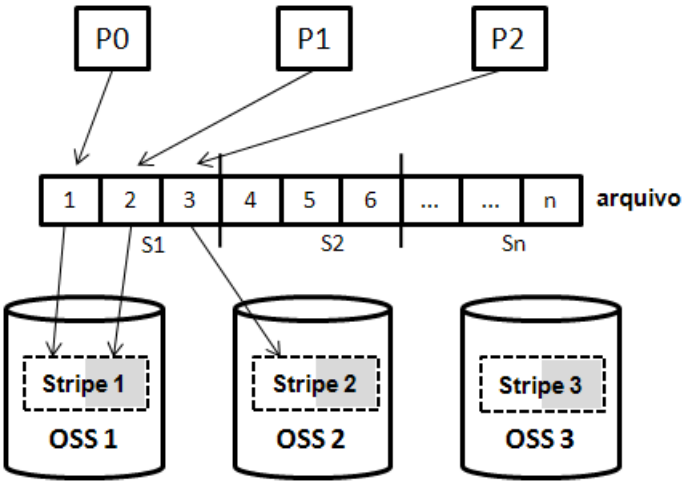


Figura 4.17: Acesso não alinhado ao tamanho do *stripe*.

Tabela 4.5: Descrição das configurações usadas no experimento.

Parâmetros	Valores
Benchmark	IOR
Número de Execuções	20
Operações	leitura
Número de Processos	3 e 6
Tamanho do Arquivo	256MB
Transferência (kB)	64kB
Tamanho do Bloco (kB)	512, 1024, 1536 e 2048
Padrão de Acesso	*simple-strided*
Biblioteca de acesso	POSIX
Stripecount	3
Stripesize	1MB

ocorre após todos os processos terminarem a leitura do segmento corrente.

A Figura 4.17 mostra um caso onde o tamanho do bloco de dados não está alinhado ao tamanho do *stripe* do arquivo. Neste caso, dois processos fazem acesso a regiões não sobrepostas do mesmo *stripe*, processos P0 e P1 acessam de forma concorrente o *stripe 1*. Durante a leitura dos demais segmentos do arquivo esse comportamento se repete.

Para investigar o impacto gerado pelo acesso não alinhado serão executados dois testes com três e seis processos lendo um arquivo compartilhado de aproximadamente 256MB. O tamanho do bloco de leitura será definido de forma a representar duas situações: *(i)* acesso alinhado e *(ii)* acesso não alinhado com o tamanho do *stripe*. Considerando a configuração do Lustre apresentada na seção 3.3.1, o arquivo será distribuído entre todos os servidores de armazenamento (*stripecount=3*), usando *stripe* igual a 1MB (*stripesize=1MB*). A Tabela 4.5 exibe os valores dos parâmetros de configuração utilizados neste experimento.

Resultados

Testes com quatro tamanhos de blocos de dados distintos foram executados a fim de representar acesso alinhado e não alinhado:

- bloco de 512kB: acesso não alinhado, dois processos acessam o mesmo *stripe*.
- bloco de 1024kB: acesso alinhado, um *stripe* por processo.
- bloco de 1536kB: acesso não alinhado, o processo acessa um *stripe* inteiro e metade do próximo.
- bloco de 2048kB: acesso alinhado, o processo acessa dois *stripe* consecutivos.

A Tabela 4.6 lista os testes realizados neste experimento. Nessa tabela estão listadas as configurações do Lustre (*stripecount* e *stripesize*), número de processos envolvidos, número de segmentos e tamanho do bloco de dados por processo. O tamanho do bloco determina o número de *bytes* que cada processo lê. Por exemplo, com *P=3* e bloco igual a 512kB, cada processo lê um bloco contíguo de 512kB, representando um segmento do arquivo. Nesse exemplo o arquivo é dividido em 170 segmentos de 1536kB.

Tabela 4.6: Lista dos testes realizados no experimento.

	Stripecount	*Stripesize* (MB)	Núm. Processos	Segmentos	Bloco por Proc
1	3	1	3	170	512kB
2	3	1	3	85	1024kB
3	3	1	3	56	1536kB
4	3	1	3	42	2048kB
5	3	1	6	85	512kB
6	3	1	6	42	1024kB
7	3	1	6	28	1536kB
8	3	1	6	21	2048kB

Na Figura 4.18 são apresentados os resultados obtidos. O acesso não alinhado (blocos de 512kB e 1536kB) impactou negativamente no desempenho do Lustre. Para *P=3*, os processos P0 e P1 fazem acesso ao *stripe 1*, enquanto que P2 faz acesso ao *stripe 2*. Neste caso, o mecanismo de travas distribuídas do Lustre serializa o acesso ao *stripe 1* limitando o desempenho da aplicação. O mesmo comportamento é observado para *P=6*.

No acesso alinhado (blocos de 1024kB e 2048kB) cada processo faz acesso a um ou dois *stripes* distintos. Os processos solicitam as travas sobre os *stripes* e executam em paralelo a leitura do arquivo.

Como se pode observar, o acesso alinhado resulta em melhor desempenho. Nos casos onde o tamanho do bloco não é múltiplo do tamanho do *stripe* a vazão de dados atingiu no máximo 2,5 MB/s para *P=3* e 6,24 MB/s para *P=6*. O fato de não alinhar o bloco de dados ao tamanho do *stripe* reduziu a vazão abaixo de 1 MB/s no pior caso. Este experimento destaca a importância de projetar aplicações paralelas de acordo com as características fornecidas pelo sistema de arquivos.

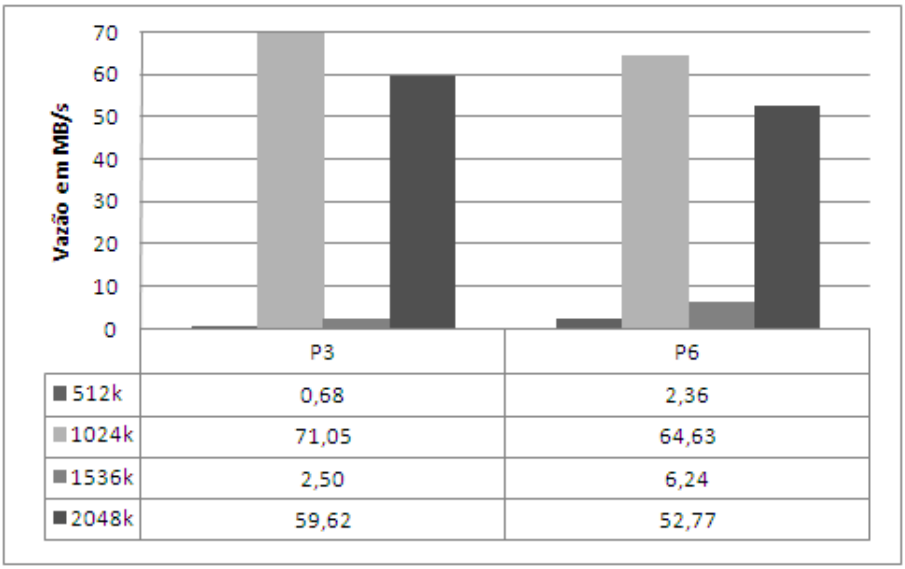


Figura 4.18: Acesso alinhado e não alinhado ao tamanho do *stripe*.

4.3.7 Resumo dos Fatores de Impacto

Esta seção apresenta um resumo sobre os fatores avaliados nos experimentos da classe ambiente de teste. Os pontos observados são apresentados a seguir:

- Tamanho do arquivo: os resultados mostraram que a escolha do tamanho do arquivo é importante para avaliar de forma efetiva o Lustre. Os mecanismos de *cache* cliente e *read-ahead* são utilizados de maneira agressiva, sendo necessário selecionar o tamanho do arquivo com cuidado. Caso contrário, os resultados obtidos podem não representar o desempenho do sistema de arquivos, mas sim, a velocidade de acesso a memória dos nós, mascarando as limitações do ambiente de teste. Além disso, os resultados também mostraram que as operações de leitura tem desempenho inferior as operações de escrita.

- Tamanho da unidade de transferência: variar o tamanho da unidade de transferência não afetou de forma significativa o desempenho do Lustre. Os mecanismos de *cache* cliente e *read-ahead* suprimiram qualquer efeito relacionado ao tamanho das transferências ao sistema de arquivos.

- Número de clientes e servidores de armazenamento: esses experimentos permitiram verificar o quão escalável o Lustre pode ser e quais são os limites do ambiente de teste. A partir da análise dos resultados, foi possível localizar os gargalos e sugerir modificações para melhorar o desempenho do subsistema de E/S do ambiente de teste.

- Distribuição de carga entre servidores de armazenamento: os resultados deste experimento mostraram que o ideal é manter o número de nós clientes múltiplo dos nós de armazenamento. Dessa maneira, os dados podem ser distribuídos de forma balanceada sobre os servidores de armazenamento, melhorando o desempenho.

- Acesso não alinhado ao tamanho do *stripe*: O acesso não alinhado entre o tamanho do bloco que um processo escreve/lê em relação ao tamanho do *stripe* no Lustre, pode serializar o acesso ao arquivo. Os resultados mostraram que o acesso alinhado é fundamental para se obter bom desempenho.

A execução desses experimentos foi importante na compreensão do funcionamento do Lustre. As análises e resultados obtidos auxiliaram na definição de alguns fatores para a próxima classe de testes, voltada aos padrões de acesso.

4.4 Avaliação com Foco nos Padrões de Acesso

Nesta seção serão avaliados alguns padrões de acesso típicos de aplicações paralelas. O foco estará voltado aos padrões de acesso, ou seja, a forma como os processos realizam as operações de E/S. Na seção anterior, a avaliação tinha como ponto de vista o ambiente de teste, permitindo configurar parâmetros do sistema de arquivos e adicionar novos servidores de armazenamento. Em muitos ambientes de produção, o usuário não tem permissão para alterar as configurações do sistema, nesses casos a aplicação é quem deve ser adaptada. Por isso a importância em compreender a arquitetura do sistema de arquivos e a influência dos padrões de acesso. Assim, os desenvolvedores podem projetar aplicações que se beneficiem das características do sistema de arquivos em questão.

Os experimentos a seguir mostram como o Lustre se comporta sob os padrões de acesso discutidos na Seção 4.1.2. Todos os testes são baseados em uma aplicação paralela com 8 processos executando operações de escrita e leitura. Os padrões de acesso que serão avaliados são: *file-per-proc*, *segmented-access*, *simple-strided* e *random-strided*.

4.4.1 *File-per-proc*

O padrão *file-per-proc* representa aplicações onde cada processo acessa um arquivo exclusivo. Tarefas como *checkpoint* e escrita de resultados são exemplos comuns do uso desse padrão.

Para simular a carga de trabalho do experimento utilizou-se o *benchmark* IOR. O *benchmark* foi configurado de forma que cada processo escreva ou leia um arquivo exclusivo de 256MB. O resultado final é o acesso a 2GB de dados. A Tabela 4.7 resume os parâmetros e valores utilizados no experimento.

Resultados

A Figura 4.19 apresenta a vazão de dados obtida. Como se pode observar, o desempenho do Lustre é superior ao NFS (em torno de 2 vezes para escrita e 5 para leitura). Neste padrão os processos acessam grandes blocos de dados contíguos. Assim, os mecanismos do Lustre como *cache* cliente e *read-ahead* são utilizado com sucesso. Este é um tipo de padrão que escala bem no Lustre, em ambientes de maior escala o desempenho pode ser melhorado adicionando mais servidores

Tabela 4.7: Descrição das configurações usadas no experimento.

Parâmetros	Valores
Padrão	*file-per-proc*
Número de Execuções	20
Operações	escrita e leitura
Número de Processos	8
Número de Arquivos	8
Tamanho do Arquivo	256MB
Transferência (kB)	64kB
Tamanho do Bloco (kB)	256 MB
Biblioteca de acesso	POSIX
Stripecount	3
Stripesize	1MB

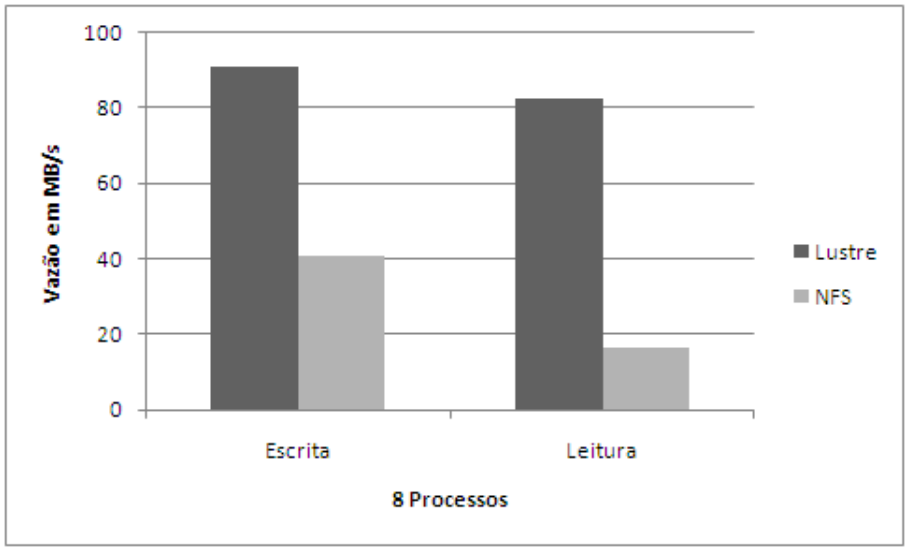


Figura 4.19: Padrão de acesso *file-per-proc*.

de armazenamento. Contudo, dependendo do número de arquivos criados/acessados o servidor de metadados do Lustre pode tornar-se o gargalo, limitando o desempenho do sistema.

4.4.2 *Segmented-access*

O padrão de acesso *segmented-access* representa aplicações onde todos os processos acessam um arquivo compartilhado. Para simular a carga de trabalho do experimento utilizou-se o *benchmark* IOR. Neste experimento todos os processos irão escrever/ler um bloco de dados de 256MB no arquivo compartilhado. O resultado final é o acesso a 2GB de dados. A Tabela 4.8 resume os parâmetros e valores utilizados no experimento.

Tabela 4.8: Descrição das configurações usadas no experimento.

Parâmetros	Valores
Padrão	*segmented-access*
Número de Execuções	20
Operações	escrita e leitura
Número de Processos	8
Número de Arquivos	1
Tamanho do Arquivo	2GB
Transferência (kB)	64kB
Tamanho do Bloco (kB)	256 MB
Biblioteca de acesso	MPI-IO
Stripecount	3
Stripesize	1MB

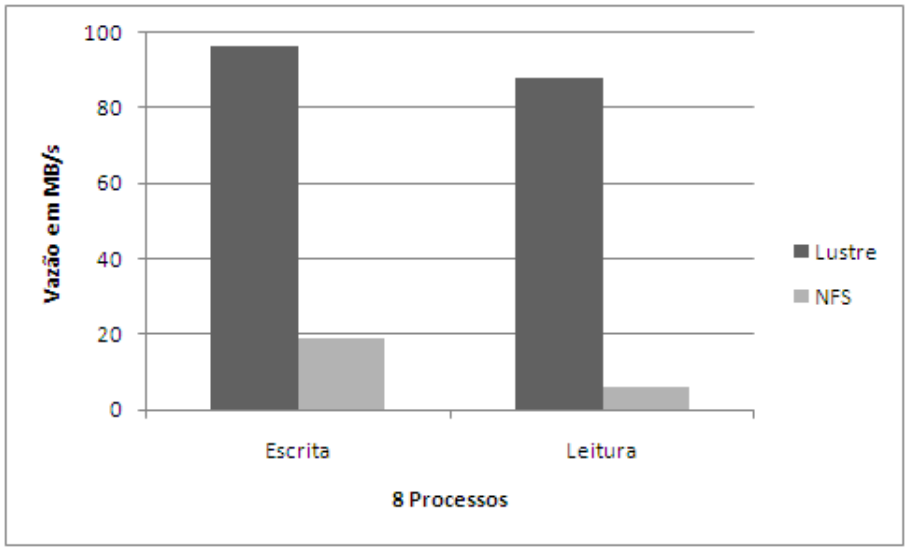


Figura 4.20: Padrão de acesso *segmented-access*.

Resultados

A Figura 4.20 apresenta a vazão de dados obtida. Neste padrão o Lustre apresentou desempenho superior ao NFS em aproximadamente 5 vezes na escrita e 15 vezes na leitura. Os padrões de acesso *file-per-proc* e *segmented-access* apresentaram desempenho semelhante. O motivo é que em ambos os padrões os processos acessam grandes blocos de dados contíguos.

Semelhante a *file-per-proc*, o padrão *segmented-access* escala bem no Lustre com a vantagem de não criar vários arquivos, que em ambientes de grande escala podem sobrecarregar o servidor de metadados do Lustre.

4.4.3 *Simple-strided*

O padrão *simple-strided* pode ser empregado em aplicações paralelas de diversas áreas. Podem-se citar aplicações que simulam fenômenos naturais onde a cada passo algumas variáveis (temperatura, pressão, umidade, por exemplo) são acessadas. O padrão *simple-strided* difere de *segmented-access* pelo fato de cada processo acessar vários blocos de dados dispersos pelo arquivo de forma ordenada. Em *simple-strided*, todos os processos fazem acesso a blocos de dados de mesmo tamanho. Dependendo da aplicação, podem-se ter blocos grandes ou pequenos. Neste experimento será avaliada a influência que o tamanho do bloco pode gerar no desempenho de aplicações que usam o padrão *simple-strided*.

A carga de trabalho será simulada através do *benchmark* IOR. Cada processo acessará blocos de dados de tamanho fixo obedecendo a um determinado deslocamento dentro do arquivo. De qualquer maneira, cada processo escreve ou lê 256MB. O tamanho do bloco define o número e tamanho dos segmentos do arquivo. Blocos pequenos tornam o acesso esparso, por exemplo, para bloco igual a 1MB, todos os processos acessarão 256 blocos ordenados de forma circular. Da mesma forma, blocos grandes tornam o acesso menos esparso. Por exemplo, para bloco igual a 128MB, todos os processos acessarão apenas dois blocos de dados. Para fins comparativos o Lustre e NFS serão submetidos à mesma carga de trabalho. A Tabela 4.9 resume os parâmetros e valores utilizados.

Tabela 4.9: Descrição das configurações usadas no experimento.

Parâmetros	Valores
Padrão	*simple-strided*
Número de Execuções	20
Operações	escrita e leitura
Número de Processos	8
Tamanho do Arquivo	2GB
Transferência (kB)	64kB
Tamanho do Bloco (kB)	64 kB a 128 MB
Biblioteca de acesso	MPI-IO
Stripecount	3
Stripesize	1MB

Resultados

A Figura 4.21 mostra a vazão alcançada ao variar o tamanho do bloco de dados que cada processo acessa. É interessante observar que o NFS apresentou melhor desempenho que o Lustre para blocos pequenos de dados ($< 1MB$) em operações de leitura. Para blocos de tamanho pequeno a vazão na leitura com o Lustre não ultrapassou 4 MB/s, cerca de 50% inferior a vazão obtida com o NFS. Na escrita o NFS apresentou desempenho superior ao Lustre somente para blocos de 64kB. Para blocos maiores ($> 512kB$) o Lustre obteve desempenho superior ao NFS para escrita e leitura. Os

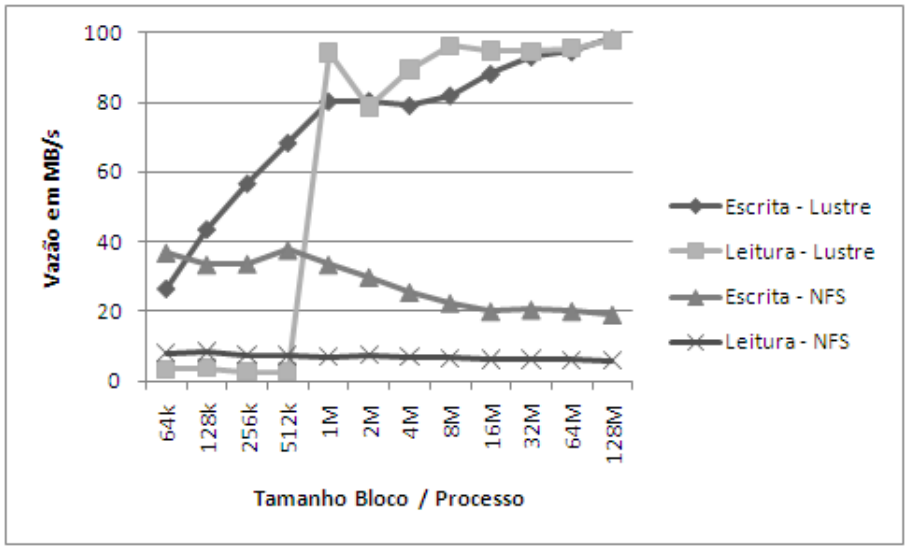


Figura 4.21: Acesso a regiões não contíguas do arquivo (*simple-strided*).

mecanismos do Lustre, discutidos anteriormente, como *cache* cliente e *read-ahead*, se adaptam melhor com aplicações onde cada processo acessa grandes blocos de dados contíguos. Para aplicações onde o acesso não é contíguo, ou seja, os processos acessam regiões esparsas dentro do arquivo, o desempenho geral sofre queda considerável.

Com base nos resultados obtidos é possível concluir que, para aplicações que desempenham o padrão de acesso *simple-strided*, é interessante utilizar estratégias que transformem acessos não contíguos em acessos contíguos ao arquivo. Uma estratégia geralmente utilizada consiste no *two-phase* I/O [41] e [42], onde inicialmente os processos redistribuem os dados entre si para, posteriormente, escrever os dados em grandes blocos contíguos.

4.4.4 *Random-strided*

Nos padrões de acesso discutidos anteriormente, o tamanho do bloco de dados que cada processo acessa é fixo. Porém, algumas aplicações podem ter acessos a blocos de tamanhos diversos a cada fase de execução. O padrão *random-strided* representa essa classe de aplicações.

O objetivo deste experimento é investigar o comportamento do Lustre simulando uma aplicação com padrão *random-strided*. O *benchmark* IOR não possui tal padrão de acesso. Para tanto, tornou-se necessário desenvolver uma aplicação paralela MPI onde o tamanho do bloco de dados fosse aleatório. De forma sucinta, a aplicação executa os seguintes passos: *(i)* processo 0 (zero) gera um vetor com o tamanho do bloco que cada processo deve escrever. *(ii)* Processo 0 (zero) distribui vetor para os demais processos. *(iii)* Todos os processos escrevem dados do tipo inteiro em um arquivo compartilhado obedecendo o tamanho do bloco. O tamanho da unidade de transferência é de 64kB. A ordem de escrita é circular (processo 0, bloco 0, processo 1, bloco 1, e assim por diante). *(iv)* Se o número de *bytes* escritos alcançou o tamanho do arquivo, passado por parâmetro, a aplicação é finalizada e a

Tabela 4.10: Descrição das configurações usadas no experimento.

Parâmetros	Valores
Padrão	*random-strided*
Número de Execuções	20
Operação	escrita e leitura
Número de Processos	8
Tamanho do Arquivo	>= 2GB
Transferência (kB)	64kB
Blocos Pequenos (kB)	64, 128, 256, 512
Blocos Grandes (MB)	1, 2, 4, 16, 32, 64
Biblioteca de acesso	MPI-IO
Stripecount	3
Stripesize	64kB, 1MB e 64MB

vazão em MB/s é exibida. Caso contrário, o fluxo de execução volta ao passo *(i)*. Dessa forma são gerados vários segmentos de tamanhos diversos até que o tamanho final do arquivo seja alcançado.

Para facilitar a implementação, todos os tamanhos de blocos sorteados são múltiplos de 2. A aplicação pára a execução quando o tamanho final do arquivo é maior ou igual a 2GB. Os testes serão divididos em duas classes: *(i)* blocos de tamanho grande e *(ii)* blocos de tamanho pequeno. O número de processos da aplicação é igual a 8. Para fins comparativos o Lustre e NFS serão submetidos à mesma carga de trabalho. No Lustre, três tamanhos de *stripes* serão avaliados (64kB, 1MB e 64MB). A Tabela 4.10 resume os parâmetros e valores utilizados.

Resultados

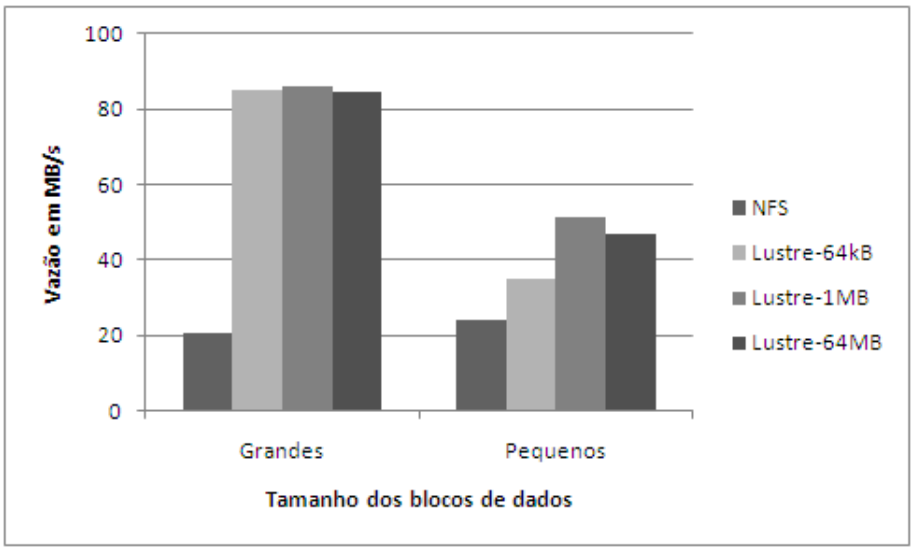


Figura 4.22: Vazão na escrita com padrão de acesso *random-strided*.

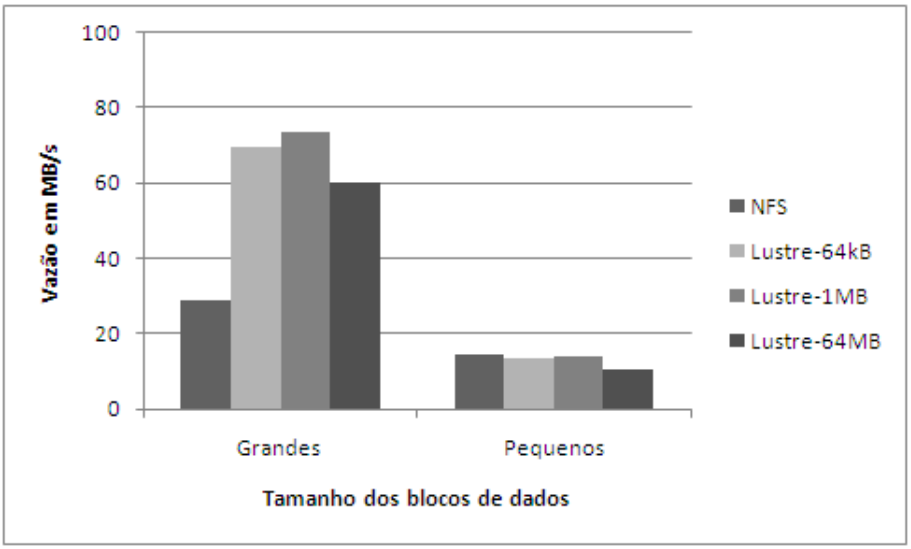


Figura 4.23: Vazão na leitura com padrão de acesso *random-strided*.

As Figuras 4.22 e 4.23 mostram a vazão de dados obtida para escrita e leitura, respectivamente. Como se pode observar, blocos de tamanho grande têm melhor desempenho. Em alguns casos o uso de blocos grandes apresentou o dobro de desempenho, os motivos estão relacionados ao uso agressivo do *cache* cliente. Além disso, como discutido anteriormente, devido ao controle de travas distribuídas utilizado no Lustre, a falta do alinhamento entre tamanho do bloco e *stripe* pode serializar o acesso reduzindo o desempenho.

De forma geral, os resultados obtidos permitem destacar alguns pontos:

- A leitura sofreu maior queda no desempenho com blocos pequenos do que a escrita. Nesse caso o Lustre apresentou desempenho semelhante ao NFS.
- Blocos de tamanhos aleatórios não permitem a distribuição homogênea da carga de trabalho entre os processos. Alguns processos ficaram responsáveis por escrever ou ler porções maiores de dados do que outros.
- O experimento também avaliou o impacto ao variar o valor do parâmetro *stripesize* do Lustre. *Stripes* pequenos (64kB) e grandes (1MB e 64MB) foram utilizados. A relação *stripes* e blocos pequenos gera *overhead* maior no envio de dados pela rede, acarretando em queda no desempenho. O manual do Lustre recomenda o uso de *stripes* entre 1MB e 4MB.

4.4.5 Comparando os padrões de acesso

Esta seção apresenta uma comparação em termos de desempenho entre os padrões de acesso avaliados sob o Lustre. As seções anteriores têm maiores informações sobre como os valores foram obtidos. A Tabela 4.11 apresenta a vazão (MB/s) da escrita e leitura para blocos de dados pequenos

($< 1MB$) e grandes($> 512kB$). Somente os melhores casos obtidos com cada padrão de acesso estão relacionados. Para os padrões *file-per-proc* e *segmented-access* não há resultados para blocos pequenos. Nesses dois casos cada processo escreve ou lê um bloco de dados de 256MB, considerado bloco de tamanho grande neste trabalho.

Tabela 4.11: Resumo das melhores taxas de transferência (MB/s) obtidas com o Lustre.

	Escrita		Leitura	
Padrões	Pequeno	Grande	Pequeno	Grande
file-per-proc	x	96,99	x	83,81
segmented-access	x	96,16	x	87,64
simple-strided	68,34	98,31	3,67	97,78
random-strided	51,26	85,83	14,16	73,46

Para todos os padrões de acesso avaliados sob o Lustre os melhores casos foram obtidos com tamanho do bloco igual ou superior a 1MB, considerado neste trabalho como bloco de tamanho grande. Para blocos pequenos o desempenho sofreu queda considerável, em alguns casos o desempenho do Lustre ficou abaixo do obtido com o NFS. Os dados apresentados reforçam a ideia de que para se obter bom desempenho, independente do padrão de acesso, é necessário tornar os acessos que o processo realiza contíguos dentro do arquivo.

5. CONCLUSÃO

O contínuo avanço na tecnologia de desenvolvimento de processadores permite a implementação de *clusters* com milhares de núcleos de processamento. Na medida em que o número de processadores aumenta, a demanda por E/S também cresce. Além disso, há um crescimento no número de aplicações intensivas em dados, corroborando com a demanda por sistemas de armazenamento eficientes.

Como os dispositivos de armazenamento não evoluíram da mesma forma que os processadores e memórias em termos de *throughput*, uma alternativa é o uso de sistemas de arquivos paralelos, onde vários dispositivos de armazenamento são utilizados em conjunto para melhorar o desempenho final.

Entretanto, esses sistemas são complexos, com vários componentes que influenciam no desempenho final, que inclusive, tornam o processo de avaliação uma tarefa complicada. Por isso a importância de avaliar esses sistemas, de forma a verificar como aplicações podem ser desenvolvidas para obter maior desempenho, ou mesmo, descobrir se o sistema de armazenamento é adequado para a classe de aplicações alvo.

A fim de contribuir com esse processo, este trabalho realizou uma avaliação do sistema de arquivos paralelos Lustre. O objetivo foi investigar o Lustre de forma a compreender melhor sua arquitetura e funcionamento, e como o mesmo se comporta sob alguns padrões de acesso de aplicações paralelas científicas. Com base nas informações obtidas, usuários com demandas semelhantes podem planejar as operações de E/S de suas aplicações para melhorar o desempenho.

O processo de avaliação foi dividido em duas classes de testes. A primeira visava investigar o comportamento do Lustre ao variar fatores como tamanho do arquivo, tamanho da unidade de transferência, número de clientes, número de servidores de armazenamento, distribuição da carga entre os servidores de armazenamento e acesso não alinhado ao tamanho do *stripe*. Os resultados permitiram melhorar o entendimento da arquitetura e configuração do Lustre, além de auxiliar na identificação das limitações do ambiente de teste. A segunda classe estava voltada sobre os padrões de acesso desempenhados por aplicações paralelas reais. Alguns estudos investigaram a forma como aplicações paralelas científicas executam operações de E/S sob máquinas paralelas (projeto CHARISMA). Essas informações foram utilizadas para elaborar as cargas de trabalho da avaliação.

A seguir as conclusões obtidas após a execução dos experimentos da primeira classe de testes:

- Tamanho do arquivo: os resultados mostraram que a escolha do tamanho do arquivo é importante para avaliar de forma efetiva o Lustre. Os mecanismos de *cache* cliente e *read-ahead* são utilizados de maneira agressiva, sendo necessário selecionar o tamanho do arquivo com cuidado. Caso contrário os resultados obtidos podem não representar o desempenho do sistema de arquivos, mas sim a velocidade de acesso a memória dos nós, mascarando as limitações do ambiente de teste. Além disso, os resultados também mostraram que as operações de leitura tem desempenho inferior as operações de escrita.

- Tamanho da unidade de transferência: variar o tamanho da unidade de transferência não afetou de forma significativa o desempenho do Lustre. Os mecanismos de *cache* cliente e *read-ahead* suprimiram qualquer efeito relacionado ao tamanho das transferências ao sistema de arquivos.

- Número de clientes e servidores de armazenamento: esses experimentos permitiram verificar o quão escalável o Lustre pode ser e quais são os limites do ambiente de teste. A partir da análise dos resultados foi possível localizar os gargalos e sugerir modificações para melhorar o desempenho do subsistema de E/S do ambiente de teste.

- Distribuição de carga entre servidores de armazenamento: os resultados deste experimento mostraram que o ideal é manter o número de nós clientes múltiplo dos nós de armazenamento. Dessa maneira os dados podem ser distribuídos de forma balanceada sobre os servidores de armazenamento, melhorando o desempenho.

- Acesso não alinhado ao tamanho do *stripe*: O acesso não alinhado entre o tamanho do bloco que um processo escreve/lê em relação ao tamanho do *stripe* no Lustre pode serializar o acesso ao arquivo. Os resultados mostraram que o acesso alinhado é fundamental para se obter bom desempenho.

A execução desses experimentos foi importante na compreensão do funcionamento do Lustre. As análises e resultados obtidos auxiliaram na definição de alguns fatores para a próxima classe de testes voltada aos padrões de acesso. Nessa classe foram avaliados os padrões *file-per-proc*, *segmented-access*, *simple-strided* e *random-strided*. Em todos os experimentos foram utilizados o mesmo número de processos e configurações do Lustre.

Os padrões *file-per-proc* e *segmented-access* são diferentes em termos de número de arquivos acessados por processo. O primeiro representa aplicações onde cada processo acessa um arquivo exclusivo, o segundo aplicações onde todos os processos acessam um arquivo compartilhado. Contudo, o desempenho obtido ao simular os dois padrões foi semelhante, em ambos, os processos acessam blocos contíguos de dados. Os resultados obtidos indicam que aplicações com esses padrões de acesso devem apresentar bom desempenho no Lustre.

Em *simple-strided*, cada processo faz acesso a blocos de dados de mesmo tamanho, porém esparsos dentro do arquivo. Dependendo da aplicação podem-se ter blocos grandes ou pequenos. Foram simuladas aplicações com ambos os casos e quando o acesso do processo se dá em blocos pequenos o desempenho geral sofre queda considerável. Para blocos pequenos, em alguns casos o NFS apresentou desempenho superior ao Lustre. Nesses casos é interessante utilizar estratégias para transformar acessos não contíguos em acessos contíguos ao arquivo. Uma estratégia geralmente utilizada consiste no *two-phase* I/O [41] e [42].

O padrão *random-strided* é semelhante ao *simple-strided*, porém os acessos são a blocos de tamanhos diversos a cada fase de execução da aplicação. Para simular esse padrão foi necessário implementar uma aplicação MPI que, a cada fase de execução, acessava blocos de tamanhos diversos. Os

resultados obtidos foram semelhantes aos obtidos com *simple-strided*, o tamanho do bloco tem forte influência na vazão de dados. Outra questão é o tamanho do bloco aleatório, que dificulta a questão de manter o tamanho do bloco alinhado ao tamanho do *stripe*. É provável que alguns acessos sejam serializados influenciando no desempenho da aplicação.

Por fim, a avaliação teve como ambiente alvo um *cluster* Linux de pequena escala, onde o NFS é o padrão de fato. Mesmo nesse tipo de ambiente o uso do Lustre mostrou-se uma alternativa interessante. Para aplicações que fazem muita E/S, o NFS cria um gargalo no acesso aos dados, prejudicando o desempenho final da aplicação. Outro fator que deve ser considerado ao usar um sistema de arquivos paralelos é a redução do número de nós de processamento, para servir como nós de armazenamento. Para não se ter reduzido esse poder de processamento, o Lustre permite a execução simultânea de *software* cliente e servidor no mesmo nó. Dessa forma, *clusters* de pequena dimensão podem tirar benefício do Lustre, sem perder capacidade de processamento.

Em termos gerais, o Lustre é um sistema de arquivos paralelos capaz de fornecer alta vazão de dados para as aplicações. Mostrou que pode ser adequado para todos os padrões de acesso avaliados neste trabalho. Entretanto, oferece um ganho maior quando a aplicação leva em consideração os aspectos de sua arquitetura. Por exemplo, aplicações onde o tamanho das requisições de E/S estão alinhados ao tamanho do *stripe* apresentam bom desempenho. Por outro lado, quando os tamanhos não estão alinhados, o desempenho sofre grande impacto. Além disso, para aproveitar os recursos fornecidos pelo Lustre, como *cache* cliente e *read-ahead*, os desenvolvedores devem, na medida do possível, tornar os acessos realizados pelos processos contíguos dentro do arquivo.

Como trabalho futuro sugere-se avaliar outros padrões de acessos de aplicações paralelas, não limitando-se somente as de cunho científico. Além disso, sugere-se avaliar outras tecnologias de redes de interconexão e configurações do Lustre, a fim de verificar o impacto sobre o desempenho do sistema de arquivos e fornecer maiores informações para profissionais de TI que pretendam usá-lo.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] J. Borrill, L. Oliker, J. Shalf, H. Shan. "Investigation of leading HPC I/O performance using a scientific-application derived benchmark". In *SC '07: Proceedings of the 2007 ACM/IEEE Conference on Supercomputing*, 2007, pp. 1-12.

[2] J. Borrill, L. Oliker, J. Shalf, H. Shan, A. Uselton. "HPC global file system performance analysis using a scientific-application derived benchmark". *Parallel Comput.*, vol. 35-6, fereveiro 2009, pp. 358-373.

[3] R. Buyya. *"High Performance Cluster Computing: Architectures and Systems"*. New Jersey: Prentice Hall PTR, 1999, 664p.

[4] P. Carns, W. Ligon III, R. Ross, R. Thakur. "PVFS: A Parallel File System for Linux Clusters". In *Proceedings of the Third Annual Linux Showcase and Conference*, 2000, pp. 317-327.

[5] P. M. Chen, D. A. Patterson. "Storage Performance - Metrics and Benchmarks". *Proceedings of the IEEE*, vol. 81-8, Agosto 1993, pp. 1151-1165.

[6] Hewlett Packard Company. "Netperf: A network performance benchmark". Capturado em: http://www.netperf.org/netperf/training/Netperf.html, Outubro 2009.

[7] J. Dollimore, T. Kindberg, G. Coulouris. *"Distributed Systems: Concepts and Design"*. Massachusetts: Addison Wesley, 2005, 944p.

[8] Message Passing Interface Forum. "MPI-2: Extensions to the Message-Passing Interface". Capturado em: http://www.mpi-forum.org/docs/docs.html, Outubro 2009.

[9] B. Fryxell, K. Olson, P. Ricker, F. X. Timmes, M. Zingale, D. Q. Lamb, P. MacNeice, R. Rosner, J. W. Truran, H. Tufo. "FLASH: An Adaptive Mesh Hydrodynamics Code for Modeling Astrophysical Thermonuclear Flashes". *The Astrophysical Journal Supplement Series*, vol. 131-1, Novembro 2000, pp. 273-334.

[10] F. Garcia-Carballeira, A. Calderon, J. Carretero, J. Fernandez, J. M. Perez. "The Design of the Expand Parallel File System". *International Journal of High Performance Computing Applications*, vol. 17-1, Mar-Abr 2003, pp. 21-37.

[11] G. A. Gibson, D. Stodolsky, F. W. Chang, W. V. C. Ii, C. G. Demetriou, E. Ginting, M. Holl, Q. Ma, L. Neal, R. H. Patterson, J. Su, R. Youssef, J. Zelenka. "The Scotch Parallel Storage Systems". In *Proceedings of 40th IEEE Computer Society International Conference (COMPCON 95)*, 1995, pp. 403-410.

[12] W. Gropp, E. Lusk, R. Thakur. *"Using MPI-2: Advanced Features of the Message-Passing Interface"*. Massachusetts: MIT Press, 1999, 406p.

[13] E. Hermann. "Dinamismo de Servidores de Dados no Sistema de Arquivos dNFSp". Dissertação de Mestrado, Universidade Federal do Rio Grande do Sul, 2006, 101p.

[14] D. Hildebrand, P. Honeyman. "Exporting Storage Systems in a Scalable Manner with pNFS". In *MSST '05: Proceedings of the 22nd IEEE / 13th NASA Goddard Conference on Mass Storage Systems and Technologies*, 2005, pp. 18-27.

[15] J. H. Howard. "On Overview of the Andrew File System". In *USENIX Winter Technical Conference*, 1988, pp. 23-26.

[16] IEEE/ANSI. *"Portable Operating System Interface (POSIX) – Part 1: System application program interface (API)"*. pub-IEEE, 1996, 784p.

[17] Cluster File System Inc. "Lustre File System: High-Performance Storage Architecture and Scalable Cluster File System". White Paper, Cluster File System Inc., 2007, 17p.

[18] Gluster Inc. "GlusterFS User Guide". Capturado em: http://gluster.com/community/documentation/index.php/User_Guide, Março 2010.

[19] Sun Microsystems Inc. "NFS: Network File System Protocol Specification (RFC 1094)". Capturado em: http://www.ietf.org/rfc/rfc1094.txt, Março 2009.

[20] R. V. Kassick. "Reconfiguração Automática de I/O para Aplicações Paralelas no Sistema de Arquivos dNFSp2". Dissertação de Mestrado, Universidade Federal do Rio Grande do Sul, 2010, 105p.

[21] D. Kotz, N. Nieuwejaar. "Dynamic File-Access Characteristics of a Production Parallel Scientific Workload". In *Supercomputing '94: Proceedings of the 1994 Conference on Supercomputing*, 1994, pp. 640-649.

[22] D. Levi, P. Meyer, B. Stewart. "Simple Network Management Protocol (SNMP) Applications". Capturado em: http://www.ietf.org/rfc/rfc3413.txt, Fevereiro 2009.

[23] W. Loewe, T. McLarty, C. Morrone. "IOR Benchmark". Capturado em: http://sourceforge.net/projects/ior-sio, Maio 2009.

[24] H. Meuer. “TOP500 Supercomputer Sites”. Capturado em: http://www.top500.org, Dezembro 2009.

[25] E. L. Miller, R. H. Katz. “Input/Output Behavior of Supercomputing Applications”. In *ACM/IEEE Conference on Supercomputing*, 1991, pp. 567-576.

[26] J. H. Morris, M. Satyanarayanan, M. H. Conner, J. H. Howard, D. S. Rosenthal, F. D. Smith. “Andrew: a distributed personal computing environment”. *Communications of the ACM*, vol. 29-3, Outubro 1986, pp. 184-201.

[27] N. Nieuwejaar, D. Kotz. “The Galley parallel file system”. In *ICS ’96: Proceedings of the 10th International Conference on Supercomputing*, 1996, pp. 374-381.

[28] N. Nieuwejaar, D. Kotz, A. Purakayastha, C. S. Ellis, M. L. Best. “File-Access Characteristics of Parallel Scientific Workloads”. *IEEE Transactions on Parallel and Distributed Systems*, vol. 7-10, Outubro 1996, pp. 1075-1089.

[29] PUCRS. “LAD - Laboratório de Alto Desempenho”. Capturado em: http://www.lad.pucrs.br/, Dezembro 2009.

[30] A. Purakayastha, C. Ellis, D. Kotz, N. Nieuwejaar, M. L. Best. “Characterizing Parallel File-Access Patterns on a Large-Scale Multiprocessor”. In *Proceedings of the Ninth International Parallel Processing Symposium*, 1994, pp. 165-172.

[31] R. Rabenseifner, A. E. Koniges. “Effective File-I/O Bandwidth Benchmark”. In *Proceedings of the 6th International Euro-Par Conference*, 2000, pp. 1273-1283.

[32] R. Ross. “MPI-IO Tile Benchmark”. Capturado em: http://www.mcs.anl.gov/research/projects/pio-benchmark/, Março 2010.

[33] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, B. Lyon. “The Design and Implementation of the Sun Network File System”. In *Proceedings Summer 1985 USENIX Conference*, 1985, pp. 119-130.

[34] P. Schwan. “Lustre: Building a File System for 1,000-node Clusters”. In *Proceedings of the 2003 Linux Symposium*, 2003, pp. 400-407.

[35] Z. Sebepou, K. Magoutis, M. Marazakis, A. Bilas. “A comparative experimental study of parallel file systems for large-scale data processing”. In *LASCO’08: First USENIX Workshop on Large-Scale Computing*, 2008, pp. 1-10.

[36] M. Seger. “Collectl – collect for linux”. Capturado em: http://collectl.sourceforge.net/index.html, Dezembro 2009.

[37] H. Shan, K. Antypas, J. Shalf. "Characterizing and predicting the I/O performance of HPC applications using a parameterized synthetic benchmark". In *SC '08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, 2008, pp. 1-12.

[38] F. Shorter. "Design and Analysis of a Performance Evaluation Standard for Parallel File Systems". Dissertação de Mestrado, Clemson University, 2003, 68p.

[39] P. K. Sinha. *"Distributed Operating Systems: Concepts and Design"*. New York: Wiley-IEEE Press, 1996, 764p.

[40] A. S. Tanenbaum, M. V. Steen. *"Sistemas Distribuídos: princípios e paradigmas"*. São Paulo: Pearson Prentice Hall, 2007, 402p.

[41] R. Thakur, W. Gropp, E. Lusk. "Data Sieving and Collective I/O in ROMIO". In *FRONTIERS '99: Proceedings of the The 7th Symposium on the Frontiers of Massively Parallel Computation*, 1999, 182p.

[42] R. Thakur, W. Gropp, E. Lusk. "On implementing MPI-IO portably and with high performance". In *IOPADS '99: Proceedings of the Sixth Workshop on I/O in Parallel and Distributed Systems*, 1999, pp. 23-32.

[43] A. Traeger, E. Zadok, N. Joukov, C. P. Wright. "A nine year study of file system and storage benchmarking". *ACM Transactions on Storage*, vol. 4-2, Maio 2008, pp. 1-56.

[44] F. Wang, Q. Xin, B. Hong, S. A. Brandt, E. L. Miller, D. D. E. Long, T. T. Mclarty. "File System Workload Analysis for Large Scale Scientific Computing Applications". In *Proceedings of the 21st IEEE / 12th NASA Goddard Conference on Mass Storage Systems and Technologies*, 2004, pp. 139-152.

[45] P. Wong, R. D. Wijngaart. "NAS Parallel Benchmarks I/O Version 2.4". Relatório Técnico, Computer Sciences Corporation, NASA Advanced Supercomputing (NAS) Division, 2003, 6p.

[46] W. Yu, H. S. Oral, R. S. Canon, R. Vetter, J. S. Sankaran. "Empirical Analysis of a Large-Scale Hierarchical Storage System". In *Euro-Par '08: Proceedings of the 14th International Euro-Par Conference on Parallel Processing*, 2008, pp. 130-140.

[47] W. Yu, J. S. Vetter, S. Oral. "Performance characterization and optimization of parallel I/O on the Cray XT". In *Proceedings of the 22nd IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2008, pp. 1-11.