UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
(INE)

Tiago Steinmetz Soares

UMA ARQUITETURA PARALELA PARA O ARMAZENAMENTO DE IMAGENS MÉDICAS EM SISTEMAS DE ARQUIVOS DISTRIBUÍDOS

Florianópolis

2012

Tiago Steinmetz Soares

UMA ARQUITETURA PARALELA PARA O ARMAZENAMENTO DE IMAGENS MÉDICAS EM SISTEMAS DE ARQUIVOS DISTRIBUÍDOS

Dissertação submetida ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Santa Catarina para a obtenção do Grau de Mestre em Ciências da Computação.
Orientador: Prof. Mario Antonio Ribeiro Dantas

Florianópolis

2012

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Soares, Tiago Steinmetz
Uma arquitetura paralela para o armazenamento de imagens médicas em sistemas de arquivos distribuídos [dissertação] / Tiago Steinmetz Soares ; orientador, Mário Antônio Ribeiro Dantas - Florianópolis, SC, 2012.
118 p. ; 21cm

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico. Programa de Pós-Graduação em Ciência da Computação.

Inclui referências

1. Ciência da Computação. 2. HDF5. 3. Sistemas de Arquivos Distribuídos. 4. DICOM. 5. E/S paralelo. I. Dantas, Mário Antônio Ribeiro . II. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Ciência da Computação. III. Título.

Tiago Steinmetz Soares

UMA ARQUITETURA PARALELA PARA O ARMAZENAMENTO DE IMAGENS MÉDICAS EM SISTEMAS DE ARQUIVOS DISTRIBUÍDOS

Esta Dissertação foi julgada aprovada para a obtenção do Título de "Mestre em Ciências da Computação", e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Santa Catarina.

Florianópolis, 3 de setembro 2012.

Prof. Ronaldo dos Santos Mello
Coordenador do Curso

Prof. Mario Antonio Ribeiro Dantas
Orientador

Banca Examinadora:

Prof. Mauricio Aronne Pillon

Prof. Luis Fernando Friedrich

Prof. Joao Bosco Mangueira Sobral

AGRADECIMENTOS

Gostaria de agradecer primeiramente aos meus pais, pelo amor, pelo carinho, pelo apoio e pela presença em todos os momentos de minha vida.

Ao meu orientador Prof. Dr. Mario Dantas, por seus ensinamentos, dedicação e pela confiança ao me incumbir vários projetos e responsabilidades. A ele também sou grato pela oportunidade de uma viagem para o Canadá, por um período de 5 meses, em um intercâmbio acadêmico na UWO (University of Western Ontario), que me proporcionou momentos incríveis e experiências que levarei para o resto da vida.

Ao meu amigo Douglas Macedo, que me introduziu e me apoiou a todo o momento neste período, repercutindo tanto em conhecimento teórico quanto no crescimento profissional.

Também gostaria de agradecer pelo professor Michael A. Bauer da UWO, que me orientou durante o período de intercâmbio no Canadá.

Ao casal de amigos Lucio e Elisa de London (CA) sou muito grato pela hospedagem e principalmente pelo apoio nos primeiros meses do intercambio.

Aos membros do Laboratório de Pesquisa em Sistemas Distribuídos (LaPeSD) e do Laboratório de Telemedicina do HU, que sempre estiveram presentes nos momentos das dúvidas e também da descontração.

Por fim, aos meus amigos e familiares, que sempre estiveram próximos nos momentos de alegrias e principalmente nas dificuldades. OBRIGADO.

Difícil não é lutar por aquilo que se quer,
e sim desistir daquilo que se mais ama.

Bob Marley

RESUMO

Com a implantação da Rede Catarinense de Telemedicina tem-se verificado um aumento significativo no volume de imagens médicas, do padrão DICOM, geradas pelos dispositivos médicos interconectados nesta rede. Visando a manipulação dessas imagens médicas, foi desenvolvido em um projeto prévio, um servidor conhecido como CyclopsDCMServer, para a manipulação das imagens DICOM considerando a abordagem usando o *Hierarchical Data Format* (HDF5). Todavia, é esperado que a abordagem venha a encontrar gargalos devido ao crescimento no volume de dados e operações simultâneas que são submetidas ao servidor.

Com o objetivo de dar continuidade ao esforço para prover uma melhor escalabilidade ao servidor CyclopsDCMServer, nesta dissertação apresenta-se uma pesquisa no sentido de potencializar a implementação de um paradigma paralelo no servidor para o armazenamento e recuperação das imagens DICOM. Desta forma, desenvolveu-se um módulo considerando bibliotecas E/S paralelas de alto desempenho. Este módulo efetua uma comunicação com o servidor que é responsável pela realização do acesso paralelo no formato de dados hierárquico.

Visando a avaliação de desempenho da abordagem paralela, foram executados experimentos em diferentes sistemas de arquivos distribuídos. Os experimentos foram focados principalmente nas operações de armazenamento e recuperação das imagens médicas. Comparou-se o tempo médio de execução de cada operação em serial e paralelo. Foi coletado também o tempo de E/S em cada operação, para averiguar somente o desempenho do processo de escrita e leitura dos dados, descartando qualquer atraso que pudesse interferir nos resultados.

Os resultados empíricos demonstraram que, independente do sistema de arquivos, a abordagem paralela ainda não apresenta uma eficiência considerável, quando comparada com a arquitetura serial. A média do declínio de desempenho pode ser considerado em torno de 45% na operação de recuperação e 71% na operação de armazenamento. Verificou-se também que o aumento do número de processos paralelos pode causar uma perda maior de desempenho nesta abordagem.

Palavras-chave: HDF5. DICOM. MPI. E/S Paralelo. Sistemas de Arquivos Distribuídos.

ABSTRACT

With the deployment of *Catarinense* Network of Telemedicine has verified a meaningful increase in volume of medical images, DICOM standard, generated by medical devices interconnected on this network. In order to manipulate this medical images was develop in one previous project, a server known as CyclopsDCMServer, to manipulate DICOM images considering the approach Hierarchical Data Format (HDF5). However, it is expected that this approach will find bottlenecks due the spread of data size and simultaneously operations submitted to the server.

With focus to continue the effort to supply better scalability to the server CyclopsDCMServer, this dissertation presents a research in the sense to empowerment the implementation of a parallel paradigm in the server to storage and retrieve DICOM images. Thus, it was developed a module considering high performance parallel I/O libraries. This module performs a communication with the server that is responsible for the creation of parallel access in hierarchical data format

Aiming at the performance evaluation of the parallel approach, experiments were performed in different distributed file systems. The experiments were mainly focused on the operations of storage and retrieval of medical images. It was compared the average execution time of each operation in serial and parallel. It was also collected the I/O time in each operation, only to ascertain the performance of the process of writing and reading data, discarding any delay that could meddle the results.

The empirical results show that, regardless of file system, the parallel approach does not present a considerable efficiency when compared to the serial architecture. The average decline in performance can be seen at around 45 % in the recovery operation and 71 % in the storage operation. It was also observed that increasing the number of parallel processes can cause a larger loss of performance in this approach.

Keywords: HDF5. DICOM. MPI.Parallel I/O. Distributed File System.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

SUMÁRIO

1 INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

Aplicações alternativas a bancos de dados convencionais para armazenamento de dados tem sido uma das soluções mais procuradas pelos cientistas e empresas quando se trata de grandes volumes de dados, em sistemas de computação de alto desempenho. Dentre os alternativos que existem disponíveis atualmente, o *Hierarchical Data Format* (HDF) tem sido o mais utilizados em diversos áreas de pesquisa. HDF5 é a quinta versão deste formato, e foi desenvolvido pelo grupo *HDF Group* da Iniversidade de Illlinois, nos Estados Unidos.

Desde o lançamento do HDF, muitas aplicações de uso geral e científico começaram a usá-lo como uma alternativa eficiente e de alto desempenho para armazenar e acessar arquivos volumosos. Como exemplos de utilização, a NASA utiliza o HDF para armazenar dados de monitoramento global; aplicações clínicas para o gerenciamento de grandes coleções de imagens de raios-X; empresas petrolíferas que armazenam grande quantidade de dados de sondagens sísmicas 3D (HDFGROUP, 2011), pesquisas de armazenamento e unificação de dados biológicos em alto desempenho para lidar com grandes quantidade de imagens (DOUGHERTY et al., 2009) e pesquisas de alto desempenho em armazenamento de cálculos de física nuclear (LAGHAVE et al., 2009). Como também na área médica os hospitais enfrentam problemas com armazenamento de grandes quantidades de imagens geradas de dispositivos médicos que, por lei, precisam ser armazenadas durante anos. Desta forma o volume de imagens gerado diariamente é um problema constante dos servidores de banco de dados, principalmente em termos de escalabilidade.

Um dos objetos desta pesquisa é o servidor de imagens médicas DICOM CyclopsDCMServer. Este servidor vem sendo desenvolvido pelo Laboratório de Telemedicina da Universidade Federal de Santa Catarina no Brasil e tem o objetivo de fornecer armazenamento de imagens médicas DICOM (PIANYKH, 2008) provido por aparelhos médicos conectados à Rede Catarinense de Telemedicina (RTCM) (WALLAUER et al., 2008). Estas imagens hoje são armazenadas em bancos de dados convencionais.

Em Santa Catarina, a RTCM conecta diferentes instituições de saúde, como hospitais e instalações de cuidados básicos englobando 286 municípios do Estado. Esta rede fornece serviço de acesso a mais de

10 modalidades DICOM. Dentre elas temos: eletrocardiogramas; ressonância magnética; tomografia computadorizada; radiografia computadorizada (MACEDO et al., 2009). Para que ocorra essa integração entre servidor e modalidades DICOM, a RCTM utiliza o Sistema de Comunicação e Arquivamento de Imagens (*Picture Archiving and Communication System - PACS*), que engloba suporte de hardware e software para a maioria dos equipamentos médicos mascarando toda a parte de comunicação, segurança e acessibilidade (HUANG, 1999).

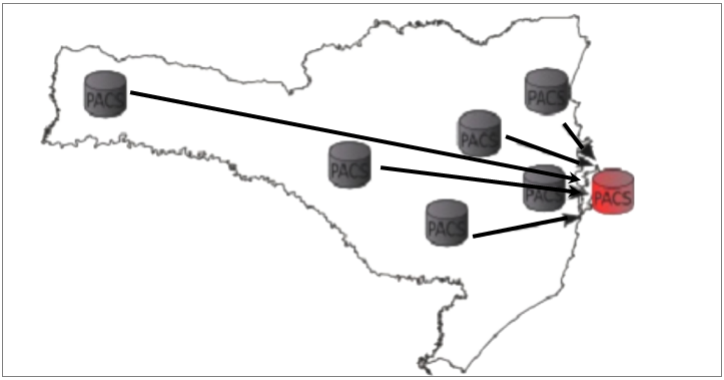


Figura 1 – Representação do sistema PACS em Santa Catarina, na qual todos os dados são replicados para uma base central na Telemedicina.

Em Santa Catarina, o sistema é acessível em praticamente todo o Estado, sendo que a tendência é, nos próximos anos, dar cobertura total a SC. Por lei, cada imagem médica deve permanecer acessível durante 20 anos, e cada imagem médica de uma tomografia possui, no mínimo, 512KBytes. É possível concluir que a base de imagens DICOM é bem abrangente e está em constante crescimento, requerendo grandes investimentos na manutenção dos dados. Analisando o problema similar em outras áreas e o resultado das pesquisas que utilizam base de dados alternativos em vez de banco de dados convencionais, o Laboratório de Telemedicina de Santa Catarina da Universidade Federal de Santa Catarina, no Brasil, iniciou a pesquisa em 2006 e o projeto piloto em 2007, procurando uma solução de armazenamento em alto desempenho, utilizando a API HDF5 e sistemas de arquivos distribuídos.

A versão de pesquisa deste servidor buscou utilizar o HDF5 como a base de dados para arquivos DICOM. Esta pesquisa iniciou-se em 2006 por Macedo *et al.*, com o objetivo de abordar questões da Teleme-

dicina, com base em sistemas de bancos de dados relacionais tais como escalabilidade, distribuição de informação, habilidade de usar técnicas de sistema de alto desempenho, visando minimizar os custos operacionais. Entre os procedimentos usuais para evitar o problema de escalabilidade, foi projetado uma camada de abstração para utilizar sistemas de alto desempenho distribuídos, como clusters e grids computacionais (MACEDO et al., 2009). Esta abordagem se distinguiu do sistema atual por armazenar todas as informações hierarquicamente, organizá-las e armazená-las no formato de dados HDF5. Os resultados dos experimentos obtidos foram bastante satisfatórios, obtendo-se rendimento de rapidez até cinco vezes superior em relação a banco de dados convencionais, em que se utiliza PostgreSQL.

Outro fator observável nas pesquisas que buscam maior desempenho em armazenamento é a utilização de bibliotecas para paralelização da leitura e escrita de grandes volumes dos dados, como por exemplos, o Dennis *et al.*(DENNIS et al., 2012) e Nam *et al.* (NAM; SUSSMAN, 2003) que são pesquisas que buscam utilizar bibliotecas de E/S em paralelo para maximizar a largura de banda de disco. Além disso, os sistemas de arquivos distribuídos nos últimos anos, desenvolveram melhorias na rotina de acesso paralelo via *middlewares*, como por exemplo, o ROMIO, que é uma biblioteca de alto de desempenho em E/S, suportado pelos sistemas de arquivos como PVFS e NFS por exemplo (DIVISION, 2011).

Desta maneira, este projeto busca utilizar as tecnologias mais atuais na área de computação distribuída, a fim de verificar se é possível melhorar ainda mais o desempenho do servidor CyclopsDCMServer. Para isso, foi projetada uma arquitetura para utilizar característica de acesso paralelo providas por sistemas de arquivos, e que possua compartibilidade com o formato HDF5 para o armazenamento de imagens médicas. Buscou-se também aplicar técnicas de alto desempenho em programação paralela em escrita e leitura de dados e, concomitantemente, verificar o desempenho desta abordagem, aplicando-a em diversos sistemas de arquivos distribuídos.

Os trabalhos submetidos (SOARES et al., 2011) e (SOARES et al., 2012) como experimentos iniciais desta abordagem mostraram que o tempo de leitura e escrita em paralelo no sistema de arquivos PVFS teve um desempenho eficiente sob determinadas imagens. Desta maneira, este trabalho tem o objetivo de realizar maior número de experimentos, com maior volume de dados e aumentar o número de componentes no ambiente de testes. Para avalizar o comportamento desta abordagem, buscou-se realizar os experimentos em diferentes sistemas

de arquivos distribuídos, que possuem como características distintas as suas técnicas de distribuição de dados e arquitetura de sistema.

1.2 OBJETIVOS

Este trabalho investiga uma maneira de contornar problemas de desempenho deparados na análise apresentada por Macedo *et al.*, e buscar novas maneiras de otimizar o desempenho desta tecnologia (HDF5), que disponibiliza várias características para aumentar a sua eficiência. Ainda será pesquisada uma abordagem buscando utilizar tecnologias de computação de alto desempenho, focando no paralelismo em cluster para obtenção de maior desempenho.

1.2.1 Objetivo Geral

Implantar uma arquitetura paralela na aplicação CyclosDCMServer e aplicar sob diversos sistemas de arquivos distribuídos, buscando aumentar o desempenho de armazenamento e recuperação de imagens médicas formatados hierarquicamente em HDF5.

1.2.2 Objetivos Específicos

Os objetivos específicos são:

- Estudar uma maneira eficiente para implantar o acesso paralelo no servidor CyclopsDCMServer;
- Pesquisar configurações necessárias para a implantação de uma API paralela em ambientes distribuídos;
- Desenvolver a camada de abstração paralela para o servidor CyclopsDCMServer;
- Realizar testes de desempenho, tanto de leitura quanto de escrita, para fins de comparação com o servidor atual;
- Realizar testes nos sistemas de arquivos distribuídos;

1.3 ORGANIZAÇÃO DO TEXTO

Esta dissertação inicia-se com a introdução dos conceitos utilizados nesta pesquisa - Telemedicina, PACS e DICOM - e, em seguida, uma introdução do que é HDF5 e suas principais estruturas, finalizando com suas características de acesso. Na sequência, no capítulo 2, são discutidos os conceitos básicos dos sistemas de arquivos distribuídos, desde as características principais até os tipos de sistemas de arquivos distribuídos. Nesta seção é dado ênfase aos agregados computacionais (clusters), pois este é o ambiente utilizado neste trabalho, discutindo os principais meios de programação paralela em clusters (MPI e PVM) e suas principais características.

No capítulo 3, são apresentados alguns exemplos de sistemas de arquivos distribuídos, com um detalhamento maior dos sistemas que foram utilizados para a realização dos experimentos.

No capítulo 4, é discutida a proposta da arquitetura, abrangendo os conceitos de abordagem serial e paralelo, a implementação e são levantados alguns trabalhos relacionados a esta pesquisa, desde trabalhos que utilizam acesso paralelo em HDF5 ou em arquivo, até trabalhos que buscam melhorar as técnicas de acesso em paralelo.

Em seguida, no capítulo 5, são descritos o ambiente e a forma de realização dos experimentos, seguindo-se os resultados e as análises. Por fim, no último capítulo, é relatada a conclusão desta pesquisa e são traçadas perspectivas sobre trabalhos futuros.

2 IMAGENS MÉDICAS

2.1 TELEMEDICINA

No fim dos anos 60 e começo dos anos 70, com a evolução da tecnologia da informação e da comunicação em geral, tendo em vista o alto custo para fornecer uma boa assistência médica, surgiu a necessidade de construir um sistema que buscasse minimizar os custos relativos ao transporte de pacientes. A partir dessa necessidade, surgiu o termo Telemedicina (BASHSHUR, 2002).

A Telemedicina vem sendo aplicada como uma forma de prover acesso à saúde para pessoas ou comunidades que, de uma forma ou de outra, estejam isoladas ou desprovidas de médicos qualificados (MACEDO et al., 2009). O objetivo é disponibilizar informações sobre o paciente de forma remota, sendo que estas informações podem ser acessadas pelos médicos de outras cidades através de aparelhos eletrônicos, como computadores, tabletes, celulares, etc. Sua utilização, nestes casos, pode reduzir o custo de transporte, diminuir os casos de mortes devido à espera de algum laudo médico ou durante um transporte de paciente e, além de aumentar a confiabilidade de uma medicação devido à integração de vários médicos especializados em uma determinada área.

Normalmente estes projetos são de grande interesse para instituições de saúde governamentais, que implementam programas de melhoria da saúde. A implantação da Telemedicina motiva-se principalmente pela diminuição de custos, porém existe um custo inicial para implantação, pois há necessidade de comprar equipamentos específicos e desenvolver softwares para análises e acessos. Como exemplos de países que implantaram a Temedicina temos: Estados Unidos, Alemanha, Japão, China e Brasil (MAIA; WANGENHEIM; NOBRE, 2006). Estes trabalhos têm em comum a implantação da Telemedicina voltada para a teleconsulta entre médico e paciente, geração de laudos por parte dos médicos e teleconferências para casos extremos.

A funcionalidade da Telemedicina se resume basicamente na transferência de informações e imagens médicas de um local pra outro. A comunicação entre médicos vai desde telefonemas até teleconferências. Quando um paciente faz algum exame de caso complicado, este é enviado a um ou mais médicos com a finalidade de se formar uma melhor opinião sobre os procedimentos a serem realizados.

No Brasil, especificamente em Santa Catarina, em parceria com

a Secretaria de Estado da Saúde de Santa Catarina, iniciou-se um projeto elaborado pelo grupo Cyclops da Universidade Federal de Santa Catarina chamado Rede Catarinense de Telemedicina (RCTM) (MAIA; WANGENHEIM; NOBRE, 2006). Este projeto possibilitou a disponibilização em rede de imagens, sinais e laudos médicos gerados a partir de estabelecimentos de saúde espalhados pelo Estado. Para este propósito, o grupo Cyclops (CYCLOPS, 2011), da Universidade Federal de Santa Catarina, foi designado para desenvolver uma solução customizada, baseada na tecnologia PACS.

2.1.1 *Picture Archiving and Communication System*

O arquivamento de imagens e sistema de comunicação (PACS) é uma tecnologia que fornece armazenamento e acesso a imagens médicas de várias modalidades, como eletrocardiogramas e ressonância magnética, via redes de comunicação. Hoje em dia, milhões de radiologistas estão analisando suas imagens através da Internet. A vantagem da utilização do sistema PACS é a de eliminar a necessidade de transmissão e acesso de arquivos fisicamente; filmes não seriam mais perdidos ou roubados, resultando na diminuição do tempo de transmissão veicular tanto da imagem como possivelmente do paciente, e além de reduzir o tempo dos laudos médicos quando há necessidade de avaliações de médicos especializados, evitando a espera do paciente.

Quando se trata de PACS, este necessariamente consiste em quatro componentes principais:

- Modalidades de imagem.
- Uma rede segura para a transmissão de informações
- Estações de trabalho para interpretar e analisar imagens.
- Armazenamento e recuperação.

No projeto RCTM de Santa Catarina, seguem esses quatros componentes principais, tendo como modalidades: radiografia computadorizada, tomografia computadorizada, ressonância magnética, medicina nuclear, ultrassom e x-ray angiografia. Essas modalidades utilizam o DICOM como padrão universal para transferência e armazenamento de imagens médicas.

2.1.2 DICOM

O padrão *Digital Imaging and Communications in Medicine* (DICOM) é um dos padrões mais conhecidos e fundamentais quando se trata de imagens médicas digitais. O padrão DICOM foi definido em 1992, e foi a terceira versão do *ACR-NEMA Standards Publication PS3*. Antes de o padrão ser estabelecido como DICOM, cada fabricante criava sua própria solução para armazenamento, visualização, impressão de imagens digitais e principalmente na comunicação, resultando diretamente em incompatibilidades entre dispositivos médicos. O primeiro padrão ACR-NEMA foi concebido em 1983 pelo Colégio Americano de Radiologia (*American College of Radiology*) com o princípio de tornar as imagens médicas digitais independentes dos fabricantes dos dispositivos, criando um padrão único para os equipamentos médicos e facilitando a expansão de imagens médicas digitais (PIANYKH, 2008).

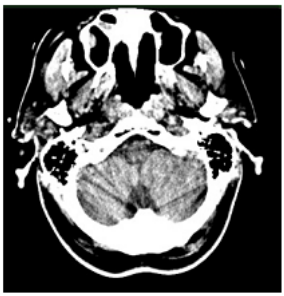


Figura 2 – Exemplo de uma imagem DICOM

Essa padronização herdou suas especificações mais importantes de outros padrões, onde as primeiras versões foram focadas na melhoria e correção dos problemas dos padrões anteriores. Essas especificações são relacionadas às interfaces de hardware introduzindo um conjunto de formato de dados e um conjunto mínimo de comandos de software. Duas versões do ACR-NEMA surgiram com o passar do tempo, mas somente em setembro de 1992 foi concluída a versão atual.

A terceira versão foi chamada DICOM 3.0 e veio com grandes revisões, fornecendo suporte a uma variedade de dispositivos digitais e suporte aos seus protocolos de comunicação. A partir desta versão, o padrão é anualmente revisto e atualizado com novos suplementos, caso necessário (PIANYKH, 2008). O padrão DICOM foi criado para cobrir

as seguintes necessidades contemporâneas:

- Um padrão universal na qual todos os dispositivos de aquisição de imagens digital produzem imagens DICOM e se comunicam através de redes DICOM.

- Excelente qualidade de imagem. DICOM suporta imagens com mais de 65,536 (16 bits) tons de cinza para exibição de imagens monocromáticas, em comparação com imagens em JPEGs e *bitmaps* (BMP), na qual são limitadas a 256 tons de cinza, oferecendo pouca visibilidade para diagnósticos. O padrão oferece uso das mais avançadas técnicas de representação de imagens digitais, proporcionando a elaboração de melhores de diagnósticos.

- Suporte completo a inúmeros parâmetros de aquisições de imagens e diferentes tipos de dados. É necessário armazenar grande quantidade de parâmetros da imagem DICOM e não exclusivamente a figura (*pixel data*. Esses parâmetros são, por exemplos, posição 3D do paciente, tamanho físico da imagem e espessura do corte. Normalmente esses dados são ricos em informações que facilitam o processamento e interpretação das imagens de diferentes maneiras.

- Codificação completa dos dados médicos. Arquivos e mensagens DICOM utilizam mais de 2000 atributos padronizados, que são codificados no dicionário DICOM. Esses atributos são utilizados para transmitir dados médicos deste nome do paciente até a cor da imagem de fundo. Estes dados oferecem essencial precisão para diagnósticos e captação de todos os aspectos da radiologia atual.

- Clareza na descrição dos dispositivos de imagens digitais e suas funcionalidades. O padrão define funcionalidades de dispositivos médicos em termos muito precisos e independentes de dispositivo. Trabalhando com dispositivos médicos através de suas interfaces DICOM torna-se um processo simples, deixando pouco espaço para erros.

DICOM foi criado para suportar diversas modalidades de imagens médicas, necessitando de um dicionário que lista todos os atributos do padrão. Os fornecedores do padrão podem criar seus próprios dicionários de atributos de dados proprietários. Neste caso, as estruturas desses dicionários devem seguir as mesmas regras pré-definidas.

Para organizar mais de 2000 itens, todos os itens são primeiramente divididos em grupos numerados (com base na similaridade do conteúdo). Os grupos são organizados por elementos individuais. Portanto, cada item é numerado com o seu próprio número, conhecido como *tag*. A *tag* é constituída de dois números chamados de grupo e elemento, e correspondem a um atributo do DICOM como "elemento de dados"(*data element*). Os grupos e elementos são números curtos, de tamanho fixo, e seguem uma estrutura hexadecimal. Como por exemplo. o atributo "Data de nascimento"do paciente é identificado pela *tag* "0010, 0030". Deste modo, todas as aplicações referem-se a um elemento usando a sua respectiva *tag* (grupo, elemento).

2.1.2.1 Estrutura hierárquica

A estrutura hierárquica de um arquivo DICOM segue por Paciente, Estudos, Séries e Imagens, respectivamente. Esta estrutura é importante para saber como consultar determinados dados de uma imagem. Por exemplo, para identificar uma série, primeiro deve-se encontrar o paciente e o estudo a que esta série pertence. Esta lógica é refletida em todas as interfaces DICOM.

Naturalmente, esta estrutura hierárquica reflete o que acontece no mundo real quando um paciente realiza algum exame médico. Uma pessoa chega a um hospital onde vários estudos podem ser programados (por exemplo, tomografia e exames de ultrassom). Vários estudos de acompanhamento podem ser necessários mais tarde, com o tempo, e cada estudo pode ter séries de imagens múltiplas (com ou sem contraste, com diferentes protocolos de imagem e assim por diante). Cada camada da hierarquia terá um único identificador (ID), tal como cada estudo de um paciente terá um único ID e assim por diante.

Como se pode observar na Figura 3, um paciente pode ter um ou mais estudos, cada estudo por conter uma ou mais séries, e cada série um ou mais imagens. A Figura ilustra também alguns atributos de cada camada da estrutura.

2.1.3 CyclopsDCMServer

Visado pela necessidade do desenvolvimento de um sistema PACS para atender as necessidades do Estado de Santa Catarina, foi necessário criar um servidor que comunique e armazene as imagens ge-

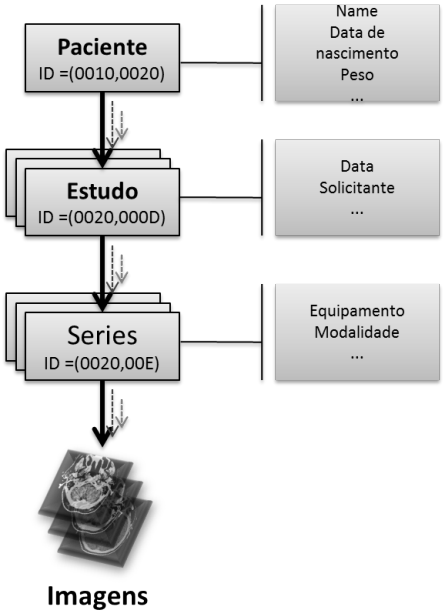


Figura 3 – Representação da estrutura hierárquica de um documento DICOM

radas pelos equipamentos médicos. Como a maioria dessas máquinas médicas utilizam o padrão DICOM para comunicação, o laboratório de Telemedicina desenvolveu um servidor DICOM chamado de CyclopsDCMServer.

CyclopsDCMServer foi desenvolvido com o objetivo de disponibilizar serviço de integração do padrão DICOM no ambiente PACS, a fim de prover armazenamento de imagens médicas e acessibilidade para a manipulação destas imagens através de estações de trabalho. O servidor foi desenvolvido e baseado no padrão DICOM. Por exemplo, o servidor implementa um tradutor das nomenclaturas (*tags*) baseado no dicionário DICOM para bancos de dados convencionais. O padrão tem como característica a flexibilidade para adicionar e remover novas funcionalidades. Desta maneira, como servidor foi desenvolvido para um ambiente específico, somente as modalidades eletrocardiogramas (ECG), ressonância magnética (MRI), tomografia computadorizada (CT), radiografia computadorizada (CR) são suportadas pelo ser-

vidor, que também suporta fácil implantação em ambientes hospitalares.

A função mais importante deste servidor é prover o armazenamento das imagens DICOM. Porém, para que ocorra a integração do servidor com as máquinas médicas, é necessário seguir o protocolo DICOM para a realização da comunicação entre eles. O desenvolvimento do servidor foi todo voltado ao protocolo, na qual é importante destacar os serviços providos. A comunicação do servidor é realizada por classes de serviços. As classes de serviço devem ser definidas através de um protocolo chamado DIMSE (DICOM Message Service Element). Este se situa na camada de aplicação que por sua vez encontra-se sobre a camada de transporte. O protocolo de transporte é o TCP (*Transfer Control Protocol*), que habilita um canal de comunicação confiável que assegura que os dados transferidos cheguem ao destino, além de ser um protocolo comum para muitos equipamentos.

Os equipamentos que participam da comunicação via protocolo DICOM possuem uma classificação dependendo do tipo da função exercida. Basicamente todos os equipamentos (servidores e máquinas) são chamados de entidades e são classificados em dois tipos: Service Class Provider (SCP) e Service Class User (SCU) . O SCP é quando a entidade provê o serviço e o SCU é quando a entidade faz a requisição do serviço. Quando um equipamento faz uma requisição de uma imagem ao servidor, este fluxo normal é dividida em três etapas. Primeiramente devem realizar uma associação a fim de verificar se as duas entidades estão registradas entre si, e a negociação, especificando quais operações podem ser executadas e quais extensões de serviço estão disponíveis para operação. Em seguida, são trocadas as mensagens referentes ao serviço a ser realizado e, por fim, duas mensagens de liberação e confirmação do serviço.

Existem 4 tipos de serviços DICOM, sendo que somente três são implementados no servidor:

- Armazenamento ou*C-Store (Composite Store)*. É o serviço em que a entidade SCP é responsável por executar o armazenamento de um objeto DICOM no bancos de dados, retornando ao SCU o resultado da operação.

- Busca ou *C-Find (Composite Find)*. Também é reponsabilidade da entidade SCP, no caso o servidor, realizar a busca do objeto em uma base de dados. A busca pode ser realizada sob qualquer nível de hierarquia mostrado na Figura 3, podendo definir qualquer atributo respectivo do nível como filtro da busca. A entidade

SCU deve fornecer a raiz da busca, o nível a ser pesquisado e um conjunto de chaves de busca e então o SCP é responsável por recuperar as informações. Por exemplo, para requerer os IDs dos estudos de um determinado paciente, podem ser utilizados os parâmetros de busca universal (ID dos estudos) e busca por valor único (nome do paciente).

- Movimentação ou *C-Move (Composite Move)*. O serviço de movimentação é utilizado no servidor CyclopsDCMServer como a forma de recuperação do objeto DICOM. A sua função é voltada para copiar o objeto do servidor para uma terceira entidade SCP, que normalmente é outro servidor de imagens. Este serviço requer duas conexões e associações diferentes para transmissão. Este procedimento torna mais seguro o acesso às imagens, pois a terceira entidade deve estar cadastrada no servidor e ainda é possível controlar o acesso via segurança *POSIX*.

- Recuperação ou *C-Get (Composite Get)*. Este serviço é similar ao de movimentação, porém não necessita de uma terceira entidade. A função de recuperação retorna o objeto que é enviado diretamente à entidade SCU que fez a requisição. Este método é menos recomendável, pois os níveis de segurança são menores. No caso do CyclopsDCMServer, este serviço não é implementado por vias de segurança de acesso.

2.2 *HIERARCHICAL DATA FORMAT*

O *Hierarchical Data Format* (HDF) (HDFGROUP, 2011) é um software que foi desenvolvido pelo grupo HDF da Universidade de Illinois, em conjunto com a NASA, no início dos anos 90, visando trabalhar com grandes quantidades de dados científicos. O principal objetivo do HDF é prover armazenamento e acesso eficiente em grandes volumes de dados, normalmente representados por matrizes extensas, sendo que comparado com uma estrutura de dados comuns, o processo seria ineficiente.

O HDF utiliza a formatação dos dados através da hierarquização, dividindo os dados em estruturas básicas similares a sistemas de arquivos. A sua forma de acesso é também diferenciada por prover acessos paralelos, via a interface MPI, proporcionando alto desempenho na leitura e escrita de dados.

O acesso é provido por meio de API disponibilizada nas lingua-

gens Fortran e C. Em ambas as linguagens, através das bibliotecas é possível gerenciar, criar, remover e alterar os dados contidos em um HDF5.

2.2.1 Estrutura do HDF5

No HDF5 existem três estruturas simples e essenciais que formam a base para a biblioteca: *dataset* (conjunto de dados), grupos e atributos.

- *dataset*. É um conjunto de dados composto por um cabeçalho e uma matriz unidimensional ou multidimensional de dados. O cabeçalho é dividido em nome, tipos dos dados, espaço dos dados e layout. O nome é utilizado para identificar os *datasets*. Os tipos de dados normalmente são atômicos (caracteres ou números) ou podem ser do tipo composto, semelhantes a estruturas da linguagem C/C++, que encapsula os tipos atômicos. O espaço dos dados (*dataspace*) é um metadado que pode ou não ser definido, que representa a dimensão dos datasets, os quais suportam tamanhos infinitos. Por fim, o layout é forma de armazenamento dos dados, que por padrão é contínuo, mas pode ser segmentado em blocos de tamanho fixo ou compacto.

- *Grupo*. São basicamente recipientes que contêm outros grupos e datasets. São análogos a diretórios de um sistema de arquivos UNIX, os quais podem conter outros diretórios ou arquivos. A grande diferença entre uma estrutura Grupo do HDF5 com um sistema de arquivos é que os grupos são ligados como um grafo direcionado, permitindo referências circulares. Grupos são importantes para organizar a estrutura de um arquivo HDF5 como uma estrutura de árvore. Todo arquivo HDF5 possui um grupo inicial chamado raiz ("/"). Na Figura 4, os grupos são representados no programa *HDFview* como diretórios. Do mesmo modo que os *datasets*, os grupos possuem nomenclatura. A partir dos nomes dos grupos, formam-se os caminhos (*paths*), os quais, são utilizados por alguma API para acessar os grupos e os *datasets*. Ambos os *datasets* e grupos podem estar associados ou compartilhados entres outros elementos similares.

- *Atributos*. São pequenos objetos metadados que descrevem a natureza de um dado primário (*dataset* ou grupo). Os atributos são caracterizados como tipos primitivos de qualquer linguagem

e são atômicos, ou seja, não podem se constituir de alguma estrutura mais complexa. Nos atributos não há armazenamentos especiais, tais como compressão ou *chunking*; não há acesso parcial ou capacidade de subdefinição para dados de atributos; não podem ser compartilhados e não podem conter outros atributos. Um atributo é armazenado no cabeçalho de uma outra estrutura, portanto, são ligados diretamente a essa estrutura.

Para melhor ilustração, a Figura 4 é retirada de uma aplicação chamada *HDFview*, que também é desenvolvida pelo mesmo grupo, e que disponibiliza uma interface gráfica para trabalhar com a maioria das funções disponibilizadas pela biblioteca. Esta Figura ilustra a estrutura de um arquivo HDF5.

A estrutura do HDF5 é similar a uma estrutura UNIX, onde há um diretório raiz que é identificado pelo caractere "/". O diretório é análogo a um grupo e o *dataset* é similar a um arquivo dentro de um grupo, ou seja, é o elemento que contém os dados. Assim, como sistema de arquivos, um grupo pode conter um ou mais *datasets* como pode conter outro grupos internamente.

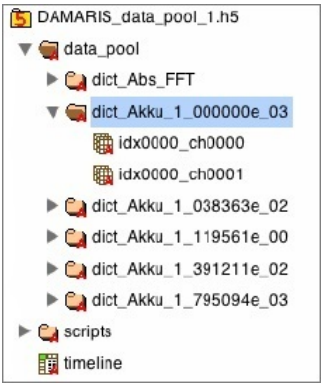


Figura 4 – Exemplo de uma estrutura HDF5 representada no programa *HDFview*

Outra estrutura interessante provida pelo HDF é a distinção de ligação simbólica (*symbolic-links*) e ligação direta (*hard-lyncs*). Ligações simbólicas são utilizadas para referenciar um *dataset* já existente, afim de que, ao remover essa ligação, não ocorra a a perda do dado. Em compensação, as ligações diretas são criadas no momento em que se

cria um *dataset* e ao serem removidas (sendo a única ligação ao objeto) acarretam a perda do acesso aos dados.

2.2.2 Características

As principais características do HDF estão relacionadas basicamente com a disposição dos dados e os modos de acesso a estes dados. A disposição dos dados é importante, pois caracteriza a sua forma de manipulação. No HDF, os dados estão disponíveis em formatos nativos e organizados em matrizes multidimensionais, mas também suporta formatos complexos, como estruturas compostas. Os formatos nativos normalmente são representados por descritores numéricos, textos e binários (por exemplo, H5T_NATIVE_INT) que tornam o acesso apropriado para qualquer plataforma. Os formatos complexos são estruturas compostas por esses diferentes formatos nativos, podendo ser desde estruturas como *Enumeration* até referencias a outros objetos de outras regiões.

A disposição dos dados é configurada no momento da criação dos próprios dados, na qual é preciso definir dois atributos antes da criação dos elementos *dataset*, *dataspace* e *dadatype*. Como pode observar na Figura 5, ambos *dataspace* e *dadatype* são composições do elemento *dataset*, e caso este seja destruído

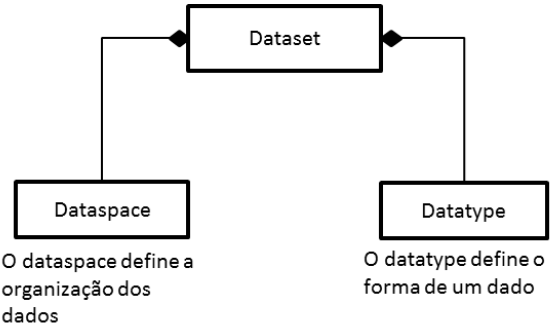


Figura 5 – Diagrama da composição da estrutura *dataset*

O datatype é o elemento que define o tipo de dados que será ou está contido no *dataset*, que como já foi discutido antes, pode apresentar diferentes tipos de dados. Normalmente para representar formatos

binários, utilizam-se números (inteiros e ponto flutuante) e caracteres (ASCII), diferenciando conforme o tipo de arquitetura. Por exemplo, linguagens como C, utilizam a representação de binário em *"char"*, *"short"*, *"int"* e *"long"*. Este tipo de elemento é importante principalmente para a interpretação dos dados corretamente, uma vez que são armazenados em sequências de *bits*.

O *dataspace* é o elemento que define as características do *dataset* relacionado à sua forma de armazenamento em disco e memória. Além de ser responsável por caracterizar o tipo de padrão de acesso utilizado. É através do *dataspace* que se obtém o número de dimensões o tamanho de cada dimensão de um vetor multidimensional no arquivo. No HDF, é possível definir um vetor de até três dimensões (x,y,z) onde cada dimensão pode conter tamanhos indefinidos. O *dataspace*, além de informar as características do *dataset* no arquivo HDF5, ele também é responsável por coordenar a forma dos dados armazenados na memória. Portanto, o *dataspace* está relacionado diretamente a operações de E/S.

Normalmente, a forma em que o vetor de dados é encontrado no disco e na memória é a mesma, mas o HDF5 também permite representar dados em memória de forma diferente dos dados em disco. Caso ambas as formas da memória e do disco sejam iguais, a definição é a mesma; caso contrário, poderá ser criado uma nova definição para a memória. Adicionalmente, nas funções de leitura, o vetor será lido em diferentes formas na memória, enquanto que na escrita os dados serão armazenados em forma especificada no arquivo. Essa característica permite que os dados do vetor possam ser acessados de diversas formas em diferentes espaços. Com isso, é possível acessar uma certa região de uma imagem, sem precisar alocar toda a imagem na memória, evitando perda de espaço na memória e processamento no parse da imagem.

2.2.2.1 Acesso Paralelo

A proposta do HDF5 paralelo (PHDF5) é prover facilidades aos usuários para usarem a biblioteca, apresentando compatibilidade com acesso serial. A Fígura 6 demonstra as camadas de acesso de uma aplicação paralela sob um sistema de arquivos paralelo. O seu desenvolvimento foi visado para dar suporte ao modelo de programação MPI. Portando, a maioria dos seus acessos paralelos são providos pela interface MPI.

A função de acesso aos dados via o *dataspace* de um *dataset* é chamado de *hyperslabs*. Esta função é responsável por combinar regiões

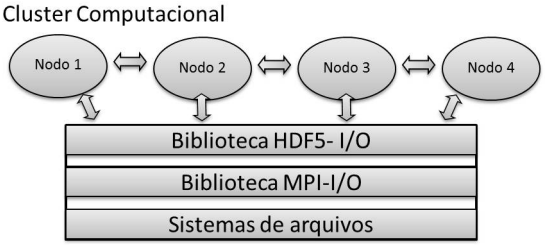


Figura 6 – Ilustração das camadas de acesso de HDF5 paralelo

de um vetor multidimensional do disco com uma região na memória, e vice-versa. Essas regiões podem conter diferentes formatos variados de objetos retangulares, podendo realizar mais uma vez a função, a fim de adquirir uma região desregular. Essa função pode ser visualizada na Figura 7, a qual ilustra o acesso de uma região em disco de uma determinada forma, guardando na memória de outra forma. No resultado dessa operação deve constar o mesmo número elementos em ambas as regiões.

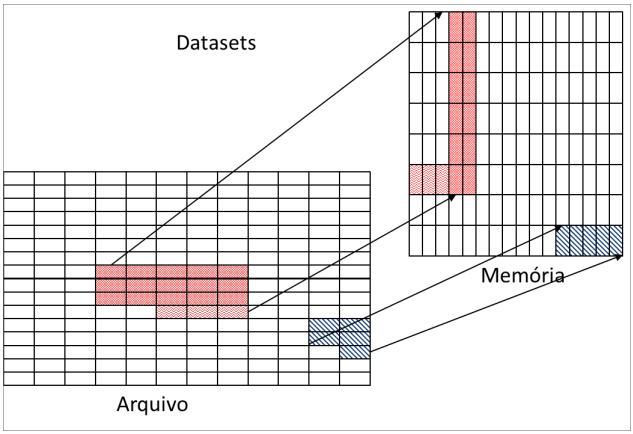


Figura 7 – Exemplo de uma transferência dos dados via *hyperslabs* do *dataspace* em arquivo para o *dataspace* da memória. Esta transferência deve manter o número de elementos de dados igual, enquanto a forma do *dataspace* pode ser diferente.

No caso do acesso paralelo em HDF5, os modelos *hyperslabs* são caracterizados pelo tipo de divisão do vetor dimensional entre os nodos paralelos e pela propriedade de acesso. A biblioteca tem disponíveis dois tipos de propriedades de acesso aos dados (coletivo e independente) e quatro modelos de hyperslabss (dados contíguos, regularmente espaçados, Padrão e *Chunk*). O acesso de dados coletivo é dependente de todos os nodos da aplicação corrente, havendo uma barreira para a concorrência dos nodos, a fim de o acesso ser simultâneo. Em compensação, no caso da propriedade de acesso independente não há necessidade de espera dos nodos na escrita ou leitura de dados, ou seja, cada nodo realiza o seu acesso independente dos outros processos.

A Figura 8 ilustra os quatro modelos implementados na API do HDF5 em paralelo, em que se utiliza a divisão das regiões em quadro nodos de E/S. No modelo Dados contíguos, a matriz pode ser dividida tanto por linhas ou por colunas de blocos de tamanhos idênticos, e gravada no sistema de arquivo em blocos. No modelo Regularmente espaçados, cada nodo contém colunas ou linhas inteiras intercaladas da matriz, que quando gravadas no arquivo, cada linha ou coluna pertence a um nodo. No modelo Padrão, cada nodo escreve uma parte da sua matriz no arquivo, não havendo divisões de acesso. Por fim, o modelo *Chunk* divide a matriz em blocos iguais de acordo com o número de nodos.

2.2.2.1.1 Configurações de acesso paralelo

Outra característica importante do PHDF5 são as suas configurações de acesso concorrente. No HDF5 há duas formas de configurações de acesso disponíveis providos também pelo ROMIO, chamados de independente (H5FD_MPIO_INDEPENDENT) e o coletivo (H5FD_MPIO_COLLECTIVE).

No acesso independente, cada processo acessa os dados concorrentemente sem precisar esperar os outros processos para a realização da tarefa. Uma vez que um arquivo é aberto em paralelo usando um comunicador (MPI_COMM), o arquivo pode ser acessado e modificado de forma independente por qualquer um dos processos no comunicador.

No acesso coletivo, no entanto, cada processo realiza a tarefa simultaneamente com os outros processos, havendo uma espera por cada processo caso necessária. Uma vez que todos os processos são obrigados a participar, há comunicação entre processos necessários que podem tornar o processo mais lento. A pesquisa de Laghave (LAGHAVE et al.,

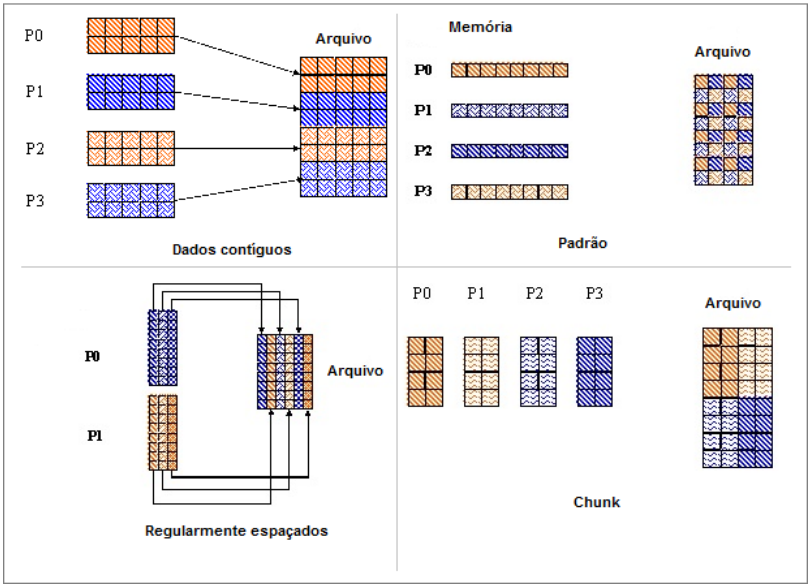


Figura 8 – Ilustrações dos tipos de *hyperslabs* implementados na API HDF5 paralelo (Figura modificada da versão original (HDFGROUP, 2011)).

2009) mostra que, dependendo da situação, o desempenho de acesso concorrente coletivo e independente pode oferecer o mesmo tempo de execução.

O H5FD_MPIO_COLLECTIVE implementa o *Collective buffering* e o Data Sieving, que provêm um conjunto de otimizações disponíveis no ROMIO que melhoram o desempenho de E/S paralelo em um arquivo compartilhado.

No caso do *Collective buffering*, por exemplo, também chamada de duas fases E/S, há quebra das operações de E/S em duas etapas. Para uma leitura coletiva (Figura 9), a primeira etapa utiliza um subconjunto de tarefas MPI (chamados agregadores) para se comunicar com os servidores da E/S (e.g. OSTs em Lustre) e ler um grande pedaço de dados em um *buffer* temporário. Na segunda etapa, os agregadores enviam os dados do *buffer* para o seu destino entre as tarefas restantes MPI usando a rotina ponto-a-ponto (*point-to-point*) do MPI.

A escrita coletiva faz o inverso, agregando os dados através de MPI em *buffer* nos nós agregadores, em seguida, escrever a partir dos

nós agregadores para os servidores da E/S. A vantagem do *Collective buffering* é prover menos nós na comunicação com os servidores da E/S, o que reduz o tempo de contenção provido da E/S em disco.

:

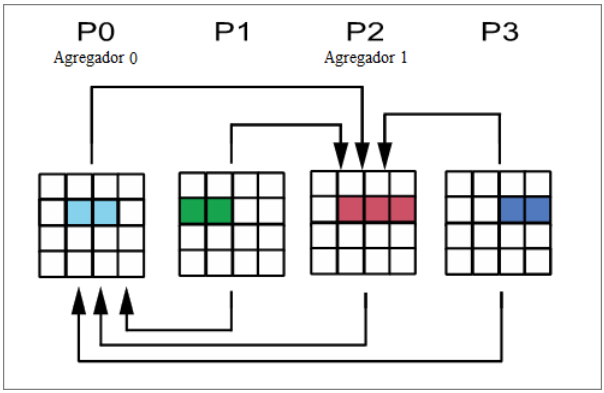


Figura 9 – Ilustração dos agregadores da E/S Coletivo (THORBECKE, 2012).

3 SISTEMAS PARALELOS E DISTRIBUÍDOS

3.1 INTRODUÇÃO

De acordo com George Coulouris (COULOURIS; DOLLIMORE; KINDBERG, 2007), sistema distribuído é: "coleção de computadores autônomos interligados através de uma rede de computadores e equipados com software que permita o compartilhamento dos recursos do sistema: hardware, software e dados". Sistemas distribuídos é resultado do avanço das tecnologias de redes de comunicação e microprocessadores de alto desempenho. São sistemas compostos por grandes números de processadores, conectados através de redes de alta velocidade e que trocam mensagens entre si. Um exemplo comum é a Internet, na qual diferentes tipos de computadores estão interligados e interagindo por trocas de mensagens de dados a fim de realizar um ou vários serviços.

Sistemas distribuídos têm com principal característica o compartilhamento de recursos. O termo recurso abrange desde componentes de hardware, como discos, impressoras e processadores, até entidades definidas pelo software, como arquivos, bancos de dados e objetos de dados de todos os tipos. O avanço de suas tecnologias, como a melhoria do tempo de execução, comunicação via redes, contornando problemas de concorrência, falhas e relógios globais, e a diminuição do valor agregado favoreceram o surgimento de sistemas distribuídos na rede mundial (COULOURIS; DOLLIMORE; KINDBERG, 2007). Outras características fundamentais para constituir um sistema distribuído que são constantes desafios são:

- *Transparência.* Para que o sistema distribuído seja utilizável para diversos usuários é necessário aplicar técnicas de transparência dos recursos distribuídos, a fim de evitar um conhecimento prévio do usuário sobre o sistema e enxergar como um sistema único. A transparência de um sistema distribuído é dividida por modalidades: Transparência de acesso, que permite operações idênticas a acessos remotos e locais; Transparência de concorrência, na qual vários processos operam juntos recursos compartilhados sem influência entre eles; Transparência de localização, que permite acesso aos recursos sem necessidade de saber a localização física.

- *Heterogeneidade.* Esta característica é importante para abranger diversos tipos de hardware de computadores (e. g. redes) e software (e. g. sistemas operacionais e linguagens de pro-

gramação). Ela permite troca de mensagens entre diferentes tipos de aplicações, hardwares e sistemas operacionais, providos por algum *middleware* ou protocolos de comunicação (Internet).

- *Sistemas Abertos.* Esta terminologia é dada quando um sistema pode ser estendido ou reimplementado de outras maneiras, ou seja, em que grau novos recursos podem ser adicionados. Caso seja heterogêneo, o sistema deve prover mecanismos de comunicação uniforme e interfaces do sistema públicas.

- *Segurança.* Quando se trata de um sistema com compartilhamento de vários recursos, com vários clientes, a segurança dos dados é essencial. A segurança deve propor confidencialidade dos dados (não expor dados privados a usuários não autorizados), integridade (proteção contra alteração ou remoção) e disponibilidade (proteção contra interferência com os meios de acesso aos recursos)

- *Tratamento de falhas.* Quando se trata de um sistema com vários computadores heterogêneos ou não, a ocorrência de falhas no sistema é comum, tanto em software como em hardware. Por exemplo, quando uma máquina desliga, as outras continuam funcionando, podendo ou não ocasionar falha no sistema. Portanto, o tratamento dessas falhas é difícil. Algumas técnicas são conhecidas para o tratamento dessas falhas, como: mascaramento de falhas, redundância e tolerância a falhas.

- *Concorrência.* O sistema tem que estar preparado para o acesso múltiplo a um determinado recurso compartilhado. Técnicas de programação, como semáforo, são recomendadas para o controle da concorrência.

Como exemplo famoso de um sistema distribuído, a Amazon se destaca pelo seu sistema bastante flexível. O foco é na utilização de computação nas nuvens e virtualização de máquinas. O serviço oferece a maioria das características que um sistema distribuído deve possuir. O foco da Amazon foi oferecer ao cliente total transparência ao usuário, onde os recursos são facilmente estendidos ou virtualizados através de um gerenciador dos recursos Web de acordo com a necessidade do usuário.

Outro fator importante relacionado ao sistema distribuído, que também é relevante quando se trata de utilização, é a sua organização no aspecto de como os computadores são conectados e como se comunicam entre si. A partir deste contexto, os computadores são divididos

em dois sistemas: multiprocessadores, na qual os computadores interligados compartilham a memória; multicomputadores, em que os computadores são interligados, porém não há compartilhamento de memória. Multiprocessadores são compostos por diversos processadores que compartilham a mesma memória por um barramento, são fortemente acoplados e normalmente gerenciados por um único sistema operacional. Multicomputadores são aglomerados de computadores conectados por uma rede comum, na qual trocam mensagens entre si. Cada máquina possui um sistema operacional próprio, onde são gerenciadas as suas memórias locais, não existindo compartilhamento de memória.

3.1.1 Tipos de SAD

Os sistemas distribuídos, de acordo com Steen e Tanenbaum (STEEN; TANENBAUM, 2007), podem ser divididos em três tipos: sistemas de informação distribuídos, sistemas embarcados distribuídos e sistemas de computação distribuída.

Sistemas de informação distribuídos são importantes, pois atualmente são bastante utilizados em grandes organizações. A sua característica foca na disponibilidade de execução de uma aplicação, compartilhando ou replicando o serviço entre várias máquinas. O princípio-base é cliente e servidor, na qual o servidor roda a aplicação e o cliente faz requisições. Essa aplicação que roda no servidor é caracterizada pela sua forma de distribuição de componentes. Esta forma pode ser dividida em dois grupos: a primeira, em que duas aplicações distintas se comunicam através de transações a fim de realizar um determinado serviço ao cliente; e a segunda, que utiliza um *middleware* de integração.

Existem vários tipos de *middleware* que proporcionaram essa infraestrutura para estes tipos de sistema, como por exemplo, RMI, RPC, Web *Services*, EJB, etc. É notável que com o desenvolvimento da tecnologia (processamento e redes), essas aplicações tornaram-se mais sofisticadas e gradualmente são mais separadas em componentes independentes, proporcionando maior interoperabilidade entre eles e disponibilidade ao cliente.

Sistema embarcado distribuído recentemente vem sendo mais comum devido a processadores de baixo consumo possuírem maior poder de processamento, em consequência, maior o poder operacional nestes dispositivos. Os dispositivos em um sistema embarcado distribuído caracterizam-se por serem pequenos, compactos, usam bateria e possuem conexão wireless. Hoje em dia são mais vistos como smartfones,

como também palmtop, tablete, etc. Um dos grandes problemas enfrentados quando se utiliza sistemas embarcados é a mudança constante de ambientes, havendo alguns locais que possuem conexão e outros não, portanto, estes dispositivos devem possuir aplicações de adaptação para essas mudanças constantes. Geralmente estes dispositivos conectam ao sistema em ordem de acessar e prover informações. Estes acessos significam fácil leitura, armazenamento, remanejamento e compartilhamento de informações entre outros dispositivos. Sabendo-se das mudanças de conectividades desses aparelhos, o local onde serão acessadas essas informações irá mudar todo o tempo.

Por fim, os sistemas de computação distribuída são sistemas de computação de alto desempenho, em que se encontra a computação paralela e descentralizada. Esse sistema divide-se em dois subgrupos: agregados computacionais (clusters) e computação em grade. Clusters consiste numa coleção de máquinas similares próximas umas das outras, conectadas em uma rede local de alta velocidade na qual cada nodo executará uma mesma aplicação. Em compensação, Grids são constituídos por um conjunto de sistemas distribuídos dispersos, normalmente conectados por uma rede comum e agregados sob um software gerenciador. O diferencial do sistema em Grid é a sua forma de submeter um serviço ao sistema que, ao invés de submeterem o serviço direto pra execução, são enviados ao gerenciador sob uma determinada pré-configuração do sistema, à qual é submetido quando este estiver disponível.

Como neste trabalho é focado em computação paralela em cluster, são discutidas na seção seguinte as características do sistemas de computação distribuídas em cluster.

3.2 AGREGADOS COMPUTACIONAIS

A capacidade de processamento mais rápido tem sido limitada por uma série de fatores como superaquecimento dos transistores, falta de espaço para o crescimento de elementos eletrônicos e atraso de comunicação. Por estas razões, poucas alternativas são conhecidas para conseguir um aumento da velocidade de processamento. As duas mais usualmente conhecidas são: técnicas de algoritmos mais eficientes, mas não é garantida para todos os tipos de aplicações; ou a opção de repartir tarefas, na qual um processo é distribuído a outros computadores para atuar no mesmo objetivo. A segunda técnica é mais utilizada e é conhecida como computação distribuída, na qual se permite que diversos

processadores conectados compartilhem recursos entre si, repartindo um trabalho entre eles a fim de ganhar desempenho na execução do mesmo. (DANTAS, 2005)

Por definição, agregados computacionais (clusters) é um aglomerado de computadores conectados, com um propósito de suprir a deficiência proveniente da limitação dos processadores. O que caracteriza um cluster é o agrupamento de máquinas em um limite geográfico. Este limite geográfico é derivado do tipo da conexão dos dispositivos, podendo ser apenas um gabinete ligado por uma interconexão ou uma sala com dispositivos separados e conectados por uma conexão local.

3.2.1 Programação Paralela em cluster

A utilização de clusters em ambientes paralelos tornou-se bastante usual devido ao seu baixo custo de implantação e limitação do poder de processamento dos processadores. Anteriormente, a programação paralela era restrita para arquiteturas massivamente paralelas e de memória compartilhada, restringindo bastante os desenvolvedores por se tratar de de uma arquitetura relativamente cara. Devido a este custo-benefício, a maioria das organizações que trabalham com alto desempenho tem adotado este paradigma.

Os gráficos da Figura 10 foram retirados do site *Top500 de supercomputadores*. O primeiro gráfico demonstra a evolução do tipo da arquitetura de sistemas utilizados desde 1993 até 2011, observando-se claramente a preponderância do uso de clusters nos últimos anos. No segundo gráfico, é ilustrada a evolução da utilização de supercomputadores em diversos segmentos, sendo que as indústrias se destacam no aumento de utilização de sistemas de supercomputação.

De fato, uma vantagem de computadores com memória distribuída é ter mais do que um simples grande processador; é também poder prover mais memória e mais cache. Quando se trata de ambiente de programação em cluster, dois softwares são usualmente conhecidos, MPI (Message passing Interface) e PVM (Parallel Virtual Machine) .

3.2.2 *Messaging Parsing Interface*

Message Passing Interface (MPI) é uma biblioteca padrão suportada nas linguagens C e Fortran77, a qual baseia-se em sub-rotinas de comunicação para programação paralela e projetada para funcionar

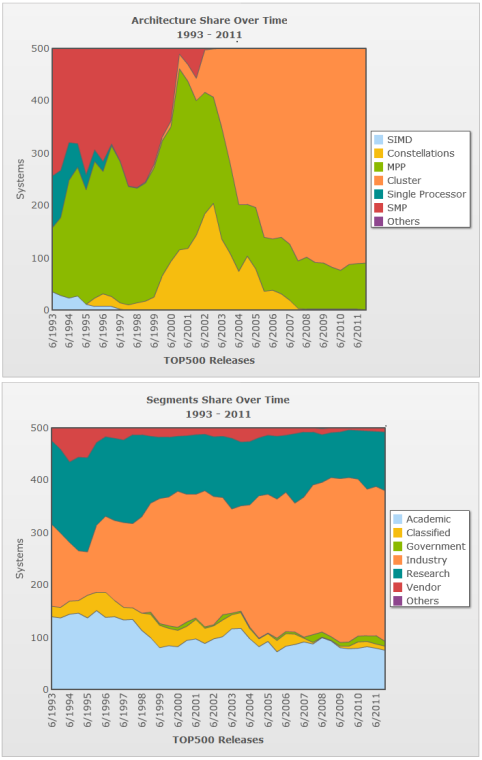


Figura 10 – Imagens do *Top500* (TOP500, 2011) mostrando a evolução do uso de diferentes sistemas de computação distribuída, e os segmentos que estão utilizando o clusters nos últimos anos

em uma ampla variedade de computadores paralelos. O seu paradigma é baseado na troca de mensagens (*message passing*) e é útil em ambos os computadores paralelos, como o SP2 da IBM, o ResearchT3D Cray, como também em máquinas conectadas a redes de estações de trabalho.

Na Figura 11, pode-se observar a principal característica da interface MPI, ilustrando a troca de mensagens entre os processos. Além da função de enviar e receber mensagens, o MPI também disponibiliza realizar primitivas como BroadCast e Gather, onde a primeira envia uma mensagem a todos os processos e o segundo recebe mensagem de todos os outros.

MPI foi desenvolvido por um fórum aberto e internacional cons-

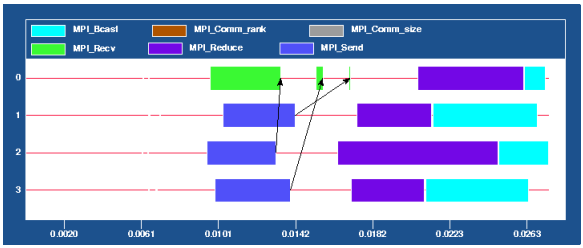


Figura 11 – Ilustração da troca de mensagens entre processos MPI, representada pela aplicação UpShot (HAATAJA, 2011)

tituído de indústrias, universidades e laboratórios do governo, com a finalidade de tornar a programação paralela mais acessível à comunidade tanto profissional como estudantil, tornando viável para os programadores desenvolverem bibliotecas de programas paralelos, com características eficientes e portáteis (PACHECO, 1997). Visando essa facilidade, múltiplas bibliotecas foram criadas por grupos distintos, diferenciando alguns focos durante os seus desenvolvimentos. Como exemplos dessas variações temos o MPICH, Open-MPI e CHIMP-MPI.

A interface MPI tende a fornecer abstração para os processos a fim de ajudar na semântica da comunicação. Pra isso, os processos são nomeados com um *rank* do grupo, baseado na ordem de inicialização. MPI também fornece suporte para ambos os modos MPMD e SPMD [1] de programação paralela. Além disso, pode suportar cálculos entre aplicações através de operações intercomunicação, que suportam a comunicação entre aplicações de um grupo e entre grupos. MPI fornece uma API *thread-safe*, que será útil em ambientes *multi-threads*, visto que atualmente o número de cores de processadores estão aumentando, podendo influenciar novas aplicações a utilizarem ambientes *multi-threads*.

As mais importantes vantagens dessa interface são propiciar universalidade (pois funciona bem em processadores separados e conectados por uma rede de comunicação lenta ou rápida), expressividade (MPI foi criado para ser usual e modelo completo para expressar algoritmos paralelos), fácil depuração (pois possui um bom gerenciador de erros

[1] SPMD (*Single process/Mutiple data*) e MPMD (*Multiple Processor/Multiple Data*) são taxonomias Flynn's para ambientes paralelos, na qual o SPMD tem como base múltiplos processos autônomos em diferentes processadores e executam o mesmo programa com diferentes entradas, enquanto o MPMD, múltiplos processos autônomos executam aos menos dois diferentes processos.

comparados com outros modelos paralelos) e principalmente desempenho, razão fundamental do *message-passing* continuar em ambientes de programação paralelo (GROPP et al., 1996).

Além de propiciar aumento do desempenho no processamento, o MPI também é utilizada para a paralelização de escrita e leitura de dados via interface *MPI-IO*. Desta maneira, o projeto ROMIO do Laboratório Nacional de Argonne em Chicago é uma implementação da interface *MPI-IO*, em que o seu objetivo foi desenvolver uma API de alto desempenho em E/S paralelo. ROMIO é basicamente uma implementação da especificação do MPI versão 2, que foi construído sob a especificação *message-passing* MPI versão 1 e do ADIO [2]. Portanto, o ROMIO é suportado por qualquer sistema de arquivo que implementa as funções ADIO na API do sistema de arquivos.

Como já citado no capítulo sobre HDF5 paralelo, as duas principais otimizações implementadas pelo ROMIO são o *data sieving* e o *Collective buffering*, em que ambos estão relacionadas com a otimização da E/S concorrente. Basicamente, ambas foram desenvolvidas para otimizar o acesso de dados armazenado de forma não contínua (técnica de armazenamento de bloco de dados, normalmente encontrado em sistemas de arquivos paralelos).

3.2.3 *Parallel Virtual Machine*

O *Parallel Virtual Machine* (PVM) também é um pacote de software sob o paradigma de trocas de mensagens cuja principal característica é a interoperabilidade entre grupos de computadores heterogêneos (Unix e Windows) por uma rede para ser utilizado como um único grande computador paralelo. O PVM foi desenvolvido por instituições compostas por Universidade do Tennessee, Laboratório Nacional de Oak Ridge e Universidade de Emory. A primeira versão foi escrita em ORNL em 1989, em seguida, a versão 2 foi reescrita pela Universidade do Tennessee em 1991 e por fim, a versão 3 foi lançada em 1993, aderindo tolerância a falhas e melhorias na portabilidade. A interface PVM busca ser simples, permitindo que a programação seja intuitiva (PRAKASH, 2011).

O usuário escreve sua aplicação como uma coleção de tarefas cooperativas, a qual acessa rotinas PVM através de uma biblioteca que permite a iniciação de tarefas através da rede, bem como comunicação e sincronização entre as tarefas. As comunicações são semelhantes às da

[2] ADIO é um conjunto de funções básicas para realização do E/S em disco.

interface MPI, a qual inclui envio e recebimentos de mensagens, como também as primitivas de programação concorrente *broadcast*, *gather* e *barrier*. Dentre as caracteristicas do PVM, temos:

- *Interoperabilidade.* Permite conexões entre diferentes tipos de arquiteturas, desde computadores massivamente paralelos (/emph-Massively Parallel Processor, MPP) a computadores simples.
- *Transparência.* A aplicação enxerga o sistema todo como um único sistema, omitindo os diferentes tipos hardware.
- *Paradigma de troca de mensagem.* Possui um comportamento semelhante ao MPI, fornecendo suporte para ambos os modos MPMD e SPMD.

Ambos MPI e PVM são constituídos por duas partes. A primeira é um *daemon* que reside em todas as máquinas, necessário para inicializar ambos os ambientes. Estes *daemon* executam em *background* a fim de realizar comunicação entre as outras máquinas, para comunicação do ambiente e outros fatores como *status* do ambiente.

A segunda parte são as bibliotecas de rotinas das interfaces. Essas bibliotecas normalmente provêm suporte para linguagem C, com o objetivo de facilitar a integração da aplicação sob o ambiente. Dentre as rotinas temos, por exemplo, a troca de mensagens, iniciação dos processos e coordenação de tarefas.

3.3 SISTEMAS DE ARQUIVOS DISTRIBUÍDOS

Atualmente, sistemas de arquivos distribuídos (SAD) vêm sendo mais constantes para os usuários, em que eles desconhecem o local físico de seus arquivos. O aumento da utilização de SADs por empresas e academias deve-se ao fato da deficiência na evolução do acesso aos discos rígidos quanto ao aumento do número de requisições, que adicionado com o crescimento do número de arquivos (tamanho e/ou quantidade), o acesso aos arquivos em um simples e único disco rígido torna-se um gargalo para diversos sistemas, principalmente quando se busca informações constantemente em disco.

Estes sistemas de arquivos têm por objetivo fornecer aos clientes os mesmos serviços providos por um sistema de arquivo convencional, com o adicional de poder ser acessado por qualquer máquina dentro de uma determinada rede, de forma centralizada ou não, com um desempenho comparável a de um disco local.

O sistema de arquivo distribuído mais conhecido atualmente por pequenas empresas e usuários comuns, é o *Sun Microsystem's Network File System* (NFS) (NOWICKI, 1989) , que busca a solução para a disponibilidade de arquivos na rede de forma centralizada. No entanto, como aplicações paralelas em cluster estão se tornando mais frequentes em aplicações que acessam grandes quantidades de dados, as soluções centralizadas tendem a saturar, principalmente, a capacidade da rede e até do disco com o aumento do número de requisições. Desta maneira, alguns sistemas buscaram explorar o paralelismo desses dispositivos para aumentar o desempenho dos sistemas de arquivos. Para isso, utilizaram-se alguns critérios para a vazão dos dados entre um conjunto de discos conectados em uma rede local (MARTIN; CULLER, 1999).

Desta forma, quando encontrada a presença de diversos clientes, procurou-se distribuir não só o armazenamento como também os servidores que atendem as requisições, a fim de dividir a carga associada à transmissão de dados na leitura ou escrita, explorando o paralelismo nas operações de entrada e saída. Essa exploração do paralelismo na leitura e escrita de dados é dada como principal característica de sistemas de arquivos paralelos (SAP). .

3.3.1 Arquiteturas de SAD

Nesta seção, são demonstrados os tipos de organizações presentes na maioria dos SAD classificados por Tanenbaum (STEEN; TANENBAUM, 2007). Estes são classificados por três tipos: cliente-servidor, baseado em clusters e sistemas simétricos.

3.3.1.1 Cliente-Servidor

A arquitetura cliente-servidor, ilustrada na figura 12, é o conceito base para o sistema de arquivos distribuídos mais comum de qualquer sistema UNIX quando se trata de compartilhamento de arquivos. A ideia por trás da arquitetura cliente-servidor é um servidor de dados disponibilizar o acesso a arquivos hospedados por qualquer cliente que esteja suscetível de utilizar o serviço. Os clientes acessam os arquivos como qualquer outro arquivo local, não importando ao cliente onde eles estão fisicamente. O NFS disponibiliza uma interface de acesso como a dos sistemas de arquivos locais, através da interface *POSIX*.

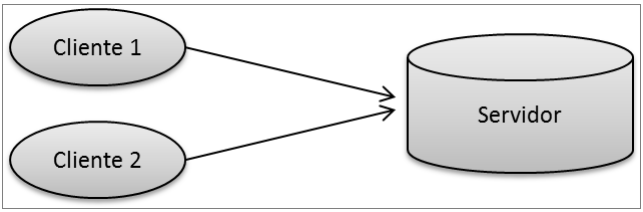


Figura 12 – Arquitetura cliente-servidor

3.3.1.2 Baseada em cluster

A arquitetura baseada em cluster (Figura 13) é uma arquitetura voltada para aplicações paralelas, que é a base para o surgimento das técnicas de distribuição de dados de um arquivo (*file-striping techniques*). Estas técnicas têm como função espalhar partes de um arquivo em múltiplos servidores, tornando possível o acesso de diferentes partes em paralelo. Para controlar a integridade dos dados sob múltiplos acessos paralelos, normalmente os SAD utilizam o mecanismo chamado de *lock*, que impossibilita qualquer alteração do conteúdo do dado por parte de outros processos, a menos que nenhum processo o esteja acessando.

Todavia, nem sempre é efetivo ou de interesse da aplicação paralela distribuir um arquivo em partes. De tal modo, alguns sistemas de arquivos permitem a configuração da distribuição de arquivos como todo, sem a necessidade de dividi-lo.

Os SAD que comportam estas técnicas de distribuição dos dados, permitindo o acesso paralelo entre os servidores de E/S, são chamados de sistemas de arquivos paralelos (SAP). Este tipo de arquitetura é eficiente quando contém um gerenciador para tratamentos de falhas e um gerenciador de coerência de dados. Normalmente, cada sistema de arquivo busca desenvolver o seu próprio gerenciador, de modo a servir de melhor maneira para o sistema em questão.

3.3.1.3 Simétrica

A arquitetura simétrica (Figura 14) é baseada na tecnologia *peer-to-peer*, em que a proposta utiliza como base uma tabela de hash distribuído (*Distributed Hash Table*) para a distribuição de dados, combi-

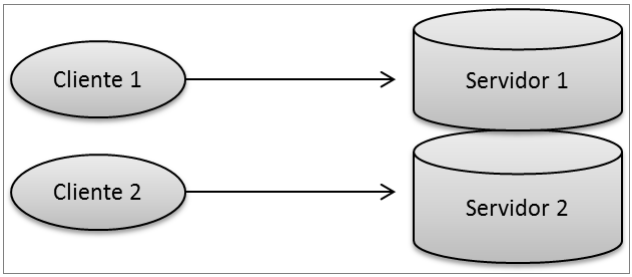


Figura 13 – Arquitetura baseada em cluster

nado com um mecanismo de pesquisa baseado em chaves. Além disso, esta arquitetura pode ser construída em cima de uma camada de armazenamento distribuído, ou todos os arquivos são armazenados localmente no nodo participante. Um exemplo de um sistema distribuído que utiliza esta arquitetura é o sistema *Apache Ivy*.

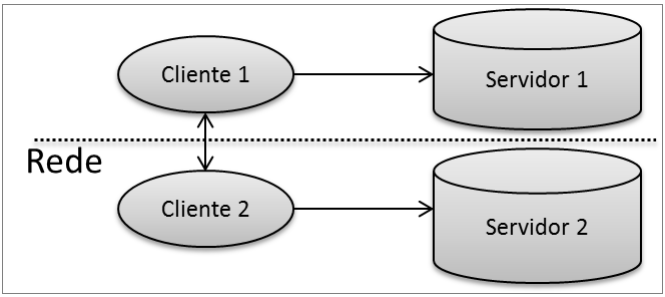


Figura 14 – Arquitetura simétrica

3.3.2 Componentes de SAD

Alguns conceitos básicos que se aplicam aos sistemas de arquivos distribuídos em geral têm como base objetivos fundamentais de fornecer ao cliente total transparência nos seus arquivos, não havendo necessidade de conhecer a localização física do arquivo e garantir disponibilidade. Para prover estas necessidades, os sistemas de arquivos distribuem as tarefas entre diferentes nodos. Esta divisão de tarefas

tende a dificultar o gerenciamento do sistema, contudo, torna o sistema mais consistente a fim de garantir um maior controle de falhas do sistema. É de importância saber até que ponto os mecanismos de consistências e sincronização influenciam no desempenho do sistema. Cada tarefa é delegada a um ou mais nodos, de modo que cada nodo pode aderir mais de um tipo de tarefa.

Basicamente, são atribuídos três tipos de tarefas:

- *Metadados.* Estes são nodos dos sistemas responsáveis por conter as informações sobre os arquivos. Estas informações tratam desde nomes, permissões, datas de acesso e principalmente as localizações dos blocos dos dados no sistema. A distribuição dos arquivos pode ser dada por blocos fixos enviados de maneira cíclica entre os nodos, ou distribuição dinâmica de acordo com o tipo do sistema ou de acordo com a necessidade da aplicação. São os primeiros nodos a serem consultados por um cliente, quando o servidor de metadados pode repassar o caminho do arquivo ou ao mapeamento dos dados.

- *Servidores de dados.* São aqueles que vão realizar as operações de entrada e saída dos blocos dos arquivos em disco. As operações são realizadas de acordo com o servidor de metadados do sistema, que este informa o tamanho do bloco de leitura e escrita, e a localização do bloco.

- *Clientes.* Os clientes realizam o acesso aos dados através de uma interface. No caso dos SAD, basicamente existe dois tipos de acesso, direto e indireto, ilustrados na Figura 15. No acesso direto, o cliente acessa os arquivos diretamente após consultar o servidor de metadados, que verifica a permissão e informa a localização dos blocos dos arquivos, para que o cliente acesse esses blocos. No acesso indireto, o cliente envia a operação ao servidor de metadados, que envia a operação para os servidores de dados que possuem os blocos do arquivo. Desta maneira, os sistemas com acesso indireto possuem um custo adicional de transmissão de dados e os metadados têm total controle sobre os acessos dos dados.

Independente do tipo de acesso, ambos trocam mensagens com o servidor de metadados, e com o aumento do número de requisições, pode tornar-se um gargalo de comunicação para o SAD. Alguns SAD como PVFS e Lustre, distribuem os servidores metadados com a finalidade de contornar casos de sobrecarga de requisições. Para que o

sistema funcione é necessário prevenir a falta de consistência de dados entre os metadados e é importante evitar processos desnecessários, como por exemplo, uma simples consulta de um cliente necessitando buscar informações dos dados em todos os metadados (KASSICK, 2010).

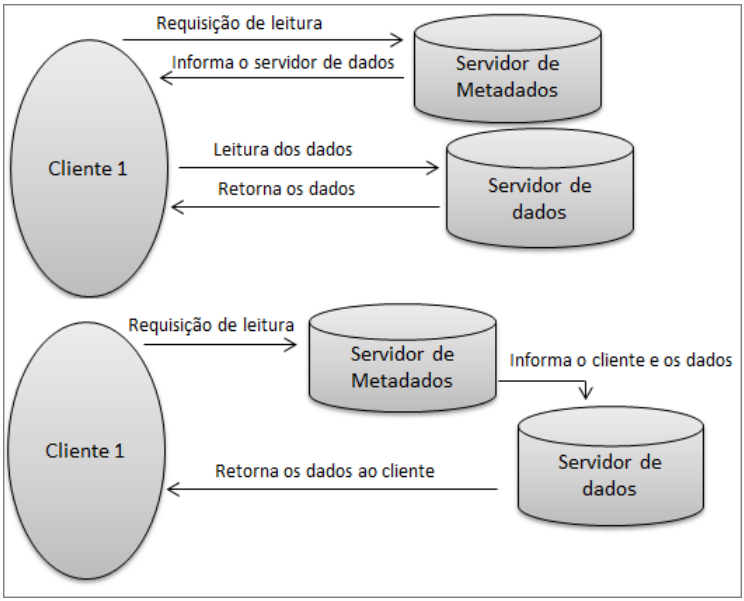


Figura 15 – Ilustração do acesso direto e indireto

3.4 EXEMPLOS DE SAD

3.4.1 *Network File System*

Quando o termo sistema de arquivos distribuídos (SAD) é citado, o primeiro sistema mais referenciado, comum e conhecido pelos usuários de UNIX é o NFS. O Network File System (NOWICKI, 1989) se tornou tão comum que é disponibilizado e suportado em quase todas as distribuições Linux de hoje em dia. Sua versão inicial foi desenvolvido pela Sun e liberado ao público em 19985, o que possibilitou que outras empresas e desenvolvedores designassem diferentes variações do sistema. Esta disponibilização do código repercutiu a integração do

protocolo NFS para diversos sistemas operacionais, inclusive o sistema operacional Windows.

O NFS possui 4 versões que são incompatíveis entre si, mas diferenciam em alguns aspectos. A versão 2 (NFSv2) e 3 (NFSv3) (PAWLOWSKI et al., 1994) aos poucos estão deixando de serem usadas e a versão 4 (NFSv4) (SHEPLER et al., 2003) começou a ser implementada a partir do kernel 2.6. Ambas as versões 2 e 3 não mantêm o estado das transações, permitindo uma implementação mais simples, sem necessidade de qualquer procedimento de recuperação. Porém, essa política gera problemas nas transações atômicas [3] de gerências de *locks*, por conseguinte, problemas no controle sobre acesso concorrente, problemas de consistência de memória cache e perdas de informações dos dados.

A partir da versão 3, foi alterado o mecanismo de controle de cache para possuir tamanho dinâmico, e a política de escrita, sendo que na versão anterior o arquivo só era escrito em disco quando fechado (write-on-close) ou somente gravado depois de um determinado tempo (delayed-write). Esse tempo de espera para realizar a E/S permite que, caso o arquivo seja modificado e depois apagado, nenhum dado será gravado em disco, evitando acessos desnecessários. Outro diferencial é o aumento do tamanho dos blocos de tráfego, que antes era limitado a 8KB, chegando a 56KB via UDP [4]. Na versão 3 foi implementado um cache de dados, de tal maneira que o servidor retorna o *status* do pedido ao cliente antes de realizar a operação de E/S, evitando a tomada de tempo de E/S encontrada na versão 2.

Por fim, outro diferencial entre essas duas versões é a questão da segurança. No NFSv2, o cliente era responsável pelo controle de acesso aos arquivos, sem nenhuma interferência por parte do servidor. No NFSv3, isso foi mudado, deixando a segurança a cargo do servidor, que utiliza o mesmo padrão de segurança de um sistema UNIX. Para ter esse controle, o servidor possui um serviço de informação criptografada de rede, que é usado para fornecer sincronização de usuários (uid) e grupos (guid) entre as máquinas na rede.

Na versão 4 do protocolo NFS (NFSv4) (SHEPLER et al., 2003) foram incorporadas novas funcionalidades e alterações, que permitiram várias vantagens em relação às versões anteriores. Uma das grandes diferenças das versões anteriores é a possibilidade de manter os estados das transações, podendo garantir atomicidade das operações e garantias

[3] Transações atômicas são operação que devem ser realizadas de forma única, evitando interrupimento durante a transação e garantindo a consistência dos dados

[4] Ambos as versões 2 e 3 utilizam conexão UDP, pois a van*tag*em é a diminuição o *overhead* provido por uma conexão TCP, ganhando mais desempenho e redes menos congestionadas.

de *locks* em arquivos pelos clientes.

Além disso, serviços de migração e replicação de arquivos foram adicionados a fim de aumentar a disponibilidade de acesso e segurança na perda de dados. No caso da migração, o sistema possui atributos que são consultados pelos clientes para indicar em qual servidor foram imigrados os dados. Estes atributos também podem ser utilizados para designar localizações alternativas de arquivos replicados, somente para leitura.

É importante observar que como o NFS possui uma arquitetura de acesso cliente-servidor, na qual diversos clientes possuem acesso a dados armazenados em um único servidor, este tipo de paradigma torna-se desvantajoso quando há muitos clientes acessando o sistema. Existem variações deste protocolo que visam a distribuição do servidor de dados, a fim de ganhar desempenho na leitura e escrita de dados paralelos. Como exemplos dessas variações temos NFSp, dNFSp, pNFS, Split-Server, NFS e NFS-CD. (KASSICK; BOITO; NAVAUX, 2011)

3.4.2 *Andrew File System*

O AFS (HOWARD; CENTER, 1988) foi um projeto iniciado pela Universidade Carniege-Mellon junto com a IBM em 1983. O objeto inicial foi projetar e implantar um sistema distribuído para um ambiente acadêmico com mais de 7000 máquinas, distribuídas entre os estudantes, professores e servidores, a fim de disponibilizar aos usuários aplicações, gerenciamento e compartilhamento de arquivos.

O seu desenvolvimento foi dividido em versões que foram tornando o sistema cada vez mais eficiente, aderindo diversos mecanismos de consistência de cache, de comunicação e infraestruturas dos servidores, resultando em um sistema mais escalável.

A sua arquitetura é parecida com NFS, possuindo um servidor de metadados, dados e clientes. No caso do AFS, os clientes são aqueles que vão manter os arquivos em caches e arquivos necessários para a inicialização da máquina, enquanto os servidores são encontrados a partir de qualquer cliente conectado na mesma rede. Toda informação do arquivo é mantida nos servidores, desde sua localização, bloqueios, até suas permissões. O cliente, quando necessita acessar um arquivo, precisa perguntar aos servidores de metadados a sua localização.

Um dos principais conceitos utilizados para amenizar problemas de escalabilidade no AFS é conceito de *callback*. O *Callback* é uma forma de controlar a consistência de dados dos clientes. Basicamente,

quando um cliente solicita acesso a um arquivo, o servidor de metadados envia uma cópia do arquivo e também uma *flag*, ambos mantidos em cache pelo cliente. Esta *flag* garante ao cliente que está utilizando a versão mais recente do arquivo. Caso o arquivo seja modificado, a *flag* fica inválida, tal que, quando o cliente for acessar novamente o arquivo, o gerenciador de cache envia novamente os blocos do arquivo modificado e uma nova *flag* atualizada. (AFS..., 1998).

A versão da *flag* só é atualizada quando o arquivo é modificado e salvo por um cliente, de tal maneira que o servidor de metadados é encarregado de atualizar o cache do arquivo e a *flag* em todos os clientes que estejam acessando o mesmo arquivo. Com esta forma de gerência de cache, o problema de escalabilidade é amenizado, devido às operações de leitura e escrita serem realizadas em cópias de arquivos. Porém, essa prática traz consequências aos clientes, que só terão acessos aos novos dados quando o outro cliente salvar o arquivo.

Os mecanismos de replicação são baseados no CODA, em que cada alteração do arquivo é propagada para todos os servidores de dados. A segurança e permissão de acesso são providos pelas aplicações Kerberos, que é um protocolo de autenticação na rede que utiliza chaves criptográficas para autenticação dos usuários.

3.4.3 CEPH

O sistema de arquivos CEPH(WEIL et al., 2006) foi desenvolvido na Universidade da Califórnia em 2006 e buscou desenvolver um sistema de arquivos mais eficiente, confiável e escalável. Os sistemas atuais que adotaram armazenamento orientado a objetos sofrem problemas de limitação de escalabilidade devido à pouca distribuição dos gerenciadores de metadados. O CEPH procura separar ao máximo os gerenciadores de metadados (MSD) dos servidores de dados (OSDs), substituindo tabelas de alocações por uma função aleatória de distribuição de dados, projetada para clusters heterogêneos e dinâmicos.

O CEPH utiliza uma arquitetura de cluster altamente adaptativo e distribuído de metadados que melhora drasticamente a escalabilidade do acesso aos metadados, e com ela, a escalabilidade de todo o sistema. Desenvolvido para sistema de larga escala, a arquitetura do CEPH é voltada para ambientes em petabytes, onde envolve milhares de computadores acessando paralelamente um mesmo arquivo e onde nodos falham o tempo inteiro. O CEPH visou contornar estes problemas através de 3 características.

- *Dissociação de dados e metadados.* Busca total divisão do gerenciamento de metadados com o armazenamento de dados. As consultas dos metadados são gerenciadas pelo servidor de metadados, enquanto as operações de entrada e saída são realizadas pelos OSDs. Os arquivos são espalhados em pré-determinados objetos, enquanto uma função de distribuição de dados chamado de CRUSH *(Controlled Replication Under Scalable Hashing)* , atribui objetos a um dispositivo de armazenamento.

- *Gerenciador dinâmico de metadados distribuídos.* Utiliza uma árvore hierárquica para o gerenciamento de mais de cem servidores de metadados, na qual cada um mantém uma partição da árvore, improvisando atualizações e consultas via padrões de acesso concorrente.

- *Armazenamento de objetos distribuídos atômicos e confiáveis (RADOS).* O aglomerado de OSD é responsável pela replicação, detecção de falhas e restauração dos dados armazenados, além de prover suas principais funções de E/S.

Desta maneira, a arquitetura é um pouco diferente da arquitetura padrão, sendo dividida em três componentes: os dispositivos de armazenamento de objetos (OSD) , que em coletivo armazenam os dados e os metadados; gerenciador de metadados (MDS) , que gerencia o arquivos e diretórios, permissões, coerência e consistência do cluster; e o cliente que utiliza uma interface *POSIX* de acesso ao sistema.

O diferencial do CEPH está no seu mapa de hierarquia de objetos. Quando um cliente necessita acessar um arquivo, primeiramente deve mandar um chamado a um MDS. O gerenciador localiza as partes do arquivo nos OSD no mapa de arquivo em questão para traduzir o nome do arquivo em um inode (número identificador de nodos), que retorna toda informação sobre a forma de distribuição do arquivo usado para mapear os arquivos em objetos. Os arquivos e diretórios são bem pequenos, sendo o diretório composto somente pelo nome e os inodes dos arquivos pertencentes ao diretório. Diferente de outros sistemas de arquivos, não há necessidade de alocar os metadados, pois os objetos são construídos a partir do número inode e distribuídos via função *CRUSH*. Isto simplifica a carga de trabalho de metadados permitindo aos MDS gerenciar eficientemente grande conjunto de arquivo de qualquer tamanho.

No CEPH, um cliente consegue localizar todos os objetos do arquivo somente com o inode, layout e o tamanho do arquivo, podendo

localizá-lo de qualquer componente (OSD ou MDS). Isso é permitido devido à função de distribuição *CRUSH*, que traduzindo significa: replicação controlada sob hashing escaláveis.

A distribuição dos dados via *CRUSH* é representado na Figura 16 e funciona da seguinte maneira. Primeiro CEPH mapeia os objetos em grupos de localização chamados de grupos de posicionamento (PG ou *placement groups*). Os grupos de posicionamento são atribuídos aos OSD usando *CRUSH*. O *CRUSH* é uma função pseudoaleatória de distribuição de dados que mapeia eficientemente cada PG para uma lista ordenada de OSD com a finalidade de armazenar réplicas de objetos. Para localizar qualquer objeto, *CRUSH* requer apenas o PG e um mapa de cluster de OSD (o mapa é uma descrição compacta hierárquica dos dispositivos que compõem o cluster de armazenamento). Desta maneira, é possível que qualquer componente calcule de forma independente a localização de qualquer objeto no sistema, sem qualquer troca de informação com os dos metadados

A vantagem desta distribuição via *CRUSH* é tentar manter balanceamento de carga entre os nodos, evitando a criação de *hot-spots* e recursos ociosos. Esta função difere dos sistemas convencionais em que a colocação de dados não depende de qualquer bloco ou uma lista de metadados dos objetos, permitindo que qualquer parte do sistema seja o cliente, servidor de metadados ou próprio OSD localize qualquer objeto.

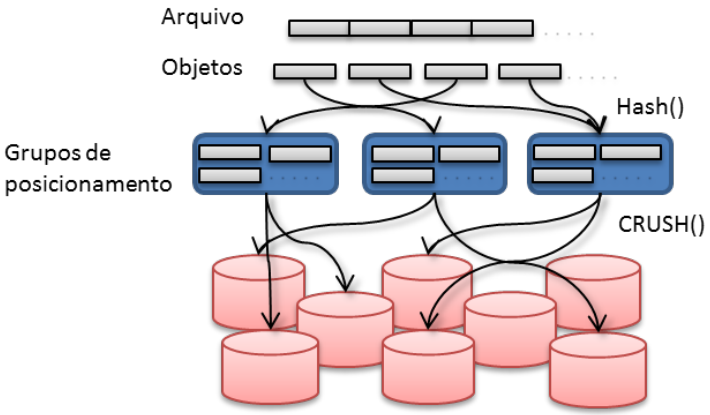


Figura 16 – Ilustração da distribuição dos dados no sistema CEPH

Os módulos de acesso do cliente podem ser tanto através de um módulo kernel quanto através de uma biblioteca acessada diretamente por uma aplicação. Em ambos os casos utilizam a semântica *POSIX*. A semântica *POSIX* exige que a leitura reflita quaisquer dados gravados anteriormente, e que escrita seja atômica. No CEPH, quando um arquivo é aberto por múltiplos clientes com múltiplas funções de escrita, o MDS irá anular qualquer cache de leitura emitido anteriormente e criar *buffer* de escritas, forçando E/S do cliente para que o arquivo seja sincronizado. Ou seja, cada aplicação de leitura ou operação de escrita irá bloquear até que este seja reconhecido pelos OSDs, efetivamente colocando a carga de atualização da serialização e da sincronização com os OSD. Porém, a sincronização da E/S possui baixo desempenho. Desta maneira o CEPH provê uma opção mais atenuada através de uma *flag* chamada *O_Lazy*, porém não garante a consistência da sincronização. Com a *flag O_Lazy*, a própria aplicação gerencia a consistência de suas operações, permitindo que ela crie os *buffer* de escritas e a cache das leituras.

3.4.4 *Hadoop File System*

O sistema de arquivos HDFS (*Hadoop File System*) (SHVACHKO et al., 2010), foi um projeto organizado pela Apache Foundation, que buscou desenvolver um sistema de arquivos distribuídos, confiável e escalável, a fim de suprir as necessidades de um sistema de armazenamento largamente escalável e de grande vazão. Desta maneira, HDFS é projetado para aplicações que gerencia grandes quantidades de dados, priorizando a alta vazão de dados e baixa latência. Portanto para aproveitar toda característica do sistema, devem-se utilizar aplicações que acessam grandes quantidades de dados.

A arquitetura deste sistema é composto pelos elementos básicos de um SAD (servidor de metadados, servidor de dados e cliente). O servidor de dados é centralizado e chamado de *NameNode*, que possui diversos servidores de dados chamados *DataNode*, e a conexão de dados entre os elementos é via protocolo base TCP.

O grande diferencial do HDFS é garantir a confiabilidade dos dados. Para isso, o sistema sempre replica os dados entre os servidores a fim de garantir a tolerância a falhas. Os dados são divididos em blocos, que são distribuídos entre os *DataNode* e replicados automaticamente. As políticas de replicação visam maximizar a vazão e ao mesmo tempo evitar perda de dados por falhas tanto em discos ou *DataNodes*

individuais, até indisponibilidades de clusters.

O mecanismo de acesso a dados consiste na ideia de write-once-read-many, em que não é permitido que mais de um cliente escreva em um mesmo arquivo. Assim, os mecanismos de consistência dos dados são simplificados. Porém, no caso de leitura, os dados podem ser lidos diversas vezes concorrentemente por uma aplicação paralela, uma vez que estes dados são replicados em outros nodos. Desta maneira, com a replicação dos dados, o mecanismo de leitura dos blocos tende a localizar os dados mais próximos em suas buscas. Outro atributo importante do HDFS é que este implementa o mecanismo MapReduce do sistema *Google File System*. MapReduce é um modelo de programação paralelo e uma implementação associada para o processamento e geração de grandes quantidade de dados. Este algoritmo se divide em dois passos

- *Mapa*. O *NameNode* recebe o arquivo, divide-o em pequenos arquivos e distribui para os *DataNodes*, que repetem o mesmo processo, conduzindo a uma estrutura de árvore de múltiplos níveis. Os últimos *DataNodes* processam o menor arquivo, e repassa a resposta para o *DataNode* "pai".

- *Reduzir*. O *NameNode*, então, recolhe as respostas para todas as sub-rotinas e combina-as de alguma forma para formar o mapa de acesso.

Como pode notar, MapReduce automaticamente paraleliza e executa o programa em um grande aglomerado de máquinas. O sistema de execução cuida de detalhes do particionamento dos dados de entrada, a manipulação de falhas e a gestão de comunicação entre máquinas.

Assim como o *Google File System*, HDFS é um sistema de arquivo voltado para serviços de internet, em que o acesso de um cliente é mais comum nesse cenário. Embora o acesso concorrente seja um cenário buscado pela maioria dos sistemas de arquivos paralelos, GoogleFS e HDFS não oferecem otimização para este caso de uso, suportando somente acesso de um único cliente para escrita em arquivo.

3.5 EXEMPLOS DE SAP

3.5.1 *NFSp e dNFSp*

O sistemas de arquivos NFSp (LOMBARD; DENNEULIN, 2002) e o dNFSp (ÁVILA, 2005) são sistemas derivados do sistema NFS, e focam

em contornar problemas encontrados na arquitetura anterior. Nesta arquitetura base, existe somente um servidor de metadados e vários servidores de dados que, com o aumento de número de requisições dos clientes, causa um *overhead* no servidor de metadados, diminuindo o desempenho no sistema.

O NFSp é um SAP cujo objetivo é fornecer maior desempenho no acesso aos dados através do mesmo protocolo NFS, não havendo nenhuma modificação no lado do cliente. O projeto foi iniciado no Laboratoire Informatique et Distribuition (ID) de Grenoble, França, no ano de 2000, como parte de um projeto de um cluster de grande porte. Tendo como motivação a falta de uma solução que contemplasse as características desejadas pelo grupo, decidiram fazer uma implementação sob o protocolo NFS.

O NFSp utiliza a versão do protocolo NFSv2 e a sua arquitetura é similar a sistemas de arquivos paralelos, sendo dividido em servidores de metadados e servidores de dados, ou usualmente chamado, IOD (*daemon* de E/S) . Estes servidores possuem os *daemon* de leitura e escrita que são responsáveis pela realização das funções E/S no disco. Esta técnica é bastante difundida em SAP, e apresenta algumas vantagens como: fácil replicação de dados, aproveitamento de espaços livres nos nodos e, principalmente, aumento da vazão de dados devido à operação paralela. Similar ao NFS, os clientes enviam requisição baseados em clusters, via acesso indireto, ou seja, quando um servidor de metadados recebe uma requisição do cliente, o servidor consulta o servidor de metadados adequado e repassa a requisição ao IOD correspondente. Este por sua vez, obtém o bloco desejado e transmite diretamente ao cliente.

O distributed NFSp (dNFSp) apresenta um modelo baseado no NFSp, e segue o mesmo paradigma de manter os objetivos originais, para não implicar em grandes mudanças na instalação ou acesso ao sistema. A grande diferença foca na distribuição do servidor de metadados, que implica escalar os sistemas com possíveis sincronizações entre os servidores de metadados a fim de garantir a leitura e escrita concorrente. O projeto foi iniciado na Universidade Federal do Rio Grande do Sul em 2003, sendo o primeiro período foi dedicado à investigação de desempenho de sistemas de arquivos.

A Figura 17 ilustra a arquitetura proposta para o dNFSp. O cliente acessa o sistema utilizando o protocolo padrão do NFSv2, envia a requisição a algum servidor de metadados disponível, que sincroniza com os outros servidores e realiza a requisição nos IODs configurados. Assim como o NFSp, os IOD respondem as requisições diretamente aos clientes. A divisão de dados é fixa entre um conjunto pré-configurado

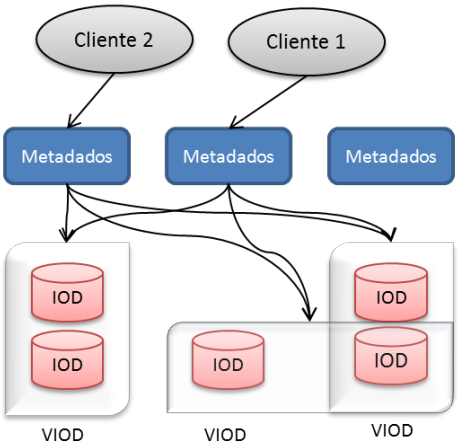


Figura 17 – Ilustração da distribuição dos dados no sistema dNFSp

de IOD, delimitando a mudança dinâmica da arquitetura do sistema.

O dNFSp utiliza a abstração de Virtual IODs (VIOD) para replicação de arquivos. Cada VIOD contém um conjunto de IODs, que podem ser adicionados ou removidos, tal que um IOD pode também pertencer a outro VIOD, para visar o espelhamento em um número mínimo de IODs. A replicação é feita pelo processo chamado de *store-and-foward*, em que após um IOD receber o bloco a ser escrito pelo servidor de metadados, este armazena o bloco e o reenvia a outro IOD. Este processo se repete até todos os IODs possuam os blocos e o último nodo envie uma mensagem ao servidor confirmando a gravação dos dados.

O processo garante a disponibilidade dos dados em todos os IODs de um VIOD. Desse modo, quando o servidor de dados recebe a requisição de leitura de um arquivo, é acionado cada IOD que contém a réplica dos dados, a fim de aumentar o desempenho da função de leitura. Outra ferramenta importante que foi incorporada é o gerenciador de tabelas de VIODs. Esta tabela é mantida junto aos servidores de metadados, informando que IODs atendem as requisições de quais VIODs. A ferramenta é também um *daemon* executado junto aos servidores de metadados, sendo que através dela são recebidos os comandos de modificação do sistema de armazenamento. Cada servidor terá uma tabela e um gerenciador, onde na inicialização do sistema é escolhido aleato-

riamente um gerenciador para receber os comandos de reconfiguração e repassá-los ao outros *daemon*.

É também de responsabilidade do gerenciador inicial verificar o funcionamento dos servidores de dados, pois no caso de que um dos IOD deixe de responder ao servidor, este é descartado da configuração do sistema. Por fim, uma outra característica interessante implantada no dNFSp é a possibilidade de adaptar o sistema para dedicar alguns IODs a uma aplicação específica que requer grande capacidade de armazenamento e alta vazão na transferência de dados do sistema. Esses servidores são chamados de Servidores de Aplicação (AppIODs) e dependendo da grandeza da aplicacão, pode apresentar um ganho significativo.

3.5.2 *Parallel Virtual File System*

O *Parallel Virtual File System* (PVFS)(CARNS et al., 2000) foi desenvolvido pela primeira vez em 1993 por Walt Ligon e Blumer Eric como parte de um projeto da NASA para estudar os padrões de E/S de programas paralelos. A versão inicial foi baseada no sistema paralelo desenvolvido pela IBM, chamada Vesta. Em 1994, a versão 1 do PVFS foi toda reescrita por Rob Ross, na qual passou a utilizar o protocolo TCP/IP para comunicação. Uma das características herdadas da Vesta é a distribuição de arquivos em vários servidores, consentindo diferentes tipos de padrão de acessos paralelos. Essa característica é que torna esses sistemas tão eficientes em termos de desempenho, pois a leitura e escrita de dados são de forma paralela. O PVFS, diferente do Vesta, foi desenvolvido focando na escalonamento das E/S de discos quando vários clientes acessam ao mesmo tempo um mesmo arquivo.

Hoje em dia, o PVFS é mantido por uma companhia chamada Omnibond, a qual provê desenvolvimento de aplicações opensource e oferece suporte a uma versão mais atual do PVFS chamada OrangeFS (OMNIBOND, 2011). As ultimas versões lançadas pela companhia procura melhorar o serviço de metadados e deficiências em desempenho em arquivos pequenos demonstrado no trabalho de Carns (CARNS et al., 2000).

O PVFS permite diferentes configurações em sua arquitetura, desde definir quantos nodos possuirá o sistema, até o número de blocos em que um arquivo será dividido. É constituído pelos três componentes básico de um SAP: servidor de metadados, servidor de dados e cliente. O acesso a esses blocos (ilustrado na Figura 18), inicia-se acessando

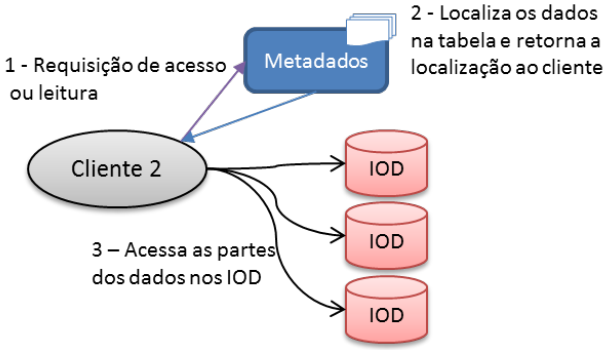


Figura 18 – Ilustração do acesso paralelo dos dados no PVFS

um servidor de metadados, sendo que que no caso do PVFS o servidor retorna uma tabela de localização dos blocos nos diferentes nodos, realizando o acesso em paralelo aos nodos que contém os blocos pertencentes ao arquivo. Este tipo de abordagem traz como consequência maior utilizacão banda da rede, podendo prejudicar o desempenho do sistema.

A Figura 18 ilustra a forma de acesso paralelo, como também a tabela de localização de dados que o servidor de metadados disponibiliza. Nesta tabela são mapeados os diretórios nos quais se encontram os arquivos nos nodos (IODs), a quantidade de blocos que cada nodo contém e a sua localização no arquivo. Dessa forma, o cliente consegue localizar os caminhos dos blocos de arquivos e construir o arquivo por completo. Este processo também é reversível, sendo que ao gravar um arquivo, este é dividido de acordo com a configuração do sistema, gerando outra tabela. Além disso, nos nodos E/S, o PVFS utiliza *daemon* que ficam executando enquanto o sistema funciona, para controlar os acessos ao disco para armazenamento e leitura de dados

A Figura 19 ilustra a distribuição de dados não contínuos do PVFS. Em sua configuração padrão, qualquer arquivo criado é dividida em blocos de tamanho constante (64 KB por padrão) que são distribuídos através da função de escalonamento *round-robin* entre os servidores de dados. O servidor que armazena o primeiro bloco é escolhido aleatoriamente durante a criação do arquivo. O tamanho do bloco pode ser configurado durante a criação do arquivo para combinar com os padrões de acesso a dados de aplicações específicas. Outros

algoritmos de distribuição de dados também podem ser aplicados a arquivos específicos. Uma das características do PVFS é a sua maneira de distribuir os dados de forma não contínua (*noncontiguous*), facilitando a utilização de múltiplos acessos em diversas regiões. No entanto, o número de chamadas de E/S pode aumentar com o número de servidores de dados, aumentando a latência de E/S no sistema.

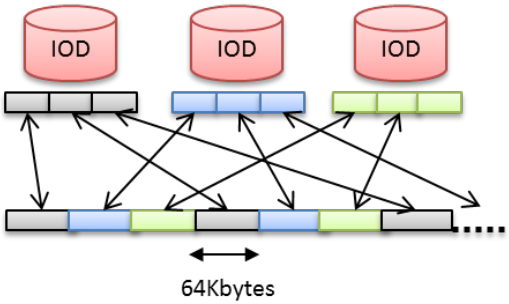


Figura 19 – Distribuição de um arquivo em dados não contínuos

O acesso aos arquivos no sistema PVFS pode ser realizado via 3 alternativas: através de uma API disponibilizada pela aplicação, na qual os comandos assemelham-se com os de sistemas UNIX; através da compilação do PVFS como módulo do kernel, permitindo que os comandos POSIX acessem o sistema; ou por uma API de alto desempenho disponível em bibliotecas como o ROMIO, permitindo integrar aplicações paralelas de alto desempenho em E/S através do MPI. O PVFS possui uma segunda versão (PVFS2), na qual se destaca pela distribuição dos servidores de metadados e pela eliminação dos *locks* de acesso aos arquivos. Esta medida tem o objetivo de permitir que o sistema escale uma grande quantidade de clientes com pouco *overhead* de gerenciamento, delegando às aplicações eventuais políticas de consistência de dados necessários devido ao acesso paralelo.

O dinamismo de nodos, a segurança e o mecanismo de tolerância a falhas deixam a desejar no sistema do PVFS. Quando se configura o sistema, deve-se definir quantos nodos o sistema terá, delimitando acréscimos de novos nodos ao sistema, tendo que gerar uma nova configuração do sistema. Este problema repete-se quando uns dos nodos falham, impedindo o funcionamento do sistema. Existem trabalhos que focam nesses problemas como o trabalho de Zhu *et al.* (ZHU et al., 2003) e Nieto *et al.* (NIETO et al., 2010), que visam explorar mecanismos de

tolerância a falhas, sendo que ambos utilizam a replicação de dados. O trabalho de (KUHN; KUNKEL; LUDWIG, 2009) procura contornar acessos a servidores de metadados, uma vez que a arquitetura contém muitos metadados, podendo tornar o acesso aos dados muito lento.

3.5.3 *Lustre*

O sistema de arquivos Lustre foi iniciado em 1999 na Universidade de Carniege Mellon de Pittsburgh, em um projeto de pesquisa de Peter Braam. Peter, que fundou uma companhia chamada *Cluster File Systems*, desenvolveu o primeiro Lustre 1.0, lançado em 2003. Em 2007, a companhia foi comprada pela Sun, que hoje pertence à Oracle, sendo que o desenvolvimento foi estagnado na versão 1.8. No final de 2010, os dois principais desenvolvedores do Lustre deixaram a Oracle para trabalharem na Whamcloud, que em 2011 teve um contrato para continuar o desenvolvimento do Lustre. Hoje o Lustre se encontra na versão 2.2. (WIKIPEDIA, 2012)

No primeiro contato com o Lustre pode-se notar um grande diferencial no número de componentes comparado com sistemas SAP em geral. O *Management Sever* (MGS) é responsável por armazenar a configuração de todo o sistema de arquivo e disponibilizar essas informações a outros componentes. Este normalmente é alocado na mesma máquina de um servidor de metadados (MDS), como mostra a Figura 20. O servidor de metadados é subdivido em dois componentes: *Metadata Server* (MDS) e *Metadata Targets* (MDT). (SCHWAN, 2003)

O MDS gerencia o armazenamento de metadados em um ou mais MDTs e controla os acessos em cada MDT. Portanto, os MDTs são aqueles que armazenam os metadados e são acoplados pelos menos a um ou mais MDS. Essa subdivisão ocorre também para os servidores de dados. Neste caso, temos os Object Storage Servers (OSS), que fornece os serviços de E/S e controla as requisições nos *Object Storage Target* (OST) locais.

Em sistemas de arquivos tradicionais do UNIX, um nodo de estrutura de dados contém informações básicas sobre cada arquivo, como a localização dos dados contidos do arquivo é armazenada. O sistema de arquivo Lustre também usa nodo, mas o nodo em um MDTS aponta para um ou mais objetos OST associados com o objeto em vez de blocos de dados. Esses objetos são implementados como arquivos nos OSTs. Quando um cliente abre um arquivo, a operação de abertura transfere um conjunto de ponteiros de objetos e sua disposição a partir das MDS

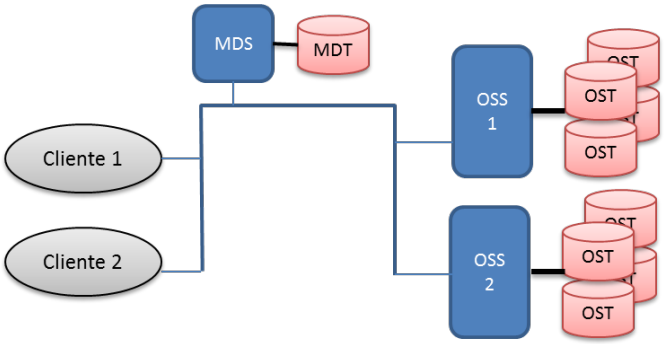


Figura 20 – Componentes básicos do Lustre

para o cliente, de modo que o cliente pode interagir diretamente com o nó OSS onde o objeto está armazenado, permitindo que o cliente realize E/S no arquivo sem comunicação com o MDS.

Os clientes Lustre acessam o sistema via operações *POSIX*, uma vez que este seja acessível ao cliente. Estes podem manter uma cache de metadados, a fim de evitar constantes acessos ao MDS, ficando a cargo do servidor controlar a validade deste cache.

Como foi dito, os OST armazenam objetos dos arquivos em cada componente. Um objeto representa uma unidade de armazenamento e pode representar desde um arquivo por inteiro ou uma fração dele. Quando armazenado em vários objetos, assim como o PVFS, estes são distribuídos pelo método *round-robin*, em que cada objeto é gerenciando somente por um OST. Por padrão, o Lustre divide os arquivos em dois objetos, distribuindo aleatoriamente entre os OSTs, podendo ser configurado o número de objetos por diretório e também por arquivo.

O sistema foi desenvolvido para ser dinâmico, permitindo a entrada e saídas de componentes, sem que comprometa o sistema. Para adicionar um novo OSS e OST, basta configurar o sistema para conectar ao MSG, que este irá orientar o resto do sistema desse novo componente. Como pode-se observar, é importante manter uma réplica do MSG pois, uma vez que este falhe, compromete todo o sistema. O Lustre para fornecer um alto controle de falhas utiliza recursos de *lock*, de gerenciamento e monitoramento dos componentes dos sistema. O recurso de *lock* protege o acesso simultâneo de múltiplos nodos, evitando a incompatibilidade dos dados. Os recursos de gerenciamento iniciam e cessam os recursos do Lustre, preservando o estado do cluster, e o

recursos de monitoramento têm a função de verificar a integridade dos recursos hardware e softwares.

Além do dinamismo, configurações de eventuais heterogeneidades são disponibilizadas nesse sistema. Através de servidores OSS, é possível configurar características a determinados OST, como por exemplos, especificar a interconexão dos nós, a velocidade de acesso dos discos, especificar a demanda em diferentes OST, como por exemplo, OST que possui discos maiores ou mais rápidos.

Para finalizar, a comunicação no Lustre é realizada pelo seu próprio protocolo de rede Lustre Networking (LNET), que provê suporte para diferentes infraestruturas de rede através de drivers chamados *Lustre Network Drivers* (LNDs), tornando o Lustre bastante adaptável a diversas tecnologias e arquiteturas.(BRAAM, 2007)

3.5.4 *Fraunhofer File System*

O sistema de arquivos paralelo *Fraunhofer File System* (FhGFS) (FS, 2012) foi desenvolvido para Linux, pelo centro de competência Fraunhofer na Alemanha, e foi projetado para lidar com problemas de escalabilidade e disponibilidade dos dados. A sua documentação é limitada para o meio acadêmico, dificultando projetos de pesquisas voltadas a este sistema.

A sua configuração é bem simples comparada com sistemas de arquivos paralelos convencionais, não requerendo configurações de sistemas de gerenciamento de permissões, segurança etc. A Figura 21 ilustra a arquitetura provida pelo FHGFS, tal que a sua estrutura segue um SAP convencional, dividindo-se em três componentes básicos: clientes, servidores de metadados e servidores de dados. Além dos três componentes básicos ,o FHGFS possui dois componentes adicionais, sendo um *daemon* de gerenciamento, que disponibiliza informações para os componentes básicos, e o serviço de administração e monitoramento, que disponibiliza uma interface Web para instalação e monitoramento do sistema.

O sistema foi projetado para ser um sistema fácil de instalar e utilizar, não requerendo de uma pessoa muito especializada em UNIX, como é encontrado em outros sistemas de arquivos. Desse modo, uma das maneiras de instalar o sistema é via uma interface gráfica amigável desenvolvida em JAVA, que com poucas configurações constrói o ambiente SAP. A sua instalação é baseada no código-fonte do kernal do sistema, possibilitando facil integração com diversos tipos de distribuições

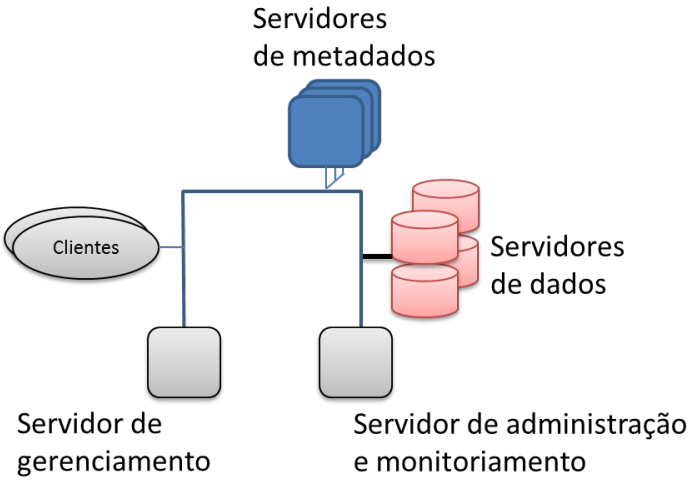


Figura 21 – Ilustração da arquitetura do FHGFS

Linux. É um sistema flexível, permitindo adicionar novos nodos sem precisar reconfigurar todo o sistema. Este sistema também disponibiliza uma interface de monitoramento e administração. A facilidade também provida pelo sistema ser totalmente dinâmico. Ao conectar qualquer componente no mesmo sistema, ou seja, onde o *daemon* de gerenciamento é comum, o componente registra os nós correspondentes e seus serviços no *daemon* de gerenciamento FhGFS automaticamente.

O FHGFS separa os metadados e conteúdo do arquivo, e disponibiliza acesso direto aos servidores de dados podendo realizar E/S com vários servidores simultaneamente, disponibilizando as aplicações realizarem acesso paralelos aos dados. A sincronização dos dados é realizada imediatamente a cada alteração de arquivos, descartando a utilização de cache de arquivo. Esta característica é eficiente principalmente para acessos paralelos, porém, como já foi comentando, demanda maior comunicação de rede.

A sua distribuição é de blocos de dados e pode ser configurada por diretório ou por arquivo. Desta maneira, cada diretório tem uma configuração padrão específica de faixa, que será derivada para novos subdiretórios e aplicada a qualquer arquivo criado no interior deste diretório. Existem dois parâmetros que podem ser configurados para cada diretório ou arquivo: o número de nós desejado para o armazenamento

dos dados, e o tamanho do bloco para cada faixa de arquivo.

A distribuição dos metadados definida por diretório significa também que cada diretório em seu sistema de arquivos pode ser gerenciado por um servidor de metadados diferente. Quando um novo diretório é criado, o sistema escolhe automaticamente um dos servidores de metadados disponíveis para ser responsável pelos arquivos dentro deste diretório. Novos subdiretórios deste diretório podem ser atribuídos a outros servidores de metadados, de modo que a carga é equilibrada em todos os servidores de metadados disponíveis. Há um nodo metadados que é representado como raiz especial, que serve como coordenador inicial e como um guia para o sistema e para os clientes. Esse nodo é eleito pelo *daemon* de gerenciamento somente durante a inicialização do sistema.

3.6 ANÁLISE DOS SAD

No capítulo anterior, foi levantado um estudo sobre diferentes tipos de sistemas de arquivos encontrado na literatura. Neste capitulo foram selecionados quatro desses sistemas de arquivos e discutidos suas principais diferenças e dificuldades de utilização.

3.6.1 Lustre

O sistema Lustre é um SAD que requer um conhecimento avançado em sua instalação, configuração e manutenção do sistema. É instalado via linha de comando e possui dependências de bibliotecas do sistema para que seja compilado. O resultado é um novo kernel com as API do Lustre. Desta maneira, torna-se um sistema mais integrado com o Sistema Operacional (SO), deixando de ser somente uma aplicação funcionando sob um SO. Durante a pesquisa, somente com o CentOS que foi possível realizar a instalação com êxito, devido esta distribuição disponibilizar pacotes pré-compilados que facilitaram a compilação. No Debian é necessário compilar o código fonte, que caso não tenha exatamente as versões das bibliotecas de dependências, o código não é compilado.

O sistema de arquivo Lustre tem como principais características:

- *Distribuição dos dados.* Os dados são encapsulados em um ou mais objetos (dependendo da configuração do sistema) e podem ser distribuídos de forma alinhada ou não, e disponibiliza diversas

configurações (tamanho do objeto, quantidade de nodos e etc.) de acordo com o interesse do usuário.

- *Componentes.* É utilizado um número maior de componentes no sistema do que encontrado na grande maioria dos SAD. Possui seis tipos de componentes que são responsáveis pela disponibilização dos dados no sistema, a fim de garantir as características necessárias de um sistema SAD. A diferença é a separação de componentes de gerenciamento dos componentes de armazenamento, resultando no adicional de quatro componentes; dois para metadados (MDS e MDT) e dois para dados (OSS e OST).

- *Acesso concorrente.* O Lustre, na sua configuração padrão, utiliza *lock* para controlar a integridade dos dados, dificultando a escrita concorrente.

3.6.2 PVFS

O PVFS é um sistema mais simples, que diferente do Lustre, é executado como uma aplicação no SO. A sua instalação é complexa devido as quantidade de dependências necessárias para compilar o código fonte, há necessidade de conhecer suas *flags* de compilação para suportar tecnologias de alto desempenho (p. e. ROMIO). Durante o inicio da pesquisa, buscou-se utilizar o PVFS devido sua forte integração com a interface ROMIO.

O sistema de arquivo PVFS tem como principais características:

- *Distribuição dos dados.* A distribuição é feita em blocos de arquivos de 64 Kbytes, podendo ser configurável, através do método *round-robin* em cada servidor de dados. Os dados são mapeados em uma tabela de localização de blocos, utilizado pelos clientes para a localização e acesso aos blocos.

- *Componentes.* Assim como a maioria dos sistemas, o PVFS possui os três componentes básico de um SAD (servidor de dados, servidor de metadados e o cliente).

- *Acesso concorrente.* O grande diferencial do PVFS é a sua eliminação dos *locks*, permitindo a escrita concorrente dos dados e delimitando a aplicação concorrente o controle de integridade dos dados. Possui também suporte a API do ROMIO, possibilitando aplicar sob o sistema diversas técnicas de E/S paralela.

3.6.3 CEPH

CEPH é um SAD recente e em desenvolvimento, que vem com o objeto de aplicar técnicas mais avançadas de distribuição de dados. A sua instalação também é complexa e deve seguir rigidamente as documentações online. A sua vantagem é a abrangencia da documentação para diversas distribuições Linux. A configuração é delicada, devido a grande quantidade de dados configuráveis.

O sistema de arquivo CEPH tem como principais características:

- *Distribuição dos dados.* Assim como o Lustre, os dados são encapsulados em objetos e a distribuição é realizada via função CRUSH. A função CRUSH replica os dados ao armazena-los e possibilita acessar qualquer dado através de qualquer componente CEPH.
- *Componentes.* O CEPH possui somente os três componentes básico, no entanto, o seu desenvolvimento buscou maior independencia na divisão de tarefas entre eles, que possibilitou a característica de consulta de dados em qualquer componente.
- *Acesso concorrente.* Utiliza *lock* para controlar acesso concorrente dos dados, porém possibilita utilizar a *flag O_Lazy* que habilita o controle de consistência dos dados por parte da aplicação.

3.6.4 FHGFS

O sistema desenvolvido pelo centro de competencia da Alemanhã é semelhante ao PVFS quando comparado com o funcionamento do sistema. É um sistema simple, com um modelo de instalação prático e eficiente via uma interface gráfica em que não requer muitos parâmetros de configuração, tornando-se bastante interessante para usuários iniciantes. Porém, a sua documentação para usuários avançados e pesquisadores é bastante limitado (indisponível), interferindo a sua utilização no meio acadêmica.

O sistema de arquivo FHGFS tem como principais características:

- *Distribuição dos dados.* A distribuição é feita em blocos de arquivos e pode ser configuravél diferentemente em diretórios e arquivos. A sincronização dos dados entre os servidores de dados é imediata, garantindo consistencia de dados entre os servidores.
- *Componentes.* Além dos três componentes básicos de um SAD, o FHGFS possui dois componentes visados principalmente para o

gerenciamento e monitoriamento do ambiente, via uma interface Web.

- *Acesso concorrente.* Apesar da falta de documentação, o comportamento do acesso concorrente foi similar ao PVFS, não utiliza *locks* para acesso concorrente e possui suporte à API do ROMIO.

4 PROPOSTA DA ARQUITETURA

4.1 IMPLEMENTAÇÃO

Neste capítulo é apresentada a proposta de arquitetura, abrangendo a descrição dos componentes utilizados. A primeira seção é a descrição da abordagem desenvolvida anteriormente, especificando as mudanças realizadas e as funções implementadas. Em seguida, é apresentada a abordagem projetada para este trabalho, focando nas principais mudanças das implementações.

4.1.1 *H5wrap*

A primeira mudança que foi necessária no servidor CyclopsDCM-Server para que comportasse o armazenamento em HDF5, foi criar uma interface que utiliza-se funções providas pela API do HDF5. A segunda mudança foi fundamentar uma estrutura hierárquica para os dados DICOM. Para transformar os dados em uma estrutura HDF, foi criada uma interface denominada H5Wrap, que basicamente encapsula os dados de uma imagem DICOM em uma estrutura hierárquica e vice-versa. Esta estrutura apresentada na Figura 22 foi introduzida no trabalho de Macedo *et al.* (MACEDO et al., 2009), em que buscou seguir a estrutura hierárquica de um arquivo DICOM demonstrada na Figura 3. Essa mesma estrutura pode ser encontrada nas tabelas relacionais do bancos de dados utilizadas no servidor original.

Diferente da estrutura hierárquica do DICOM, os atributos hospital e paciente estão contidos na camada Estudo. O motivo da utilização somente do modelo de estudo como raiz do nível de busca é baseado na premissa de que os grupos do HDF5 devem ter nomes únicos para identificação global, evitando a ocorrência de que pacientes com mesmo nome interfiram na recuperação de dados que não lhe pertence. Outro fator que é utilizado para identificação de cada camada é o número de identificação (ID), e cada camada da imagem DICOM possui um. Portanto, a estrutura HDF é dividida em três camadas: Estudo, Séries e Imagem.

Como se pode observar na Figura 20, os grupos estão representados em elipses, que contêm um ou mais grupos do próximo nível e *dataset*, podendo conter vários ou nenhum *dataset* no grupo. Logo, abaixo do grupo raiz ("/"), há o nível de estudo, que além de conter os

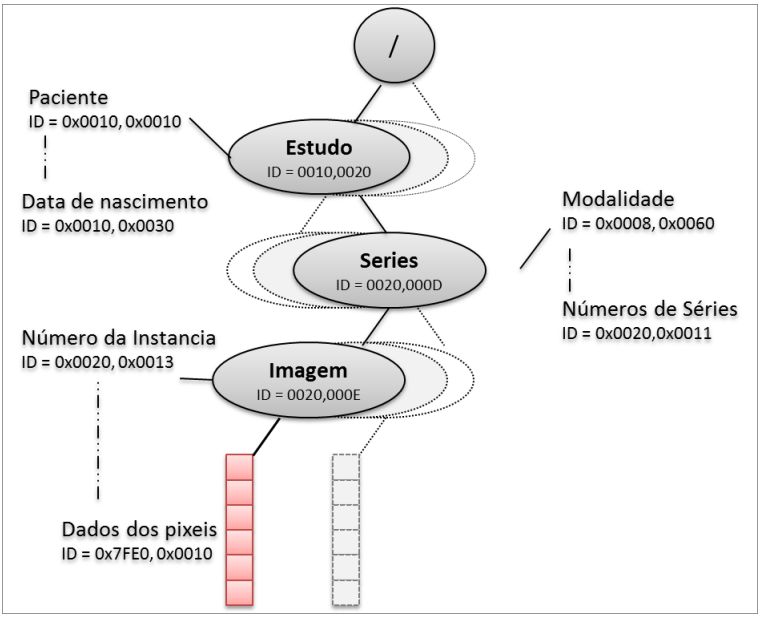


Figura 22 – Estrutura hierárquica em HDF5 de uma imagem DICOM

metadados sobre o exame e o paciente, contém os grupos das séries pertencentes ao exame. No último nível, ligada à série, há uma instância que é composta de vários metadados e os dados dos pixel.

Chamado de *pixel data*, este contém os binários dos pixel da imagem. É neste elemento que contém os dados de maior volume, pois uma vez que a qualidade dessas imagens é de grande importância, resulta um conjunto de dados consideravelmente grande. Dependendo do tipo da modalidade que, como já dito antes, essas imagens podem chegar até 600Mbytes.

Esta abordagem, inicialmente apresentou baixo desempenho na busca de uma imagem, devido ao formato de dados HDF5 utilizar internamente uma implementação de árvore B. Consequentemente, o gerenciamento de consultas de dados é desfavorável, e o HDF não provê nenhum elemento indexador dos objetos, tornando a busca de um dado um trabalho bruto. A fim de tornar a consulta mais eficaz, foi projetado um indexador em Clucene (SOURCEFORGE, 2011) pesquisada no trabalho de Thiago (PRADO, 2012), a fim de melhorar o desempenho na consulta dos dados. CLucene é uma plataforma de alto de desem-

penho desemvolvido para a linguagem C, com uma grande variedade de recursos voltado a indexação e busca de dados que são baseadas na plataforma Apache Lucene em Java. Os resultados demonstraram um ganho de desempenho bastante motivador.

Lembramos que o servidor CyclopsDCMServer implementa 3 comportamentos básicos de um gerenciador de protocolo DICOM: gravação, movimentação e consulta. Com a utilização do HDF5 e do indexador Clucene, o H5Wrap mantém o mesmo comportamento provido anteriormente, do qual daremos ênfase às funções de gravação e movimentação, pois são a base para entender a abordagem paralela estudada nessa dissertação.

4.1.2 Análise do Servidor

Atualmente, o servidor CyclopsDCMServer compilado com a API HDF5 possui um tamanho de aproximadamente 600 MB, com mais de 5928 arquivos, em que houve diferentes programadores contribuindo para o seu desenvolvimento. Portanto, a paralelização de uma aplicação de grande porte como esta torna o processo difícil, pois além de que seria preciso ter um conhecimento das classes mais importantes, cada processo paralelo teria um tamanho similar do servidor, podendo comprometer o desempenho da aplicação e de suas funcionalidades.

Outro ponto importante está na forma de paralelização. A princípio, as melhores opções são paralelizar a aplicação, armazenar um arquivo DICOM para cada processo paralelo, ou paralelizar o armazenamento do arquivo DICOM entre os processos. Como estamos buscando utilizar sistemas de arquivos paralelos e distribuídos, a paralelização do arquivo DICOM é vantajosa, pois busca utilizar adequadamente as características de acesso dos sistemas de arquivos paralelos, que são voltados para comportar aplicações paralelas de alta vazão de dados.

As alternativas que foram analisadas na paralelização do arquivo DICOM nos levam a dois princípios. Primeiramente, é separar as classes mais importantes, visando excluir funcionalidades do servidor não conveniente para esta abordagem, para facilitar a inclusão dos paradigmas de programação paralela. A segunda alternativa é criar uma aplicação paralela que se comunica com o servidor, e que somente forneça o paralelismo do E/S dos dados com o sistema de arquivos. Desta maneira, seria resolvido o problema de conhecimento de todo código, redirecionando o foco para a abordagem sugerida. A primeira solução requer mais tempo para análise profunda de código, enquanto

que na segunda são previstos problemas de latência com a comunicação de dados.

Ao paralelizar o arquivo DICOM, é importante observar que este possui uma grande quantidade de pequenas informações (por exemplo, nome do paciente, nome do hospital e tipo de modalidade) protocoladas no padrão DICOM, constituindo informações de no máximo 100 caracteres. Essas informações ao todo, compõem aproximadamente 10% de uma imagem DICOM, enquanto que a informação volumosa está relacionada ao *pixel data*. A paralelização dos metadados não apresentariam ganhos, por constituírem dados tão pequenos que resultaria em perda de desempenho no acesso a estes dados. Portanto, foi focado no acesso concorrente somente do *pixel data*, em que este é subdividido de acordo com o número de nodos paralelos.

O software utilizado para a comunicação entre os processos é o MPI. Esta escolha é devido ao fato de a biblioteca PHDF5 dar suporte para este paradigma. Especificamente, utilizamos a biblioteca MPICH2 que disponibiliza a API portátil ROMIO para realizar leituras e escritas paralelas em alto desempenho. O ROMIO é utilizado automaticamente pela API quando o sistema de arquivos suporta esse tipo de acesso.

Neste projeto foi buscado desenvolver ambas as alternativas porém, durante os estudos, foi necessário proceguir somente na segunda alternativa, pois a primeira apresentou problemas na execução do servidor. Assim, são apresentados experimentos utilizando a segunda alternativa para a paralelização do servidor. A arquitetura abordada é apresentada a seguir, e é chamada de PH5wrap.

4.1.2.1 PH5Wrap

O PH5wrap é o nome dado a esta abordagem que representa a camada de comunicação desenvolvida no H5wrap com os processos paralelos. A linguagem utilizada foi a Linguagem C, a fim de desenvolver a aplicação paralela de forma "mais simplista", evitar a utilização de bibliotecas que pudessem influenciar no desempenho do servidor e para obter maior integração com a biblioteca do HDF5. No entanto, esta medida trouxe dificuldades iniciais, pois foi necessário tratar a transmissão de dados e erros de transferência de dados como por exemplo, dados indevidos na memória e perdas de dados, encontrado na comunicação entre o servidor CyclopsDCMServer com o processo Master. Esta comunicação, especificamente, é realizada via *socket*.

No começo, foi desenvolvida uma arquitetura em que os processos

paralelos eram iniciados por uma chamada de sistema via código do servidor, porém, esta medida não era interessante, pois era necessário esperar o carregamento dos processos paralelos na memória para depois realizar a comunicação. Desde então, ao carregar o servidor são também carregados os processos paralelos, para que não houvesse espera do servidor pela iniciação dos processos.

A Figura 23 ilustra a arquitetura do PH5wrap, mostrando a camada H5Wrap, a comunicação via *socket* com o processo Master e as comunicações entre os processos MPI. O grande diferencial do PH5wrap é o seu comportamento ao acessar o *dataset* do *pixel data*. O foco da paralelização neste conjunto de dados ocorre porque ele representa o maior elemento de um arquivo DICOM. Em relação ao H5wrap, foi necessário adicionar uma interface de comunicação via *socket*, com o objetivo de transmitir os dados necessários para cada tipo de operação ao processo Master.

4.1.2.1.1 Armazenamento

Para realização da função de armazenamento são necessárias duas entidades SCP e SCU ativas. No caso de experimentos, utilizou-se uma aplicação que representa uma entidade DICOM provida na instalação das bibliotecas, chamada de storescu, que como o nome já descreve, é a entidade que faz a requisição do recurso de armazenamento, enquanto que o servidor CyclopsDCMServer é a entidade SCP, que realiza a requisição e retorna ao SCU o *status* da operação. A Figura 24 ilustra essa requisição.

A entidade StoreSCP necessita de alguns parâmetros para a enviar uma imagem DICOM ao servidor. Estes parâmetros são dados identificadores da entidade SCP como o nome da entidade, o IP e a porta utilizados pelo servidor. O servidor cria um *thread* de conexão, para receber os dados da entidade SCU, que fica responsável por receber e tratar os dados transmitidos. Os dados são analisados e mapeados para o formato HDF5 seguindo a estrutura de um documento XML que é carregado junto com a iniciação do servidor. O documento tem a função também de estruturar quais informações de cada nível (Estudo, Série e Imagem) serão indexadas. Por convenção, as informações que não estão associadas em nenhum nível são colocadas no último grupo da camada HDF5.

Em geral, as operações de criação, abertura, leitura e de escrita, providas pela biblioteca do HDF5, retornam um valor positivo inteiro

ou -1. O valor positivo é um número identificador local da própria API, utilizado para a manipulação dos elementos, sendo gerado um número diferente a cada instância do objeto. O segundo valor negativo é utilizado para identificar um erro de operação. Portanto, é visível a necessidade do servidor realizar tratamento de erros.

Na operação de armazenamento são realizados dois processos simultâneos: a criação dos dados em HDF5 e a criação dos índices desses dados. No primeiro caso é verificada a existência do arquivo HDF5 em disco. Em seguida, é realizada uma tentativa de acesso ao grupo correspondente ao nível da informação, que caso o retorno da

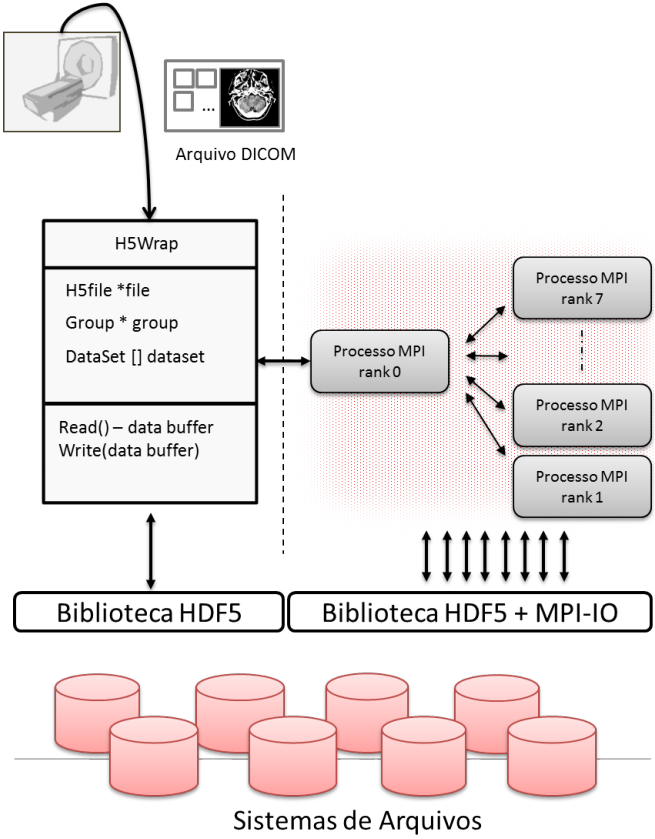


Figura 23 – Ilustração da abordagem PH5Wrap

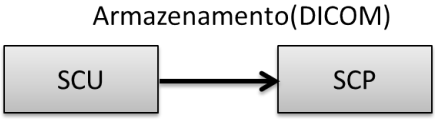


Figura 24 – Requisição do armazenamento através das entidades SCU e SCP

função da interface de programação indique falha, é criado um novo grupo correspondente. O mesmo acontece ao inserir os *data elements* (metadados) deste grupo em que a *tag* é utilizada para tentar acessá-lo, em caso de falha um novo *dataset* com nome da *tag* é criado. Após a operação de criação ter sido realizada com sucesso, são escritos os dados referente à *tag* e associados ao grupo corrente.

O outro processo da operação de armazenamento é a montagem do documento de índices do Clucene. Primeiramente é verificado através de uma consulta com o valor do ID único, se os dados correntes já foram indexados. Caso não tenham sido indexados, é criado um novo documento correspondente ao nível (Estudo, Série, Imagem), os campos deste documento são criados de acordo com o grupo de cada *data elements*, os valores são processados pelo analisador do Clucene e inseridos no documento, que por fim é adicionado ao índice.

No processo de criação de grupos é utilizada a função H5Gcreate, que utiliza um número identificador local de um grupo superior, e que caso seja o grupo raiz, utiliza-se o número gerado na abertura do arquivo. O outro parâmetro é nome do grupo, composto por uma cadeia de caracteres, similar a um caminho de um diretório dos sistemas de arquivos LINUX, que irá representar o nome do grupo. Essa cadeia de caracteres é representada pelo valor da *tag* referente a cada nível, que torna a ser identificado como chave quando inserido no documento de índices. Criado o grupo, o procedimento segue com a criação dos *datasets* pertencentes a cada grupo.

A escrita dos metadados do DICOM se inicia com a criação do elemento *dataset* via função H5Dcreate, em que necessita identificar o número identificador do grupo e o nome do *dataset*. Criado com êxito é retornado um número identificador do *dataset*, que é utilizado para realizar a função de escrita H5Dwrite. Nesta função é preciso configurar o número identificador do *dataset*, o formato de dados que serão escritos (por exemplo, caracteres, inteiros e ponto flutuante), o espaço da memória, o espaço em disco, a forma de acesso e o *buffer* em

memória que será escrito.

Ambos os parâmetros espaço em memória e espaço em disco são definidos pelo elemento *dataspace*, representados pela função H5Screate simple, que no caso do armazenamento, utiliza como parâmetro o tamanho do *buffer* que está em memória, e o tamanho do *buffer* que será escrito em disco. Estes espaços são definidos via a API do HDF, em que não necessariamente ambos os espaços devem possuir a mesma forma de matriz, porém devem constituir um mesmo tamanho. A Figura 7 mostrada anteriormente, ilustra essa diferença de forma dos espaços de memória e disco. Todos os dados dos metadados nesta implementação são matrizes de ordem um e idênticas, na qual suas dimensões são baseadas somente no tamanho dos elementos de dados (data element) das imagens DICOM.

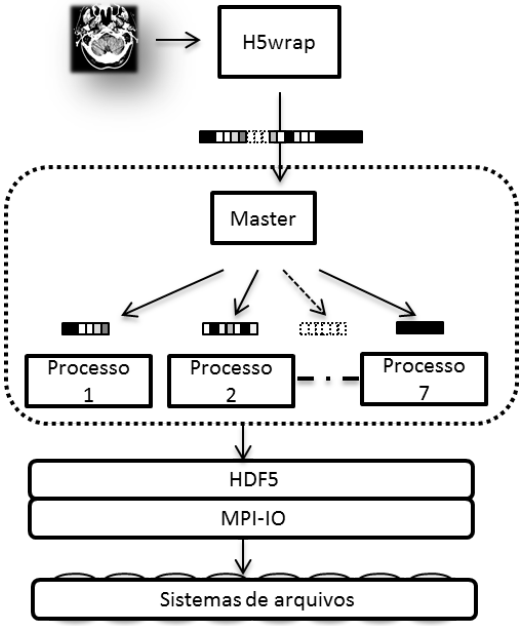


Figura 25 – Ilustração do armazenamento do *pixel data* no PH5wrap

Após o armazenamento dos metadados do DICOM, o H5Wrap verifica se o elemento que será armazenado contém a tag "7FE0,001", então inicia o processo de enviar este *buffer* à aplicação paralela. Pri-

meiramente, o servidor envia dois dados referentes ao tipo de função que será realizado e o endereço (os grupos para o acesso do *dataset*) que será armazenada, e logo em seguida envia o *buffer* que será armazenado. Esse procedimento pode ser visualizada na Figura 25.

No momento em que o H5wrap envia os dados (caminho dos grupos, e a flag da função) ao processo Master, este se responsabiliza por distribuir estes dados aos outros nodos via a função MPI Bcast (*Broadcast*)[1]

Após o processo máster enviar os dados comuns para os outros nodos, é enviado para cada processo uma partição do *buffer*, na qual cada partição é gerada dividindo o *buffer* pelo número de processos ativos (*rank*). Ou seja, caso haja somente dois nodos ativos, o *buffer* será dividido pela metade. Este envio é lento, pois o processo envia os buffer para cada processo, sequencialmente.

Quando todos os processos receberem uma partição do *buffer*, eles seguem o processo de escrita dos dados. O cálculo do *dataspace* em arquivo de cada nodo é realizado dividindo o tamanho do *buffer* original em tamanhos iguais e multiplicando esse resultado com o seu *rank* para identificar qual índice do vetor será acessado pelo processo (*start*). Desta maneira, cada processo identifica a sua região de acesso.

Em seguida, todos os processos criam na memória local um novo *dataset* via função H5Dcreate, sendo que este retorna um número identificador único para cada processo. Os nodos definem a função *hyperslabs* de dados contínuos, e realizam a operação de escrita no respectivo espaço de dados no *dataset*, via a função *H5Dwrite*. Para a finalização, o processo Master envia uma mensagem para o servidor com o *status* da operação. O servidor termina a operação reenviando para o cliente o *status* da operação.

$$size = \frac{dataset}{totalrank} \tag{4.1}$$

$$start = rank * size \tag{4.2}$$

[1] *Broadcast* refere-se a um método de transferência de uma mensagem para todos os destinatários simultaneamente.

4.1.2.1.2 Recuperação

A função de movimentação requer mais uma entidade no procedimento, como ilustrado na Figura 26. Como citado no capítulo 2.1.3, essa entidade é a responsável por receber o arquivo DICOM retornado da função de recuperação dos dados do servidor. Da mesma maneira que é realizado em experimentos de armazenamento, utiliza-se a aplicação entidade moveSCU para inicializar o serviço de movimentação no servidor, e a aplicação entidade storeSCP para receber o DICOM restaurado do servidor. Portanto, o servidor representa duas entidades, devido as funcionalidades de requisitar e prover um serviço, representando a entidade SCP para o moveSCU e a entidade SCU para o storeSCP.

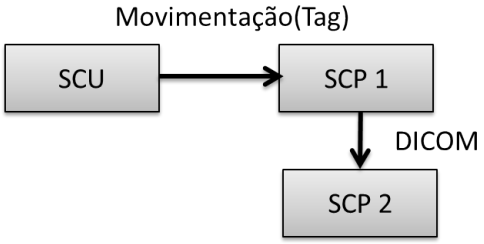


Figura 26 – Requisição de recuperação através das entidades SCU e SCP

Ao instanciar a entidade *movescu*, necessita identificar as entidades envolvidas e os filtros dos níveis do parâmetro de consulta. Esses filtros são necessários devido à necessidade de fazer uma busca inicial para verificar se os dados existem. Normalmente, esses filtros são realizados por um nível de hierarquia e os dados configurados nesses filtros são as chaves únicas do determinado nível. Por exemplo, quando se deseja recuperar um estudo de um paciente, é preciso especificar pelas *tag*s, o nível da hierarquia (“0008,0052=STUDY”) e o identificador único desse nível (“0020,000D=1.3.12.2.1107.5.5.6.953201.20110622135033.66019”). A partir do valor da *tag* (que, por convenção, chamaremos de chave) é acessado o documento de índices do Clucene, onde então são buscadas todas as outras chaves de todas as séries e imagens pertencentes a este estudo, as quais utilizadas para para acessar os grupos e dados do arquivo HDF.

Quando encontradas as chaves dos níveis no documento de índices, para cada nível, é realizado o acesso ao grupo respectivo na estrutura HDF e são carregados todos os *datasets* pertencentes a esse grupo. Para acessar os grupos e *datasets* do HDF, dois dados são importantes. O primeiro é o número identificador local do HDF5 (número positivo) de, pelo menos, um grupo antecedente ao grupo em questão, que caso seja o grupo raiz, deve-se utilizar o identificador gerado na abertura do arquivo. Em seguida, deve-se utilizar o caminho desse grupo, representado por uma cadeia de caracteres similar a um caminho de um diretório dos sistemas de arquivos LINUX.

Este caminho pode variar dependendo do grupo identificador utilizado no primeiro parâmetro, e caso se deseje somente acessar os *datasets* e os grupos deste nível, é possível especificar somente os caminhos locais. No caso do servidor, esses caminhos são construídos na operação de armazenamento, na qual os endereços dos grupos são representados pelas chaves provindas da busca no documento de índices.

As funções utilizadas para o acesso de grupo e *datasets* são os H5Gopen e H5Dopen respectivamente, em que ambas retornam um número identificador local do grupo ou *dataset*. No caso dos grupos, esses identificadores locais, além de proporcionar o acesso aos grupos, são utilizados para outras funcionalidades de programação, como por exemplo, buscar o número de nodos filhos do grupo em questão. Quanto aos *datasets*, além dos identificadores serem utilizados para realização das leituras e escritas dos dados, são importantes também para configurar a forma de acesso que será realizado no *dataset*. No caso do paralelismo do HDF5, é através destes identificadores que é configurada a maneira de realizar o paralelismo no determinado *dataset*.

Adquirido o identificador do *dataset*, é chamada a função H5Dread, para realizar a leitura dos dados em disco. A configuração desta função é similar à função de escrita H5Dwrite, na qual os parâmetros são idênticos, com a distinção da origem e do destino dos dados. Neste caso, a operação de leitura irá acessar o espaço em disco definido pela função H5Screate simples, e carregado no espaço de memória.

Do mesmo modo da operação de armazenamento, quando o H5wrap depara-se com a tag "7FE0,001"do *pixel data*, inicia o processo de comunicação com os processos MPI. O servidor H5wrap envia ao processo master os dados referente ao tipo de função e o endereço, que são distribuídos via *broadcast* aos processos.

Cada nodo do processo chama uma função que calcula qual intervalo (*dataspace*) do vetor será acessado pelo processo com base do seu *rank* no ambiente MPI. Para isso, são realizadas duas funções:

a primeira, que calcula o índice inicial de cada processo, e a outra, que calcula o tamanho do "subarray"onde será lida do arquivo para a memória.

Cada nodo do processo chama uma função que calcula qual intervalo (*dataspace*) do vetor será acessada pelo processo com base do seu *rank* no ambiente MPI. Para isso, é realizado duas funções, a primeira que calcula o índice inicial de cada processo, e a outra que calcula o tamanho do subarray onde será lida do arquivo para a memória.

Essas funções são as mesmas demonstradas na seção anterior, em que o cálculo do tamanho do *buffer* em cada processo é realizado dividindo o dataset em tamanhos iguais e multiplicando esse resultado com o seu *rank* para identificar qual índice do vetor será acessado pelo processo.

A partir do tamanho do *buffer* e do índice do vetor, é criado espaço em memória (*dataspace*) onde serão alocados os dados restaurados e, em seguida, são definidos a forma da função *hyperslabs* e a forma de acesso (coletivo ou independente). Definidas essas configurações, cada processo realiza a função de leitura H5Dread, retornando zero quando a operação foi realizada com sucesso, e negativo caso contrário. Logo, todos os processos enviam, via função MPI Gather [2], para o processo Master a sua respectiva parte lida do *dataset*. Por fim, o processo Master envia o *buffer* completo para o servidor.

O servidor após receber o buffer do *pixel data*, todos os dados em memória são empacotados em estruturas DICOM, e enviado ao servidor storeSCP o arquivo DICOM.

4.2 TRABALHOS RELACIONADOS

Os dois primeiros trabalhos são bastante interessantes para esta pesquisa, pois são temas que buscam utilizar bibliotecas paralelas como uma solução para contornar problemas de gargalos de disco para grandes volumes de dados gerados em cálculos científicos. O primeiro trabalho relacionado de Laghave *et al.* (LAGHAVE et al., 2009), foca no uso do HDF5 paralelo para contornar problemas de gargalos de disco gerado pela aplicação que armazena os resultados em disco serialmente. Esta aplicação envolve férmions dinâmicos para a estrutura nuclear (MFDn) e utilizaram o Lustre como sistema de arquivos. Neste trabalho, realizaram-se testes com os modelos coletivos e independentes

[2]Gather refere-se a um método de envio de mensagens de vários remetentes simultaneamente para um mesmo destinatário.

disponibilizados no HDF5. Os resultados demonstraram que com os arquivos de tamanho acima de 20 GB, o HDF5 paralelo tornou-se mais produtivo do que sequencial e que o modelo independente demonstrou ser mais rápido que o coletivo.

O trabalho de Adelmann *et al.* (ADELMANN et al., 2006) foca em utilizar E/S paralela para partículas baseados em simulação de acelerador que envolveu grandes quantidades de dados e arrays dimensionais. Eles utilizaram duas API: a primeira é o próprio MPI-IO puro, e a segunda é a API do HDF5 paralelo. Ele comparou o desempenho de leitura e gravação em simulações entre HDF5, MPI-IO e um arquivo por processo (serial). O HDF5 mostrou bom desempenho na escrita, embora o MPI-IO tenha mostrado melhores resultados.

No trabalho de Cohen *et al.* (COHEN et al., 2006) é apresentado um estudo baseado em extensões de sistemas gerenciadores de bancos de dados relacionais que permitem a representação de dados científicos em formatos multidimensionais. Para efeito de comparação, foram utilizados os NetCDF e o HDF, dois dos formatos hierárquicos nativos mais populares com o Oracle 10g. Nesse trabalho, ainda foram realizadas operações estatísticas utilizando as extensões dos SGBDs em comparação com as operações nativas do NetCDF e HDF. Foi concluído que o banco de dados teve um bom desempenho quando o número de elementos por matriz era pequeno, mas não teve uma boa escala como os modelos científicos de arquivo quando o número de elementos aumentou.

Outro trabalho interessante do Howison *et al.* (HOWISON et al., 2010b), em que é proposta uma forma para otimizar o desempenho das bibliotecas HDF5 e MPI-IO para o sistema paralelo de arquivos distribuídos Lustre. Mediram o desempenho em três sistemas diferentes para demonstrar a robustez de otimizações em diferentes configurações e obtiveram otimizações em desempenho paralelo de E/S, em alguns casos, de até 33 vezes mais eficiente.

Os trabalhos de Fahey *et al.* (FAHEY; LARKIN; ADAMS, 2008) Laros *et al.* (LAROS et al., 2007) e Weikuan Yu *et al.* (YU; VETTER; ORAL, 2008) são interessantes artigos pois têm como foco trabalhar com comparações e melhorias de desempenho de E/S em plataformas como Jaguar e Red Storm. Estes são plataformas de supercomputação, onde provêm serviços computacionais, como por exemplo, GTC para *spectrum*, programa oceano paralelo (POP) de modelagem de oceano, programa de cálculos físicos, entre outros. Em destaque o trabalho proposto por Weikuan *et al.* tem interessantes comparações com diferentes tipo de E/S. Eles propõem melhorar e aperfeiçoar o acesso em paralelo

de uma aplicação chamada Jaguar do Laboratório Nacional Ridge Oak e visaram vários objetivos dentre eles, a caracterização do desempenho E/S paralelo de uma unidade de armazenamento e a escalabilidade de todo o sistema. Os resultados demonstraram um melhor desempenho e menos escalabilidade de E/S paralelo em modo compartilhado e que o FLASH E/S pode melhorar 34% com certos ajustes em E/S compartilhado.

Tratando de sistemas de arquivos, temos o trabalho de Boito *et al.* (BOITO et al., 2011) que realizou três experimentos em um sistema de modelo atmosférico. Os experimentos tratam de aplicar uma mesma aplicação com um diferencial na sua arquitetura que adiciona *thread* nos seus processos. Os testes realizados com escrita local (disco local) obtiveram o melhor desempenho por não necessitar comunicação entre os nodos do sistema de arquivos. O ambiente com *threads* teve um melhor desempenho quando utilizado os sistemas de arquivos distribuídos PVFS, devido à utilização das facilidades do MPI. Nos experimentos realizados foram utilizados 30 clientes para um ambiente PVFS de 4 nodos. Como cada processo acessa constantemente o sistema de arquivo para escrever os resultados, quando se tem mais de um processo por máquina, estes competem pelo acesso ao disco, diminuindo o tempo de execução. Isso não ocorre na aplicação com *Threads*, uma vez que roda um processo por máquina e melhor uso da memória

5 AMBIENTE E RESULTADOS EXPERIMENTAIS

Os experimentos realizados neste projeto têm como objetivo avaliar o desempenho de acesso do servidor com a aplicação paralela em relação à aplicação serial utilizando o HDF5 sob um sistema de arquivos distribuído e avaliar a escalabilidade de acordo com o aumento do número de imagens no sistema. O ambiente utilizado para a realização de testes foi um cluster não dedicado de oito máquinas disponíveis, pertencentes ao laboratório de Telemedicina. Estas máquinas são utilizadas para o desenvolvimento de aplicações. Portanto, a arquitetura de redes utilizada não é favorável se comparada com um cluster dedicado somente para o armazenamento. A sua arquitetura de rede é ilustrado na Figura 27.

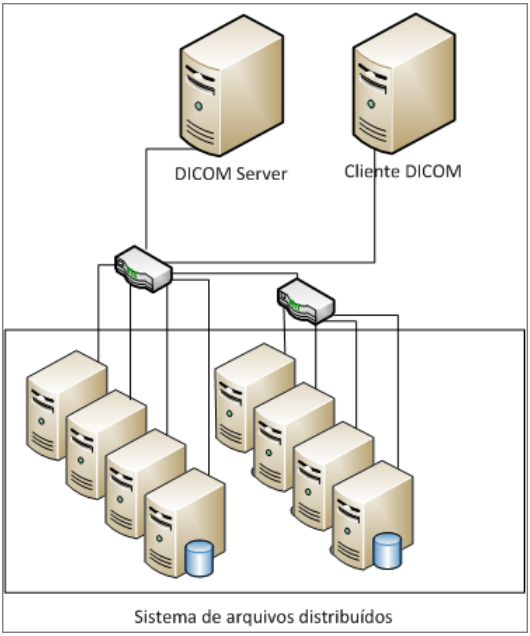


Figura 27 – Arquitetura da rede do cluster

O cluster com 8 máquinas possui 4 máquinas que estão ligadas a um suíte ethernet de 10/100 MBytes que esta conectado a um outro suíte gigabit. Desta maneira, pode-se relatar que a conexão com esses

quatro computadores pode comprometer o desempenho da aplicação e dos SAD dependendo da largura de dados transferidos. Essas máquinas diferem também no hardware conforme a tabela abaixo.

Tabela 1 – Configuração dos nodos do cluster

HOSTNAME	MEMÓRIA	PROCESSADOR	DISCO
Cluster1	8Gbytes	Intel Core i5 2.8 GHz	20 GBytes
Cluster2	8Gbytes	Intel Core i5 2.8 GHz	20 GBytes
Cluster3	8Gbytes	Intel Core i5 2.8 GHz	20 GBytes
Cluster4	8Gbytes	Intel Core i5 2.8 GHz	20 GBytes
Cluster5	8Gbytes	Intel Core i5 2.8 GHz	20 GBytes
Cluster6	8Gbytes	Intel Core i5 2.8 GHz	20 GBytes
Cluster7	2Gbytes	AMD Athlon x2 2.0 GHz	20 GBytes
Cluster8	8Gbytes	Intel Core i5 2.8 GHz	20 GBytes

O sistema operacional utilizado em todas as máquinas do cluster é o CentOS 6.2. O motivo da utilização do CentOS é o fato de a distribuição fornecer os pacotes da aplicação Lustre, sem a necessidade de compilar a fonte da aplicação, que é mais intricado se comparado com outras aplicações utilizadas nos experimentos. As versões das aplicações são listadas abaixo.

- HDF5 - Versão 1.8.9.
- MPI - MPICH2, versão 1.4.1-p1.
- PVFS - OrangeFS, versão 2.8.5.
- Lustre - Versão 2.1.55.
- FHGFS - Versão 2011.04-r19.
- CEPH - Versão 0.47.2-4

A máquina que hospeda o servidor DICOM é uma máquina servidora de máquinas virtuais para fins de experimentos, em que foi virtualizada uma máquina com a configuração de 1GB de RAM, processador Intel de 1.6 GHz com 10 GByte de disco e sistema operacional GNU/Linux CentOS versão 6.2. A máquina do cliente DICOM é um servidor de backup de imagens DICOM reais com processador AMD 64 x2 1 GHz e 2Gbytes de memória.

Para a realização dos experimentos, foram coletados os tempos em segundos da execução das funções de escrita e leitura somente do *pixel data*, e os tempos totais de construção e armazenamento das imagens DICOM (metadados e *pixel data*) a fim de comparar o desempenho de ambos as arquiteturas paralela e serial sob diferentes sistemas de arquivos. A base de imagens utilizadas nos experimentos, são imagens coletadas aleatoriamente sob a base de dados de produção. Além isso, foi buscado comparar diferentes tipos de sistemas de arquivos distribuídos, a fim de detectar o sistema com melhor desempenho de acesso e de melhor controle de escalabilidade.

A coleta dos tempos foi realizada via código das aplicações, através da diferença entre o tempo inicial da execução dos comandos de armazenamento ou recuperação e o tempo final e a diferença do tempo inicial e final do comando de escrita ou leitura.

Foram realizados cenários de experimentos de leitura serial e paralela, como também cenários de escrita serial e paralela. Em todos os cenários foi associado somente um serviço SCU para um SCP, descartando ambientes com múltiplos clientes, pois estes tratam de trabalhos futuros. Foram utilizadas variações de volumes de imagens de 1000, 2500, 5000 e 10000 a fim de avaliar a escalabilidade dos sistemas.

Em cada volume foram executados, tanto para leitura quando para escrita, 25 iterações para garantir significância estatística aos resultados. Esses volumes de dados são imagens aleatórias que estão armazenadas na máquina cliente, e estas imagens são constituídas de tomografias computadorizadas, angiografias, ultrassons, ressonâncias magnéticas e raios-x. Para as 25 iterações dos cenários de leitura e escrita paralela, utilizou-se o mesmo número de servidores de dados para servir como processos do MPI-IO, a fim de observar o comportamento da escalabilidade do sistema de arquivos e desempenho da aplicação com todo ambiente disponível para os experimentos.

No entanto, foram realizados testes em quantidades menores de nodos paralelos, para comparar o desempenho do E/S em diferentes sistemas de arquivos. Para este caso, somente foi realizada uma iteração por experimento a fim de verificar o impacto do número de processos paralelos.

A coleta de tempo dos experimentos foi realizada via código, através da diferença do tempo inicial da execução dos comandos de armazenamento, ou recuperação, com o tempo final. Foram mensurados também os tempos de E/S em disco e o tempo de transmissão do *pixel data* pelo *socket* do H5wrap para o nodo Master para então calcular o tempo médio de perda de desempenho desta comunicação.

Na montagem dos sistemas de arquivos, utilizou-se um padrão de arquitetura para todos os SADs, em que a máquina Cluster 1 atua como servidor de metadados e servidor de dados, enquanto as outras 7 somente serão utilizadas como servidores de dados. Foi definida esta abordagem devido ao fato de o ambiente de experimentos somente conter uma aplicação cliente no sistema de arquivos, buscando contornar problemas de *overhead* no servidor de dados, além de o cluster se constituir de poucos nodos de servidor de dados, não necessitando mais de um servidor de metadados.

Nas próximas seções, os gráficos dos resultados são apresentados com o eixo horizontal representando a variação do volume dos exames enviados ou presentes na base e o eixo vertical a média de tempo da execução da operação das funções, das bibliotecas empregadas, em segundos.

5.1 ARMAZENAMENTO

Os experimentos aplicados no armazenamento são divididos em volumes de imagens (1000, 2500, 5000, 10000) para cada forma de acesso serial e paralelo, a fim de averiguar o comportamento de cada armazenamento de acordo com a escalabilidade dos sistemas.

O gráfico representado na Figura 28 compara o desempenho de armazenamento de uma imagem DICOM em HDF5 em paralelo e serial nos sistemas de arquivos paralelos FHGFS e PVFS. Para os ambientes CEPH e Lustre, não foi possível realizar o armazenamento paralelo devido a restrições que serão discutidas no final deste capitulo, enquanto os experimentos com o servidor serial é estudado na próxima subseção.

Como se pode observar, independente do sistema de arquivos, a aplicação composta somente do armazenamento serial obteve um desempenho superior ao do PH5wrap. Este desempenho no caso do PVFS teve uma média em torno de 0.3369 segundos de diferença no pior caso, enquanto no FHGFS foi de 0,2558 segundos, também no pior caso.

Outro dado que é possível observar é o desempenho de cada sistema de arquivos. É visível que o servidor sob o FHGFS foi o sistema que obteve o melhor desempenho tanto no serial quanto no paralelo, em que a sua média de armazenamento em todos os volumes manteve-se próximo 0.828 segundos. Como também nota-se que o armazenamento no FHGFS em serial manteve o desempenho mesmo com o aumento do volume de dados, enquanto o PVFS serial teve uma perda de desempe-

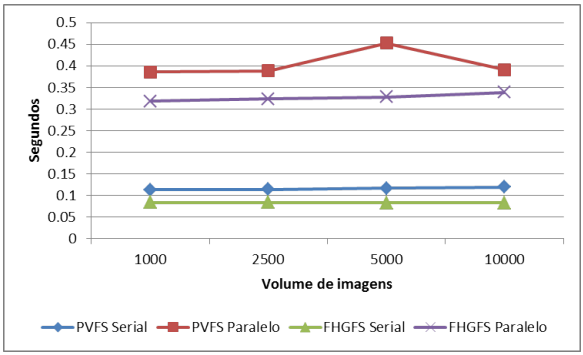


Figura 28 – Tempo médio do armazenamento paralelo e serial em HDF5

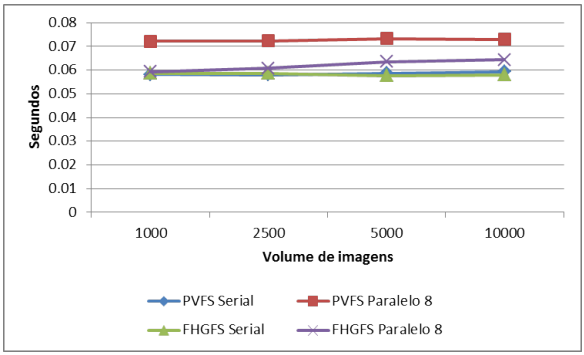


Figura 29 – Tempo médio de escrita do *pixel data*

nho de até 5% comparando os volumes de 1000 e 10000. Os ambientes paralelos FHGFS e PVFS obtiveram uma perda de 6,35% e 1,36%, respectivamente. Esta mesma análise é verificada também na Figura 29.

No gráfico 30, buscou-se uma comparação do tempo de E/S dos processos paralelos, variando a quantidade de nodos paralelo da aplicação, a fim de verificar a escalabilidade do PH5wrap em termos de número de processos paralelos. Para estes experimentos apresentados na Figura 30, em exceção, foi realizada somente uma tentativa, não havendo significância estatística, e o sistema de arquivos utilizado foi o PVFS.

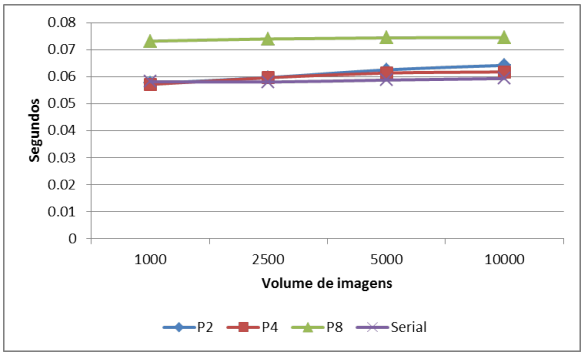


Figura 30 – Tempo médio das escritas dos *pixel data* de acordo com o número de nodos paralelos

O objetivo deste gráfico é buscar um melhor ambiente paralelo, uma vez que estamos trabalhando com uma coleção de imagens aleatórias e a média do tamanho das imagens pode se adaptar melhor de acordo com o número de processo. Isso se deve ao fato de que cada *pixel data*é dividido de acordo com o número de processos paralelos existentes no ambiente, e dependendo do tamanho da imagem, a divisão desta em muitas regiões pode levar à depreciação do desempenho da aplicação.

É possível analisar que com este conjunto de imagens aleatórias utilizadas para os experimentos, quando o E/S paralelo utiliza o número de nodos igual 4, o desempenho médio comparado com o 8 nodos é de 0,013 segundos, ficando próximo do desempenho da escrita serial. Quanto ao volume de 1000 dados, o ambiente paralelo com 4 nodos teve uma sutil vantagem de 1,03 milissegundos, porém, com o volume de 10000 dados a vantagem é perdida, sendo que a diferença é de 2,44 milissegundos

Em relação aos outros resultados, com 8 processos paralelos, manteve-se uma média mais constante comparada com as demais, mesmo com o aumento do volume de dados. Enquanto com 8 processos houve uma perda de desempenho de aproximadamente 1,8% quando calculado a diferença entre o volume de 10000 e 1000, com 4 e 2 processos obteve-se uma perda de 2,91% e 2,71%, consecutivamente.

5.1.1 Armazenamento Serial

Nesta seção é discutido o desempenho dos quatros sistemas de arquivos citados nesta pesquisa, apresentadas nas Figuras 31 e 32. Para os experimentos foram utilizados somente o servidor com o H5wrap, e cada teste segue o padrão dos testes anteriores. Em todos os ambientes, buscou uma configuração simples e padrão do sistema quando este é instalado.

É notável que o sistema de arquivos CEPH tenha obtido um rendimento na escrita bem superior que os outros sistemas, enquanto o Lustre obteve baixo desempenho na escrita dos dados. No armazenamento do DICOM, os sistemas PVFS, FHGFS, Lustre e CEPH tiveram o desempenho médio de 0,11490, 0,08291, 0,11497 e 0,01714 segundos, respectivamente, sendo que o CEPH teve um desempenho de 82,59% superior à média do sistema ao todo e a escrita do pixel data em torno de 92.11% mais eficiente.

Outra análise que se pode fazer é o baixo desempenho do Lustre ao escrever o *pixel data*, de 85,08% inferior em relação a média de todos os SAD. Contudo, a escrita dos metadados foi mais eficiente do que comparado com o PVFS, que mesmo tendo a escrita do pixel data superior ao Lustre, a média de armazenamento de todos os dados do DICOM é equivalente ao do Lustre, sendo a média de ambas em torno de 0.1149 segundos.

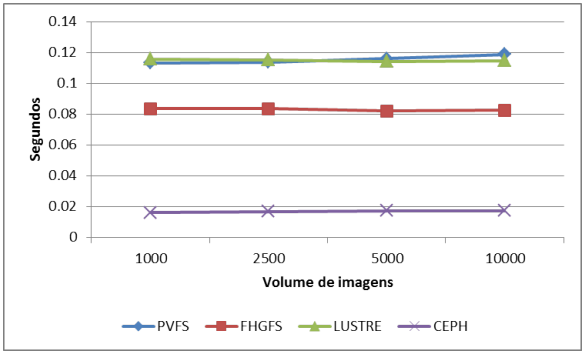


Figura 31 – Comparação do tempo de armazenamento em serial dos sistemas de arquivos

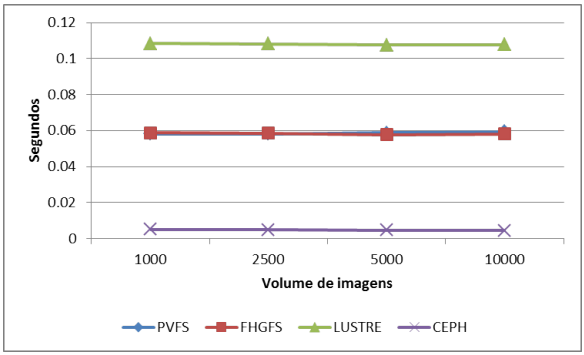


Figura 32 – Comparação do tempo de escrita serial dos sistemas de arquivos

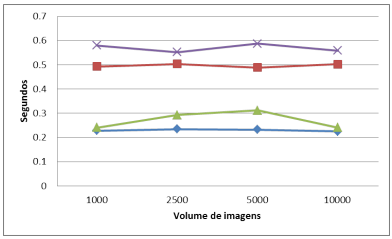
5.2 RECUPERAÇÃO

Assim como os experimentos de armazenamento, nos experimentos de recuperação não foi coletado o tempo do indexador Clucene. O Clucene somente é usado para a recuperação da localização de cada exame dos cenários de estudo (centenas de exames), série (dezenas de exames) e imagem, em que são recuperados todos os UID para construção do caminho para extração (path) do conteúdo do HDF5. A partir dos resultados dos caminhos obtidos, é iniciada a contagem de tempo e é somente finalizada quando recuperado os dados e convertidos a um arquivo DICOM.

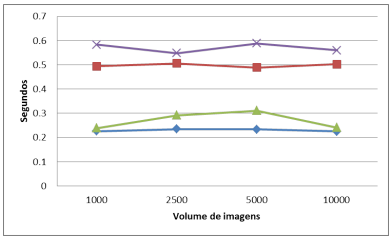
A realização dos experimentos de recuperação é projetada de acordo com a hierarquia definida nos projetos anteriores, em que são recuperados os dados (metadados e *pixel data*) de um estudo completo e uma série completa, ou seja, restaurando todos os dados das hierarquias correspondentes ao estudo e série, e por fim, são feitos experimentos utilizando somente a hierarquia da imagem. No caso do estudo selecionado para os experimentos, nas bases de dados geradas através dos experimentos de armazenamento há uma única série, e esta série possui 4 imagens. Desta maneira, foram calculadas as médias das somas de todas as imagens recuperadas dos estudos, das séries e das imagens.

Os sistemas de arquivos utilizados nos experimentos são os mesmos utilizados no armazenamento. Na Figura 28 é comparado o desempenho do servidor paralelo PH5wrap com o servidor serial H5wrap

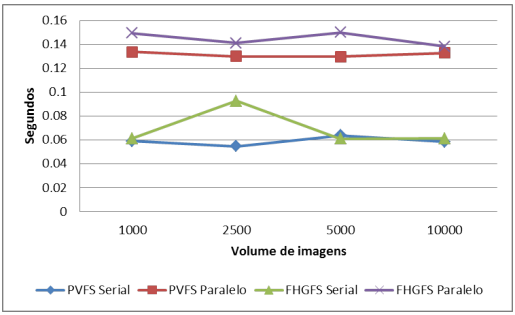
ambos nos sistemas de arquivos PVFS e FHGFS. No caso da leitura paralela, não requer o mesmo controle de escrita concorrente provido pela maioria dos sistemas de arquivos, que dificultou a realização dos experimentos anteriores, sendo possível realizar os experimentos de leitura paralela nos sistemas CEPH e Lustre. Mas para manter a coerência nos resultados, só será demonstrado o acesso paralelo dos mesmos sistemas de arquivos utilizados nos experimentos de armazenamento.



(a) Estudo



(b) Serie



(c) Imagem

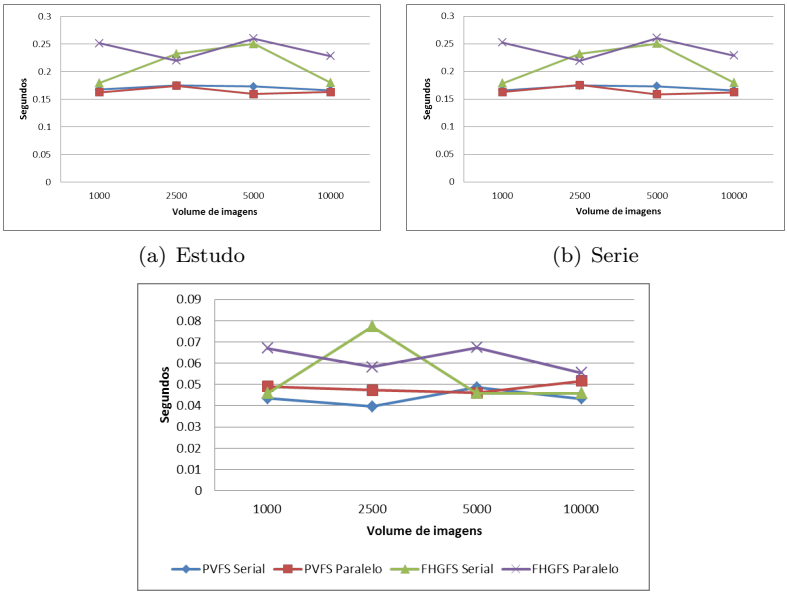
Figura 33 – Tempo médio de recuperação de um estudo completo, uma série completa e de uma imagem em paralelo e serial

Nos gráficos da Figura 33 são ilustrados os resultados em 3 gráficos referentes a cada hierarquia. Os resultados dos estudos e das séries são semelhantes, devido a hierarquia da série utilizada possuir a mesma quantidade de imagens recuperadas dos estudos. Pode-se notar, primeiramente, que o servidor serial manteve vantagem na recuperação das imagens DICOM, e em destaque, o PVFS se sobressaiu nos experimentos serial e paralelo, comparado com o FHGFS. Nos experimentos de estudos e séries, essa diferença manteve-se em torno de 0.036 segundos no serial e 0.074 no paralelo, um ganho de 4% e 7% ,

respectivamente.

Outro ponto observável é a média de tempo de recuperação que é mantida nos SAP, principalmente no PVFS, que mesmo com o aumento do volume de dados, manteve-se na média de 0.23 segundos no serial e 0.498 segundos no paralelo em ambos os experimentos de estudo e série.

Os experimentos demonstrados no terceiro gráfico mostram o comportamento na recuperação de uma única imagem. Neste caso, a recuperação do FHGFS em serial possuiu um desempenho semelhante do PVFS nos diferentes volumes exceto de 2500 imagens, que pode ter tido influências mais significativas da rede de comunicação.



(a) Estudo

(b) Serie

(c) Imagem

Figura 34 – Tempo médio de leitura do *pixel data* de um estudo completo, uma série completa e de uma imagem em paralelo e serial.

Em resumo a Figura 34, basicamente, a aplicação paralela no PVFS teve um desempenho inferior de aproximadamente 46% no estudo e na série e 45% inferior na imagem, enquanto que no FHGFS esse desempenho foi de aproximadamente 47% e 42%.

A Figura 34 mostra resultados bem interessantes, que da mesma

maneira que foi coletado o tempo de escrita, nestes gráficos é coletado o tempo de leitura do *pixel data*, de cada experimento realizado na recuperação e mostrou-se que a leitura paralela teve um desempenho equivalente à leitura serial em determinados casos. Por exemplo, na leitura em paralelo no PVFS teve um desempenho de 4,41% em relação à leitura serial PVFS nos casos do nível de estudo e série, porém, teve uma perda de desempenho em torno de 10,15% no nível de imagem.

É possível acrescentar que o tempo de comunicação do *socket* determinou bastante a perda de desempenho em ambas as funções de recuperação e armazenamento em paralela. A sua média de tempo de *socket* independente do ambiente, foi de 0.078 segundos para cada envio do *pixel data*. No caso da recuperação de um estudo ou uma série, a média da soma de todos os tempo de *socket* foi de 0.267 segundos. É notável a influência negativa deste tempo para abordagem, porém, quando retiramos esse tempo da média do tempo total da recuperação serial, o desempenho ainda é baixa, comparado com o tempo da abordagem serial.

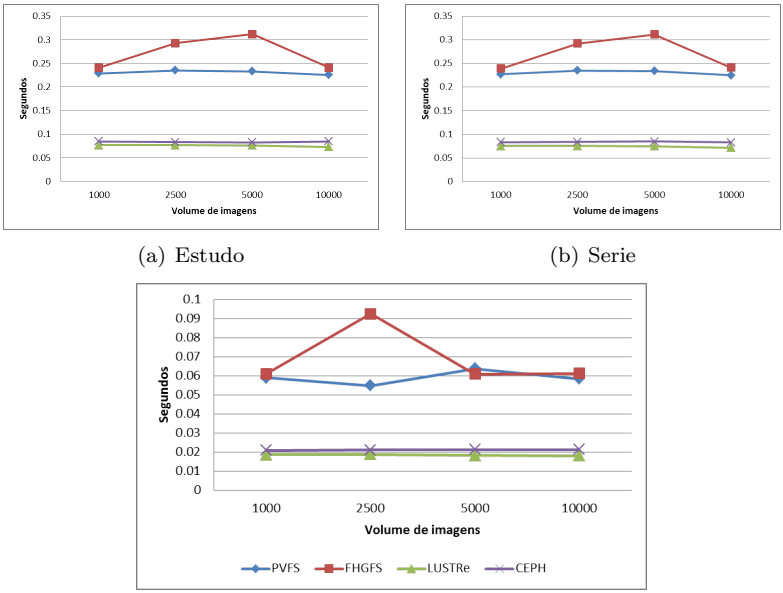
5.2.1 Recuperação Serial

Nos resultados de recuperação serial apresentadas na Figura 35, nos quatro ambientes distribuídos é possível observar que ambos o Lustre e o CEPH obtiveram um desempenho superior comparado aos sistemas de arquivos paralelos FHGFS e PVFS. No caso do Lustre, obteve uma média de desempenho superior de 67,36% sob o PVFS e 71,38% sob o FHGFS, enquanto o CEPH obteve um desempenho médio de 63,73% e 68,59%.

Além disso, ambos os sistemas de arquivos CEPH e o Lustre, ao contrário do FHGFS, também mostraram manter a escalabilidade do sistema equivalente com o aumento do volume de dados. A média de recuperação de um estudo completo e de uma série completa é de 0,075 segundos para Lustre e 0,083 segundos para o CEPH e a média de recuperação do nível de imagem é de 0,018 segundos no Lustre, e 0,021 segundos no CEPH

5.3 ANÁLISE DOS SISTEMAS LUSTRE E CEPH

Como se pode observar nos resultados somente dos experimentos em serial, ambos Lustre e CEPH demonstraram maior capacidade de



(a) Estudo

(b) Serie

(c) Imagem

Figura 35 – Tempo médio de recuperação de um estudo completo, uma série completa e de uma imagem em paralelo e serial, sob os sistemas de arquivos.

acesso serial nos experimentos de recuperação e o bom desempenho do CEPH no armazenamento. No entanto, a realização do acesso paralelo foi impossibilitada pelos sistemas de arquivos devido a fatores de configuração dos sistemas que comprometeram os experimentos de acesso paralelo. Nesta seção, será discutida a experiência encontrada nos experimentos e as possíveis razões que podem ter dificultado a realização dos testes.

Através da realização dos experimentos do sistema de arquivos Lustre, notou-se que na escrita paralela em arquivo HDF5, na medida em que o volume de dados ultrapassava aproximadamente 2600 imagens, o processo tornava-se extremamente lenta até que, ou o servidor DICOM congelava todo o sistema operacional, ou a escrita paralela falhava seguidamente. Assim, foi pesquisado em fóruns e lista de e-mails o motivo deste comportamento e verificou-se que o acesso paralelo via MPI E/S no Lustre é um problema conhecido na área cientifica.

Verificou-se que o acesso utilizado pelo ROMIO para escrita em paralela através da técnica *data sieving* não é bem suportado pelo Lustre. Isto ocorre porque a maioria dos sistemas de arquivo são suporta ou não tem configurada (como o caso do Lustre) a característica de suporte a *locking* [1]

Sistemas de arquivos implementam este bloqueio para evitar que o cenário de atualização clássico intercessão de múltiplos acessos, fazendo que a camada do MPI-IO fique estagnada quando mais um cliente escreve no mesmo arquivo.

Para contornar esse problema no Lustre, é recomendando utilizar a *flag "lock"* pelo cliente ao montar a partição do Lustre. Esta flag habilita ou remove caraterística de travamento de acesso concorrente à um arquivo. O *"lock"* disponibiliza dois tipos de trava, *share lock* e *exclusive lock*, que como no nome já diz, o primeiro tipo permite que múltiplos acessos ao mesmo tempo, enquanto o segundo tipo, apenas um processo por vez.

Outra recomendação que algumas listas de e-mails comentam é desabilitar o data sieving do ROMIO e configurar somente o *collective buffer* que também é uma técnica de E/S implementada no ROMIO, além de que esta técnica provê maior suporte pelo Lustre. Buscou-se aplicar esta técnicas nos experimentos do Lustre, porém à medida do aumento do número de imagens armazenadas no sistemas de arquivos, ocorre uma saturação da rede e finalizando com a queda do sistema.

Nos sistemas de arquivos CEPH, notou-se um problema com característica similar a estes derivados de travas de acesso, pois nos experimentos, o PH5wrap trava na escrita do *pixel data* da terceira imagem enviada pelo cliente. No CEPH, quando múltiplos processos invocam a operação de escrita, o sistema bloquea o acesso concorrente e revoga qualquer cache de leitura e *buffer* de escrita que seja invocada anteriormente por outras aplicações, e somente irá liberar a escrita quando for de conhecimento de todos os servidores de dados (OSD).

Está política de acesso pode ser um dos motivos que impossiblitou a escrita concorrente nos experimento, uma vez que para cada acesso paralelo, ocorre um acesso serial por parte do servidor, que pode estar bloqueando a escrita paralela.

Uma possível solução, porém não muito recomendado pelos desenvolvedores do CEPH devido à perda de desempenho, é a utilização da *flag* disponível no CEPH chamada de O_LAZY. Esta *flag* basicamente permite que aplicativos deixem de preocuparem-se com a coerência

[1] *Locking* é um mecanismo que restringe o acesso a um arquivo, permitindo o acesso apenas de um cliente ou processo.

requerida pelo sistema de arquivos na escrita paralela, de tal maneira que o aplicativo controla sua própria consistência de dados.

6 CONCLUSÕES E TRABALHOS FUTUROS

Este projeto apresentou uma nova abordagem para o servidor de imagens médicas DICOM, chamado de CyclopsDCMServer, buscando melhorar o desempenho de armazenamento e recuperação das imagens sob o formato de dados hierárquicos HDF5. Desta maneira, buscou-se aplicar técnicas de escrita e leituras concorrentes e distribuir estes dados via sistemas de arquivos distribuídos, para prover melhor disponibilidade dos processos concorrentes. O mecanismo utilizado para a paralelização da escrita e leitura de dados foi a API de alto desempenho em E/S em disco, chamado ROMIO, que disponibiliza técnicas de E/S concorrentes abrangidas por diversos sistemas de arquivos distribuídos.

Neste trabalho, optou-se por uma arquitetura voltada para realização de escrita e leitura paralela da estrutura *dataset* do HDF5, que contém os dados do *pixel data* provindas de uma imagem DICOM. Foi necessário criar uma aplicação externa encarregada da realização do E/S paralelo, devido à complexidade encontrada no servidor atualmente. O efeito colateral desta arquitetura é esperado devido à comunicação de dados necessário entre o servidor e a aplicação paralela. Toda a interface entre o servidor e a aplicação paralela, as funções de escrita e leitura no arquivo HDF5 foram chamadas de PH5wrap. Além da proposta de uma arquitetura paralela, neste projeto procurou-se aplicar os experimentos em sistemas de arquivos distribuídos, a fim de avaliar o desempenho da aplicação nos sistemas mais conhecidos na área científica (Lustre, PVFS, CEPH, FHGFS).

Foram realizados experimentos em diferentes sistemas de arquivos, em que no decorrer da pesquisa, foi constatado dificuldades para realizar a escrita paralela nos sistemas Lustre e CEPH, devido ambas implantarem em suas configurações, caraterísticas de acesso que dificultaram a realização dos experimentos de escrita paralelo. No entanto, os ambientes PVFS e FhGFS provêm esta compatibilidade do sistema com a escrita paralela fornecida pela interface MPI. Desta maneira, os resultados demonstraram que esta abordagem apresentada não é eficaz para este tipo de problema, por apresentar baixo desempenho na escrita dos dados em paralelo, além de que, o PH5wrap também apresentar um acréscimo de penalidade devido à comunicação de rede adicional.

No entanto, como esta abordagem foi aplicada para uma base de dados reais do ambiente da Telemedicina, o tamanho das imagens não foi adequado para este tipo interface paralela. Para um bom desempenho de E/S, no entanto, imagina-se que o tamanho de um pedido

de E/S deve ser maior (na ordem de megabytes), tendo em vista que o desempenho de E/S sofre consideravelmente com penalidades se as aplicações de acesso a dados fazem pequenos pedidos de E/S. Desta maneira, como trabalho futuro, espera-se estudar esta abordagem em imagens DICOM de tamanhos superiores a estes aplicados nos experimentos.

Verificou-se também que apesar de não proporcionarem por padrão características voltadas para a escrita paralela, ambas os sistemas Lustre e CEPH, demonstraram melhores desempenhos na leitura e escrita serial nos dados, principalmente o sistema CEPH que manteve o melhor desempenho na escrita serial. Em trabalhos futuros, é esperado aplicar as configurações explicadas no capítulo anterior, a fim de corrigir os problemas encontrados e proporcionar mais resultados nesta pesquisa.

A proposta para trabalhos futuros, que possam desempenhar melhores resultados, é buscar melhorar as configurações das tecnologias utilizadas neste trabalho. Como exemplo, a pesquisa de Howison *et al.* (HOWISON et al., 2010a), que é um estudo voltado para otimizar o MPI-IO do HDF5 sob o Lustre. Como também o trabalho do Philip Carns *et al.* (CARNS et al., 2009), que avaliou 5 técnicas para otimizar a escrita de dados pequenos no PVFS. Outra característica que é recomendada nos sistemas de arquivos como o CEPH e o FHGFS é utilizar o XFS como sistema de arquivo local, pois este provê uma melhor escalabilidade e taxa de acesso que os sistemas ext3 e ext4, especialmente para RAID.

Aplicar melhorias que podem desempenhar um ganho de desempenho recomendados pelo grupo HDF e alterar a estrutura do *dataset* utilizada para armazenar o *pixel data*. Desde a implantação do HDF5 no servidor, a estrutura é armazenada em um vetor simples, impossibilitando a utilização de diferentes acessos hyperlabs, técnicas de compactação de dados providos pela API HDF5, como também a realização de armazenamento dos dados diferentes da cor preta do *pixel data* de uma imagem DICOM.

Por fim, como o acesso paralelo em um mesmo *dataset* mostrou-se ser desvantajoso neste determinado ambiente, seria interessante como trabalho futuro realizar estudos de uma abordagem utilizando as API paralelas estudadas nesta abordagem, em que cada processo realiza o armazenamento de um *pixel data* em datasets diferentes, não havendo concorrência e espera dos processos na escrita dos dados.

REFERÊNCIAS

ADELMANN, A. et al. H5part: A portable high performance parallel data interface for particle simulations. In: IEEE. *Particle Accelerator Conference, 2005. PAC 2005. Proceedings of the*. [S.l.], 2006. p. 4129–4131. ISBN 0780388593.

AFS Frequentelly Asked Questions. July 1998. Acessado em Maio de 2012. Disponível em. <http://www.angelfire.com/hi/plutonic/afs-faq.html>.

BASHSHUR, R. Chapter 1: Telemedicine and health care. *Telemedicine Journal and e-health*, Mary Ann Liebert, Inc., v. 8, n. 1, p. 5–12, 2002.

BOITO, F. et al. I/o performance of a large atmospheric model using pvfs. *Rencontres francophones du Parallélisme (RenPar'20)*, 2011.

BRAAM, P. Lustre networking. *A White Paper from Cluster File System, Inc*, 2007.

CARNS, P. et al. Small-file access in parallel file systems. In: IEEE. *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*. [S.l.], 2009. p. 1–11.

CARNS, P. H. et al. Pvfs: a parallel file system for linux clusters. In: *Proceedings of the 4th annual Linux Showcase & Conference - Volume 4*. Berkeley, CA, USA: USENIX Association, 2000. (ALS'00), p. 28–28. <http://dl.acm.org/citation.cfm?id=1268379.1268407>.

COHEN, S. et al. *Scientific formats for object-relational database systems: a study of suitability and performance*. [S.l.]: ACM, 2006. 10–15 p.

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. *Sistemas distribuídos-conceitos e projeto*. [S.l.]: Bookman, 2007.

CYCLOPS. *Cyclops Project*. 2011. Acessado em Janeiro de 2011. Disponível em. <http://www.cyclops.ufsc.br>.

DANTAS, M. *Computação distribuída de alto desempenho: redes, clusters e grids computacionais*. [S.l.]: Axcel Books, 2005.

DENNIS, J. et al. An application-level parallel i/o library for earth system models. *International Journal of High Performance Computing Applications*, SAGE Publications, v. 26, n. 1, p. 43–53, 2012.

DIVISION, M. *ROMIO: A High-Performance, Portable MPI-IO Implementation*. 2011. Acessado em Março de 2011. Disponível em. <http://www.mcs.anl.gov/research/projects/romio/>.

DOUGHERTY, M. T. et al. Unifying biological image formats with hdf5. *Commun. ACM*, ACM, New York, NY, USA, v. 52, n. 10, p. 42–47, out. 2009. ISSN 0001-0782. <http://doi.acm.org/10.1145/1562764.1562781>.

FAHEY, M.; LARKIN, J.; ADAMS, J. I/o performance on a massively parallel cray xt3/xt4. In: IEEE. *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*. [S.l.], 2008. p. 1–12.

FS, F. *FraunhoferFS - User Guide*. 2012. Acessado em Maio de 2012. Disponível em. <http://www.fhgfs.com/wiki/wikka.php?wakka=UserGuide>.

GROPP, W. et al. A high-performance, portable implementation of the mpi message passing interface standard. *Parallel computing*, Elsevier, v. 22, n. 6, p. 789–828, 1996.

HAATAJA, J. *Universal Rule*. 2011. Acessado em Novembro de 2011. Disponível em. <http://radio-weblogs.com/0112083/images/2002/08/22/upshotfig.gif>.

HDFGROUP. *The HDF Group*. 2011. Acessado em Janeiro de 2011. Disponível em. <http://www.hdfgroup.org>.

HOWARD, J.; CENTER, C.-M. U. I. T. *An overview of the andrew file system*. Carnegie Mellon University, Information Technology Center, 1988. <http://reports-archive.adm.cs.cmu.edu/anon/anon/home/ftp/itc/CMU-ITC-062.pdf>.

HOWISON, M. et al. H5hut: A high-performance I/O library for particle-based simulations. In: IEEE. *Cluster Computing Workshops and Posters (CLUSTER WORKSHOPS), 2010 IEEE International Conference on*. [S.l.], 2010. p. 1–8.

HOWISON, M. et al. Tuning hdf5 for lustre file systems. In: *Workshop on Interfaces and Abstractions for Scientific Data Storage*. [S.l.: s.n.], 2010.

HUANG, H. *PACS: basic principles and applications*. [S.l.]: Wiley, 1999. 521 p. ISBN 0471253936.

KASSICK, R. Reconfiguração automática de i/o para aplicações paralelas no sistema de arquivos dnfsp2. Universidade Federal do Rio Grande do Sul. Instituto de Informática. Programa de Pós-Graduação em Computação., 2010.

KASSICK, R.; BOITO, F.; NAVAUX, P. Dynamic i/o reconfiguration for a nfs-based parallel file system. In: *Parallel, Distributed and Network-Based Processing (PDP), 2011 19th Euromicro International Conference on*. [S.l.: s.n.], 2011. p. 11 –18. ISSN 1066-6192.

KUHN, M.; KUNKEL, J. M.; LUDWIG, T. Dynamic file system semantics to enable metadata optimizations in pvfs. *Concurrency and Computation: Practice and Experience*, John Wiley & Sons, Ltd., v. 21, n. 14, p. 1775–1788, 2009. ISSN 1532-0634. <http://dx.doi.org/10.1002/cpe.1439>.

LAGHAVE, N. et al. Benefits of parallel i/o in ab initio nuclear physics calculations. In: *Proceedings of the 9th International Conference on Computational Science: Part I*. Berlin, Heidelberg: Springer-Verlag, 2009. (ICCS '09), p. 84–93. ISBN 978-3-642-01969-2.

LAROS, J. et al. Red storm io performance analysis. In: IEEE. *Cluster Computing, 2007 IEEE International Conference on*. [S.l.], 2007. p. 50–57.

LOMBARD, P.; DENNEULIN, Y. nfsp: a distributed nfs server for clusters of workstations. In: *Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM*. [S.l.: s.n.], 2002. p. 35 –40.

MACEDO, D. D. et al. An architecture for dicom medical images storage and retrieval adopting distributed file systems. *International Journal of High Performance Systems Architecture*, Inderscience, v. 2, n. 2, p. 99–106, 2009. ISSN 1751-6528.

MAIA, R.; WANGENHEIM, A. von; NOBRE, L. A statewide telemedicine network for public health in brazil. In: IEEE.

Computer-Based Medical Systems, 2006. CBMS 2006. 19th IEEE International Symposium on. [S.l.], 2006. p. 495–500.

MARTIN, R.; CULLER, D. Nfs sensitivity to high performance networks. In: ACM. *ACM SIGMETRICS Performance Evaluation Review*. [S.l.], 1999. v. 27, n. 1, p. 71–82.

NAM, B.; SUSSMAN, A. Improving access to multi-dimensional self-describing scientific datasets. In: *Cluster Computing and the Grid, 2003. Proceedings. CCGrid 2003. 3rd IEEE/ACM International Symposium on*. [S.l.: s.n.], 2003. p. 172 – 179.

NIETO, E. et al. Fault tolerant pvfs2 based on data replication. In: *Parallel Distributed and Grid Computing (PDGC), 2010 1st International Conference on*. [S.l.: s.n.], 2010. p. 107 –112.

NOWICKI, B. *Nfs: Network file system protocol specification*. [S.l.]: Sun Microsystems, Incorporated, 1989.

OMNIBOND. *OrangeFS*. 2011. Acessado em Março de 2011. Disponível em. <http://www.orangefs.org>.

PACHECO, P. *Parallel programming with MPI*. [S.l.]: Morgan Kaufmann, 1997.

PAWLOWSKI, B. et al. Nfs version 3 design and implementation. In: *Proceedings of the Summer 1994 USENIX Technical Conference*. [S.l.: s.n.], 1994. p. 137–151.

PIANYKH, O. *Digital Imaging and Communications in Medicine (DICOM): A practical introduction and survival guide*. [S.l.]: Springer Verlag, 2008. ISBN 354074570X.

PRADO, T. C. *Otimização da persistência de dados em pacs empregando modelos de dados hierárquicos indexados*. Dissertação (Mestrado) — Universidade Federal de Santa Catarina, June 2012.

PRAKASH, A. *Parallel Virtual Machine*. 2011. Acessado em Outubro de 2011. Disponível em. <http://dspace.cusat.ac.in/xmlui/handle/123456789/2012>.

SCHWAN, P. Lustre: Building a file system for 1000-node clusters. In: *Proceedings of the 2003 Linux Symposium*. [S.l.: s.n.], 2003. p. 400–407.

SHEPLER, S. et al. Network file system (nfs) version 4 protocol. *Network*, 2003.

SHVACHKO, K. et al. The hadoop distributed file system. In: *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*. [S.l.: s.n.], 2010. p. 1 –10.

SOARES, T. et al. A parallel architecture using hdf for storing dicom medical images on distributed file systems. *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'11)*, part of WORLDCOMP'11, p. 42–47, 2011. Sunnyvale, California.

SOARES, T. S. et al. An approach using parallel architecture to storage dicom images in distributed file system. *Journal of Physics: Conference Series*, v. 341, n. 1, p. 012021, 2012. <http://stacks.iop.org/1742-6596/341/i=1/a=012021>.

SOURCEFORGE. *CLucene*. 2011. Acessado em Janeiro de 2011. Disponível em. <http://clucene.sourceforge.net>.

STEEN, M.; TANENBAUM, A. *Distributed Systems: Principles and Paradigms, 2/E*. [S.l.]: Prentice Hall, 2007.

THORBECKE, J. *I/O Optimization*. 2012. Acessado em Maio de 2012. Disponível em. <http://user.cscs.ch/fileadmin/user-upload/customers/CSCS-Application-Data/Files/Presentations/Courses-Ws-2011/Cray-XE6-Workshop/09-CSCS–IO- Optimization.pdf>.

TOP500. 2011. Acessado em Março de 2012. Disponível em. <www.top500.org>.

ÁVILA, R. B. *Uma Proposta de Distribuição do Servidor de Arquivos em Clusters*. Dissertação (Mestrado) — Universidade Federal do Rio Grande do Sul, 2005.

WALLAUER, J. et al. A telemedicine network using secure techniques and intelligent user access control. In: IEEE. *21st IEEE International Symposium on Computer-Based Medical Systems*. [S.l.], 2008. p. 105–107.

WEIL, S. A. et al. Ceph: a scalable, high-performance distributed file system. In: *Proceedings of the 7th symposium on Operating systems design and implementation*. Berkeley, CA, USA: USENIX

Association, 2006. (OSDI '06), p. 307–320. ISBN 1-931971-47-1. <http://dl.acm.org/citation.cfm?id=1298455.1298485>.

WIKIPEDIA. *Lustre file system*. 2012. Acessado em Abril de 2012. Disponível em. <http://en.wikipedia.org/wiki/Lustre_(file_system)>.

YU, W.; VETTER, J.; ORAL, H. Performance characterization and optimization of parallel i/o on the cray xt. In: IEEE. *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*. [S.l.], 2008. p. 1–11.

ZHU, Y. et al. Improved read performance in a cost-effective, fault-tolerant parallel virtual file system (ceft-pv*O_Lazy*). In: *Cluster Computing and the Grid, 2003. Proceedings. CCGrid 2003. 3rd IEEE/ACM International Symposium on*. [S.l.: s.n.], 2003. p. 730 – 735.

APÊNDICE A – Trabalhos Publicados

A.1 PUBLICAÇÕES

- **Título:** *A Parallel Architecture Using HDF for Storing DICOM Medical Images on Distributed File Systems*

- **Evento:** PDPTA 2011 - The 2011 International Conference on Parallel and Distributed Processing Techniques and Applications

- **Autores:** Tiago Steinmetz Soares, Douglas D.J. de Macedo, M.A.R Dantas, Michael A. Bauer

- **Resumo:** The Hierarchical Data Format (HDF) is an interesting approach for developing scientific applications where a large amount of data must be stored and accessed. A telemedicine project underway in the State of Santa Catarina (SC), in Brazil, has developed a server called the CyclopsDCMServer, which adopts the HDF for the manipulation of medical images (DICOM). This paper proposes a new approach for the parallel implementation of I/O operations for the medical images stored on this server. This effort was based upon the MPI paradigm that is supported by the version 5 of the HDF. Early experiments indicate that the proposed approach can achieve very good performance when compared to the standard HDF implemented in the CyclopsDCMServer.

- **Estrato:** B3

- **Título:** *An Approach Using Parallel Architecture to Storage DICOM Images in Distributed File System*

- **Evento:** HPCS 2011 - The 2011 International Conference on High Performance Computing and Simulation

- **Autores:** Tiago S Soares, Thiago C Prado, M A R Dantas, Douglas D J de Macedo e Michael A Bauer

- **Resumo:** Telemedicine is a very important area in medical field that is expanding daily motivated by many researchers interested in improving medical applications. In Brazil was started in 2005, in the State of Santa Catarina has a developed server called the CyclopsDCMServer, which the purpose to embrace the HDF for the manipulation of medical images (DICOM) using a distributed file system. Since then, many researches were initiated in

order to seek better performance. Our approach for this server represents an additional parallel implementation in I/O operations since HDF version 5 has an essential feature for our work which supports parallel I/O, based upon the MPI paradigm. Early experiments using four parallel nodes, provide good performance when compare to the serial HDF implemented in the Cyclops-DCMServer

- **Estrato:** B2