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

DIEGO DA SILVA GOMES

JavaRMS: um Sistema de Gerência de Dados para Grades Baseado num Modelo Par-a-Par

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

Prof. Dr. Cláudio Fernando Resin Geyer
Orientador

Porto Alegre, Abril de 2008

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Gomes, Diego da Silva

JavaRMS: um Sistema de Gerência de Dados para Grades Baseado num Modelo Par-a-Par / Diego da Silva Gomes. – Porto Alegre: PPGC da UFRGS, 2008.

115 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2008. Orientador: Cláudio Fernando Resin Geyer.

1. Computação em grade. 2. Gerência de dados. 3. Modelo par-a-par. 4. Replicação. 5. Arquivamento. I. Geyer, Cláudio Fernando Resin. II. Título.

“A teoria sempre acaba, mais cedo ou mais tarde,
assassinada pela experiência.”
— ALBERT EINSTEIN

AGRADECIMENTOS

Os agradecimentos, em ordem aleatória:

- Ao meu orientador durante parte do mestrado, professor Geyer, por ter me mostrado um caminho a seguir quando pensava em desistir, por partilhar sua experiência, pelos diversos conselhos dados no decorrer dessa etapa de minha vida e pelas oportunidades oferecidas;
- Aos integrantes do grupo de pesquisa, em especial ao Marko Petek, pela ajuda técnica, pelos conselhos e pelo incentivo moral;
- Ao pessoal das salas 205 e 207, Éder, Gustavo, João, Luciano, Sônia, Eduardo e Émerson, pelo companheirismo dos almoços no RU, pela troca de idéias e pela ajuda técnica;
- Aos meus pais e também professores, Odilon e Elisabete, por me mostrar os caminhos da vida que me permitiram chegar até aqui, pela educação que foi a mim concedida, pelo apoio afetivo e, é claro, pelo suporte financeiro;
- Ao meu irmão Samuel, à minha irmã Daniele, à Rosângela e à toda minha família que mora em Rio Grande, pela torcida por mim.
- À minha tia Rosa e toda a família que mora em Porto Alegre, pelos almoços, cafés, aniversários, conversas e por todo o apoio;
- À minha querida Lisiane, pelo carinho, pelas risadas e pela paciência;
- Aos Paretos, Rafael, Alex, Paulo e Bruno, pela sua imensa amizade e ajuda. Um agradecimento especial ao Rafael, por partilhar as mesmas angústias do mestrado e, juntamente com a Giseli e o Leandro, por compartilharem comigo o mesmo lar durante esses anos;
- À CAPES e ao CNPQ, pelo auxílio financeiro durante a maior parte do tempo do mestrado.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

IDA	*Information Dispersal Algorithm* (Algoritmo de Dispersão de Informação)
P2P	*Peer-to-Peer* (Par-a-Par)
RMS	*Replica Management System* (Sistema de Gerenciamento de Réplicas)
RLS	*Replica Location Service* (Serviço de Localização de Réplicas)
LHC	*Large Hadron Collider*
CMS	*Compact Muon Solenoid*
HEP	*High Energy Physics* (Física de Altas Energias)
Cern	Organização Européia para Pesquisa Nuclear
DHT	*Distributed Hash Table* (Tabela Hash Distribuída)
GPPD	Grupo de Processamento Paralelo e Distribuído
CA	*Certificate Authority* (Autoridade Certificadora)
GSI	*Globus Security Infrastructure* (Infra-estrutura de Segurança do Globus)
LFN	*Logical File Name* (Nome de Arquivo Lógico)
PFN	*Physical File Name* (Nome de Arquivo Físico)
DN	*Distinguished Name* (Nome do Usuário ou Máquina na Grade)
TLS	*Transport Layer Security* (Segurança da Camada de Transporte)
SSL	*Secure Sockets Layer* (Camada de *Sockets* Seguros)

LISTA DE FIGURAS

RESUMO

A grande demanda por computação de alto desempenho culminou na construção de ambientes de execução de larga escala como as Grades Computacionais. Não diferente de outras plataformas de execução, seus usuários precisam obter os dados de entrada para suas aplicações e muitas vezes precisam armazenar os resultados por elas gerados. Apesar de o termo Grade ter surgido de uma metáfora onde os recursos computacionais estão tão facilmente acessíveis como os da rede elétrica, as ferramentas para gerenciamento de dados e de recursos de armazenamento disponíveis estão muito aquém do necessário para concretizar essa idéia. A imaturidade desses serviços se torna crítica para aplicações científicas que necessitam processar grandes volumes de dados. Nesses casos, utiliza-se apenas os recursos de alto desempenho e assegura-se confiabilidade, disponibilidade e segurança para os dados através de presença humana.

Este trabalho apresenta o JavaRMS, um sistema de gerência de dados para Grades. Ao empregar um modelo par-a-par, consegue-se agregar os recursos menos capacitados disponíveis no ambiente de Grade, diminuindo-se assim o custo da solução. O sistema utiliza a técnica de nodos virtuais para lidar com a grande heterogeneidade de recursos, distribuindo os dados de acordo com o espaço de armazenamento fornecido. Emprega-se fragmentação para viabilizar o uso dos recursos menos capacitados e para melhorar o desempenho das operações que envolvem a transferência de arquivos. Utiliza-se replicação para prover persistência aos dados e para melhorar sua disponibilidade. JavaRMS lida ainda com a dinamicidade e a instabilidade dos recursos através de um modelo de estados, de forma a diminuir o impacto das operações de manutenção. A arquitetura contempla também serviços para gerenciamento de usuários e protege os recursos contra fraudes através de um sistema de cotas. Todas as operações foram projetadas para serem seguras. Por fim, disponibiliza-se toda a infra-estrutura necessária para que serviços de busca e ferramentas de interação com o usuário sejam futuramente fornecidos.

Os experimentos realizados com o protótipo do JavaRMS comprovam que usar um modelo par-a-par para organizar os recursos e localizar os dados resulta em boa escalabilidade. Já a técnica de nodos virtuais se mostrou eficiente para distribuir de forma balanceada os dados entre as máquinas, de acordo com a capacidade de armazenamento oferecida. Através de testes com a principal operação que envolve a transferência de arquivos, comprovou-se que o modelo é capaz de melhorar significativamente o desempenho de aplicações que necessitam processar grandes volumes de dados.

Palavras-chave: Computação em grade, gerência de dados, modelo par-a-par, replicação, arquivamento.

JavaRMS: a Grid Data Management System based on a Peer-to-Peer Model

ABSTRACT

Large scale execution environments such as Grids emerged to meet high-performance computing demands. Like in other execution platforms, its users need to get input data to their applications and to store their results. Although the Grid term is a metaphor where computing resources are so easily accessible as those from the eletric grid, its data and resource management tools are not sufficiently mature to make this idea a reality. They usually target high-performance resources, where data reliability, availability and security is assured through human presence. It turns to be critical when scientific applications need to process huge amounts of data.

This work presents JavaRMS, a Grid data management system. By using a peer-to-peer model, it aggregates low capacity resources to reduce storage costs. Resource heterogeneity is dealt with the virtual node technique, where peers receive data proportionally to their provided storage space. It applies fragmentation to make feasible the usage of low capacity resources and to improve file transfer operations performance. Also, the system achieves data persistence and availability through replication. In order to decrease the impact of maintenance operations, JavaRMS deals with resource dinamicity and instability with a state model. The architecture also contains user management services and protects resources through a quota system. All operations are designed to be secure. Finally, it provides the necessary infrastructure for further deployment of search services and user interactive tools.

Experiments with the JavaRMS prototype showed that using a peer-to-peer model for resource organization and data location results in good scalability. Also, the virtual node technique showed to be efficient to provide heterogeneity-aware data distribution. Tests with the main file transfer operation proved the model can significantly improve data-intensive applications performance.

Keywords: grid computing, data management, peer-to-peer model, replication, archiving.

1 INTRODUÇÃO

Desde os primórdios da computação existe a necessidade de fornecer dados de entrada para os programas e obter dados de saída. Mesmo nos sistemas computacionais que eram ausentes de memória na forma como é hoje conhecida, existiam mecanismos para os usuários assim interagirem com a máquina computacional a eles disponível. Nos computadores que utilizavam cartões perfurados com esse objetivo, por exemplo, era comum o usuário ter que preparar e organizar uma grande quantidade de cartões, muitos dos quais eram guardados em ficheiros para posterior reutilização ou análise. Na medida que a quantidade de cartões aumentava, os usuários precisavam se valer de grandes habilidades de gerenciamento para encontrar seus dados (cartões!) e saber em qual ficheiro colocar cada cartão produzido.

Quando surgiram as primeiras tecnologias para armazenamento, um grande volume de dados anteriormente guardados em cartões pôde ser facilmente colocado em um único disquete ou fita magnética, viabilizando inclusive o transporte desses dados entre diferentes computadores. Entretanto, as necessidades de gerenciamento continuaram existindo: ao invés de organizar cartões, os usuários passaram a organizar arquivos. Conforme os sistemas computacionais evoluíam, possibilitava-se que um volume cada vez maior de arquivos fosse gerado. Esses arquivos poderiam inclusive ocupar mais de uma unidade de armazenamento, fazendo com que o usuário precisasse gerenciar também um grande número de disquetes ou fitas, nada muito diferente do que fazia com os cartões. Mesmo com o crescimento da capacidade das unidades de armazenamento e o esvaziamento das pilhas de disquetes, o problema de organizar os arquivos continuou existindo, apesar da evolução dos sistemas de gerenciamento de arquivos. A capacidade de armazenamento também não era suficiente para suprir as necessidades cada vez maiores dos usuários que, por sua vez, também tinham a necessidade de mover esses dados de um computador para outro.

O advento das redes de computadores facilitou a movimentação de dados, porém a dificuldade de obtê-los se tornou ainda maior: os usuários passaram a ter que organizá-los em múltiplas máquinas. Essas redes evoluíram e viabilizaram os primeiros sistemas de arquivos distribuídos que, apesar de terem facilitado a vida dos usuários, já apresentavam dificuldades para fornecer as mesmas abstrações dos sistemas de arquivos convencionais. A evolução das redes também permitiu que elas se tornassem globalizadas, dando surgimento aos sistemas largamente distribuídos de hoje. As Grades (FOSTER; KESSELMAN, 1999) e os sistemas Par-a-Par (P2P) (ANDROUTSELLIS-THEOTOKIS; SPINELLIS, 2004; COULOURIS; DOLLIMORE; KINDBERG, 2005) são exemplos desses ambientes, que surgem como uma tentativa de atender à demanda cada vez maior por armazenamento, processamento e transporte de dados. Não diferente de outros momentos da história, seus usuários necessitam localizar e obter os dados de entrada para suas

aplicações. Entretanto, fornecer abstrações de sistemas de arquivos se tornou ainda mais difícil, ou mesmo inadequado, pois além de a infra-estrutura de rede ainda não atender à demanda por transporte, o ambiente computacional se tornou heterogêneo, dinâmico e não confiável (ROWSTRON; DRUSCHEL, 2001a; ADYA et al., 2002). Especialmente no caso das aplicações que precisam processar grandes volumes de dados, torna-se crítico fazer o gerenciamento de dados de forma eficiente para obter um bom desempenho (BUNN; NEWMAN, 2003).

O tema desta dissertação, a modelagem e implementação de um sistema de gerência de dados para Grades, surge da necessidade de viabilizar a concepção dessas aplicações que lidam com grandes quantidades de dados em ambientes largamente distribuídos. Segundo outros autores (CAMARGO; KON, 2007; BUNN; NEWMAN, 2003), fornecer uma solução para esse problema é uma tarefa difícil devido à complexidade imposta por esse ambiente. Mesmo que novas tecnologias de armazenamento, processamento e transporte de dados venham a ser inventadas pelo homem e tornem as Grades obsoletas, novas aplicações surgirão e precisarão de quantidades ainda maiores de dados. Espera-se, contudo, que as idéias apresentadas neste trabalho possam continuar sendo úteis em soluções futuras.

1.1 Motivação

Os primeiros *middlewares* de Grade disponibilizavam para as aplicações apenas um conjunto mínimo de ferramentas usadas para movimentar arquivos entre as máquinas da Grade. Essas ferramentas (ex.: *bbcp* (HANUSHEVSKY; TRUNOV; COTTRELL, 2001) e *GridFTP* (ALLCOCK et al., 2005)) foram especialmente desenvolvidas para redes de grandes distâncias, típicas desse ambiente. Conforme o volume de dados aumentava, os usuários começavam a ter dificuldades para gerenciar seus dados. Logo surgiram ferramentas para catalogar os arquivos e suas réplicas (cópias), tais como o *Giggle* (CHERVENAK et al., 2002) e o *Replica Location Service* (RLS) do *toolkit* Globus (FOSTER; KESSELMAN, 1997). O objetivo era facilitar a localização dos dados por parte dos usuários, embora os recursos de armazenamento continuassem sendo gerenciados manualmente, ou seja, os usuários é quem deveriam decidir onde e como os dados seriam armazenados. Essas primeiras ferramentas implementavam catálogos centralizados, ou mesmo hierárquicos, que apresentavam escalabilidade limitada (CAI; CHERVENAK; FRANK, 2004; CHERVENAK et al., 2004; RIPEANU; FOSTER, 2002). Com o tempo, surgiram as soluções de catalogação baseadas em modelos P2P, que tinham como característica a boa escalabilidade (suportavam eficientemente grandes quantidades de requisições e podiam catalogar grandes quantidades de arquivos). Entretanto, essas ferramentas continuavam não dando nenhuma garantia quanto à consistência, persistência e segurança das réplicas, assim como continuavam não gerenciando os recursos de armazenamento da Grade. Simultaneamente, adicionou-se aos *toolkits* de Grade ferramentas para automatizar o registro dos novos arquivos criados pelos usuários (ex.: *Data Replication Service* do Globus e o *Replica Manager* do EU DataGrid Work Package 2 (CAMERON et al., 2004)) e para dar persistência às transferências (ex.: *Reliable File Transfer* (ALLCOCK; FOSTER; MADDURI, 2004) do Globus). Juntos, esse conjunto de ferramentas para transferir e catalogar os dados formavam os chamados Sistemas de Gerenciamento de Réplicas (*Replica Management Systems* – RMS).

Apesar de o termo Grade ter surgido de uma metáfora onde os recursos computacionais são acessados tão facilmente quanto os recursos de energia da rede elétrica, o

suporte ao gerenciamento de dados e ao gerenciamento de recursos está muito aquém do que uma grande classe de aplicações precisaria para concretizar essa idéia. De fato, os RMS geralmente têm como alvo plataformas computacionais de alto desempenho com aplicações que precisam processar massivas quantidades de dados (na ordem de *PetaBytes* – $1PB = 10^{15}$ *Bytes*) e que são compostas por supercomputadores interligados por redes de altíssima velocidade (CAMARGO; KON, 2007). Essas ferramentas rodam em servidores dedicados dessa infra-estrutura onde a confiabilidade, disponibilidade e segurança dos serviços recaem sobre recursos humanos (ADYA et al., 2002). Os dados são ainda armazenados em grandes sistemas de armazenamento baseados em discos e fitas que possuem um altíssimo custo associado.

A plataforma-alvo dos RMS está disponível apenas para instituições que podem arcar com seu alto custo (tanto em recursos computacionais quanto em humanos). Por outro lado, ambientes largamente distribuídos como os de Grade possuem uma grande quantidade de recursos ociosos. Esses recursos poderiam ser utilizados por diversas aplicações, tanto para processar quanto para armazenar os dados. Laboratórios formados por estações de trabalho, pequenos *clusters*, ou mesmo *desktops* dos usuários que precisam dessas aplicações são exemplos de recursos que constituem uma plataforma de execução com grande potencial. Combinando-se o espaço livre em disco de algumas centenas de máquinas é possível obter muitos *TeraBytes* de espaço de armazenamento distribuído. Usar essas máquinas para armazenar e gerenciar os dados melhoraria a utilização dos recursos e permitiria soluções de baixo custo para o gerenciamento de dados em instituições com orçamento limitado, tais como aquelas típicas de países em desenvolvimento (CAMARGO; KON, 2007). Até mesmo as aplicações que lidam com grandes volumes de dados (e que hoje versam em uma plataforma de alto custo) poderiam utilizar esses serviços a fim de melhorar seu desempenho e o acesso aos dados, pois estes ficariam mais próximos dos usuários que desejam processá-los e analisá-los. O armazenamento agregado permitiria também utilizar uma maior quantidade de réplicas, melhorando a disponibilidade desses dados.

No entanto, agregar esses novos recursos representa um desafio para uma solução de gerenciamento de dados. Além de não serem confiáveis, essas máquinas são freqüentemente desligadas ou reiniciadas e seus administradores podem querer compartilhá-las apenas quando estiverem ociosas ou segundo políticas próprias de uso (CAMARGO; KON, 2007). Diante desse ambiente altamente dinâmico e instável, ainda precisa-se dar garantias quanto à disponibilidade dos dados gerenciados. O *hardware* também se torna muito mais heterogêneo do que aquele encontrado nos grandes centros de computação, exigindo mecanismos eficientes de distribuição dos dados, aliados a outros para balanceamento de carga. Por fim, uma solução nesse sentido deve utilizar protocolos seguros e que escalem para a grande quantidade de recursos que poderá ser agregada.

Muitas dessas características já são atendidas por outros sistemas que utilizam o modelo P2P de comunicação (ex.: CFS (DABEK et al., 2001), PAST (DRUSCHEL; ROWSTRON, 2001), OceanStore (RHEA et al., 2003) e FreeLoader (VAZHKUDAI et al., 2005)). Diferentemente do paradigma de Grade, esses sistemas lidam com o auto-gerenciamento dos recursos, são naturalmente descentralizados e executam suas funcionalidades através de protocolos tolerantes a falhas (HASAN et al., 2005). Entretanto, para serem utilizadas para o gerenciamento de dados num ambiente de Grade, essas soluções precisam ser essencialmente seguras e devem poder ser integradas com outros serviços, tais como os de escalonamento de aplicações.

Apesar de esses aspectos já serem suficientes para que uma nova solução de gerenci-

amento de dados fosse concebida, foram outros aspectos que motivaram a concepção do JavaRMS neste trabalho. Caso fossem integrados às Grades que dão suporte às aplicações que processam grandes volumes de dados, essas soluções falhariam no gerenciamento dos dados e dos recursos de armazenamento. No novo ambiente que se forma, necessita-se de mecanismos para aproveitar tanto os recursos de baixa capacidade (como os *desktops* com baixa conectividade) quanto aqueles de alta capacidade e valor agregado, como os dispositivos de armazenamento em massa. Devido à necessidade de lidar com grandes quantidades de dados, precisa-se ainda que o sistema seja projetado para minimizar custos de operações que movimentam arquivos.

Por fim, este trabalho também foi motivado pela identificação da necessidade por serviços de gerenciamento de dados mais abstratos e voltados ao usuário final, aliados a mecanismos de busca mais sofisticados. Somente quando esses aspectos forem tratados é que os usuários terão a sensação de estarem usando os recursos computacionais de forma análoga àqueles da rede elétrica.

1.2 Objetivos

O objetivo geral deste trabalho é projetar e implementar um sistema de gerenciamento de dados capaz de melhorar o desempenho de aplicações que necessitam acessar grandes quantidades de dados em ambientes heterogêneos, dinâmicos e de larga escala como os de Grade, ainda que atendendo à exigência por segurança e fornecendo persistência aos dados gerenciados. Em torno desse objetivo geral, definiu-se os seguintes objetivos específicos:

- Fornecer uma mídia de armazenamento global e de fácil acesso para viabilizar o compartilhamento de dados entre os usuários e suas aplicações;
- Agregar recursos com facilidade a fim de aumentar a capacidade global de armazenamento, porém obedecendo às políticas de uso estabelecidas pelos administradores desses recursos;
- Facilitar a localização dos dados no ambiente de Grade, fornecendo bases para que mecanismos de busca sofisticados possam ser concebidos;
- Disponibilizar um protótipo do sistema que seja capaz de interoperar com os demais serviços da Grade;
- Permitir que os usuários tenham vantagens de armazenamento conforme eles explicitem suas exigências quanto ao desempenho no acesso aos dados, quanto à disponibilidade desejada para esses dados e quanto ao custo de armazenamento que estão dispostos a pagar.

1.3 Contexto de Pesquisa

Este trabalho está inserido no contexto do desenvolvimento de *software* de Grade para dar suporte aos mais recentes experimentos da área da Física de Altas Energias (*High Energy Physics* – HEP). Esses experimentos, atualmente em construção na Organização Européia para Pesquisa Nuclear (Cern), tem o objetivo de investigar novos processos físicos que comprovariam grandes teorias sobre a origem da matéria e do universo (BUNN;

NEWMAN, 2003). Mais especificamente, o acelerador de partículas *Large Hadron Collider* (LHC) (LHC: THE LARGE HADRON COLLIDER, 2008) está sendo ampliado para comportar quatro grandes experimentos que produzirão massivas quantidades de dados durante seus 15 anos de operação previstos. Somente em um deles, o detector de partículas *Compact Muon Solenoid* (CMS) (NEWBOLD, 1994), serão produzidos cerca de 5 *Petabytes* (1 *Petabyte* = 10^{15} *Bytes*) de dados anualmente. É previsto que, já para a próxima década, tenha-se atingido a ordem de *Exabytes* de dados produzidos. Cerca de 2 mil físicos de 150 instituições em mais de 30 países estão envolvidos nesse projeto.

O Grupo de Processamento Paralelo e Distribuído (GPPD) da Universidade Federal do Rio Grande do Sul (UFRGS), através da Universidade Estadual do Rio de Janeiro (UERJ), coopera com o subgrupo de Caltech que desenvolve *software* de Grade para o experimento CMS. O *Clarens* (STEENBERG et al., 2004), como é chamado o *middleware* de Grade desenvolvido por esse subgrupo, carece de soluções para diversos problemas relacionados ao gerenciamento dos dados. Juntamente com o trabalho do doutorando Marko Petek, membro do grupo de pesquisa do GPPD, pretende-se fornecer uma solução para o gerenciamento de dados em Grades que processam grandes volumes de dados. Ao viabilizar o uso de recursos de menor custo na solução, objetiva-se uma maior participação nacional nessas mais recentes pesquisas da Física de Altas Energias.

A intenção deste trabalho nesse contexto de pesquisa é desenvolver o núcleo (componentes essenciais) de uma arquitetura para a gerência de dados projetada para ser escalável, interoperável e segura. A concepção da arquitetura em si foi realizada em conjunto com o doutorando Marko Petek, como resultado da evolução da proposta feita inicialmente pelo grupo (PETEK et al., 2006).

1.4 Principais Contribuições

As principais contribuições esperadas com este trabalho são:

- Modelagem e prototipação de um sistema para o gerenciamento de grandes quantidades de dados que seja escalável, seguro, interoperável e que dê garantias de persistência e disponibilidade para arquivos e seus atributos em um ambiente de Grade;

- Elaboração e validação de uma estratégia de distribuição dos dados baseada em um modelo P2P que permita diminuir custos de operações que envolvem a movimentação de grandes quantidades de dados;

- Definição de serviços para o tratamento da heterogeneidade de recursos que permitam agregar máquinas não confiáveis e mais instáveis junto ao ambiente de Grade;

- Validação de uma arquitetura orientada a serviços que seja interoperável com as demais ferramentas de Grade e que permita integrar a solução aqui proposta com *middlewares* de Grade com facilidade, tais como o *Clarens*.

1.5 Organização do Texto

O texto deste trabalho está organizado em outros seis capítulos e um anexo, descritos a seguir:

- O segundo capítulo traz a revisão bibliográfica, contextualizando o leitor quanto aos aspectos envolvidos com o gerenciamento de dados. Caracterizam-se ambientes típicos de Grade e de sistemas P2P quanto aos recursos, dados e usuários, mostrando-se as exigências feitas por aplicações que versam sobre eles;
- O capítulo seguinte resume o estado da arte sobre o tema deste trabalho e aponta os principais aspectos não atacados pelos trabalhos relacionados a fim de justificar a concepção de uma nova solução;
- No capítulo quatro, descreve-se o modelo proposto como solução para o gerenciamento de dados que atende aos requisitos para Grades como a do HEP;
- Na sequência, o capítulo cinco descreve o protótipo, mostrando quais das funcionalidades foram implementadas, os mecanismos de programação por elas utilizados e que diferenças apresentam quanto às funcionalidades esperadas pelo modelo;
- O capítulo seis apresenta os experimentos realizados para validar o modelo proposto, analisando-se aspectos relativos à escala, ao desempenho das principais operações, e à distribuição dos dados;
- No sétimo e último capítulo, são feitas considerações finais e apontadas sugestões para a continuação do trabalho;
- Por fim, o anexo apresenta uma descrição do algoritmo Kademlia, que é usado para estruturar a rede P2P.

2 EM DIREÇÃO AOS SISTEMAS DE GERÊNCIA DE DADOS EM AMBIENTES LARGAMENTE DISTRIBUÍDOS

2.1 Evolução dos Sistemas de Gerência de Arquivos

Um **arquivo** é uma abstração básica para lidar com dados que surgiu da necessidade de armazená-los. Além de fazer o mapeamento dos arquivos para dispositivos de armazenamento, um **sistema de gerência de arquivos** pode ter diversos outros objetivos:

- Garantir a integridade dos dados armazenados;
- Fornecer uma *interface* de programação de forma a permitir operações básicas sobre os arquivos;
- Fazer o controle da concorrência que surge quando múltiplos processos operam sobre um mesmo arquivo;
- Fornecer segurança (autenticação e autorização) no controle do acesso aos arquivos;
- Movimentar arquivos entre diferentes recursos de forma eficiente;
- Prover mecanismos para que os arquivos sejam localizados.

Essa lista pode ainda vir a conter diversas outras funções, dependendo do **ambiente computacional** onde o sistema de gerência de arquivos se faz necessário e às exigências feitas por esse ambiente. Por exemplo, num ambiente mono-usuário, não existe a necessidade de mecanismos que garantam a confidencialidade dos dados. Já num ambiente onde os dispositivos de armazenamento não são confiáveis, surge a função de lidar com a redundância dos dados de forma a garantir a disponibilidade dos mesmos. De fato, um ambiente computacional é caracterizado por:

1. *Plataforma de execução*: mono/multi-usuário, multiprogramação, multiprocessamento, etc;
2. *Recursos*: quanto à confiabilidade, heterogeneidade, disponibilidade, quantidade, presença de políticas de uso, distribuição e compartilhamento;
3. *Usuários*: quanto ao anonimato, quantidade (escala) e confiabilidade;
4. *Dados*: quanto à volatilidade, variabilidade (mutáveis/imutáveis), confidencialidade, integridade e durabilidade;

Associado aos arquivos, existe um conjunto de atributos, também chamados de **metadados**, que dão informações a respeito dos dados neles contidos. A finalidade principal desses metadados é auxiliar os usuários na identificação dos arquivos que contém os dados procurados. Esses atributos podem ainda ter outros objetivos, tais como dizer o formato em que os dados estão organizados no arquivo, ou ainda dizer que usuários tem permissão de acessá-lo. O próprio nome de arquivo é um metadado que dá uma idéia do seu conteúdo.

Um **Sistema de Arquivos** (SA) é um modelo de gerência de arquivos e metadados amplamente aceito pela comunidade. Seu conjunto de operações é simples e os usuários se acostumaram com a estrutura de nomes baseada no conceito de diretórios. Entretanto, o modelo de SAs precisou se adaptar aos diferentes ambientes que surgiram ao longo de muitos anos. De fato, Satyanarayanan (SATYANARAYANAN, 1989) propôs uma classificação evolutiva dos sistemas de arquivos quanto ao ambiente computacional e suas exigências:

1. Sistemas mono-usuários, mono-programados, mono-processados, não distribuídos: os sistemas de arquivos nesse ambiente tem como função fazer o mapeamento dos arquivos para os dispositivos de armazenamento, garantir a integridade dos arquivos na ocorrência de falhas ou ao se desligar o sistema, fornecer uma *interface* de programação para as aplicações (um conjunto de operações com uma semântica conhecida) e definir uma estrutura de nomes de forma que os usuários possam encontrar seus arquivos intuitivamente. Um exemplo desse ambiente é o computador IBM PC com o sistema operacional DOS;

2. Sistemas mono-usuários, multi-programados, não distribuídos: nesse ambiente, surge a necessidade de considerar o controle de concorrência no projeto da *interface* de programação e na implementação do sistema de arquivos. Um exemplo desse ambiente é o fornecido pelo sistema operacional OS/2;

3. Sistemas multi-usuários, multi-programados, não distribuídos, também conhecidos como sistemas de tempo compartilhado: a segurança dos dados passa a ser um aspecto importante no desenvolvimento dos sistemas de arquivos. Mecanismos de segurança incluem autenticação e autorização de usuários, controle de cotas, políticas de uso dos recursos, entre outros. Os sistemas UNIX representam bem esse ambiente.

4. Sistemas multi-usuários, distribuídos: sistemas de arquivos distribuídos constituem o maior nível da taxonomia, podendo serem vistos como uma implementação distribuída dos sistemas de tempo compartilhado. O grande objetivo é fornecer a mesma abstração dos sistemas de arquivos das categorias anteriores de forma eficiente, segura e robusta. Aspectos como a localização dos arquivos e a disponibilidade dos dados passam a ser importantes nesses sistemas. Cocanet, Sun NFS e CODA são alguns exemplos.

Com o passar dos anos, os sistemas distribuídos aumentaram consideravelmente em escala, ultrapassando limites administrativos e geográficos. O ambiente do SA distribuído passou a incluir redes de alta latência e alta vazão, os recursos de armazenamento nele presentes tornaram-se heterogêneos, menos acoplados e susceptíveis a um número consideravelmente maior de falhas. No entanto, os usuários continuaram a ter necessidade

de compartilhar seus dados e de torná-los acessíveis de forma globalizada. Surgiram também novos tipos de aplicações, onde o tipo de acesso aos dados é bastante diferenciado daquele típico de aplicações em um ambiente de LAN.

Esse novo ambiente, chamado de **largamente distribuído**, apresenta um número muito maior de preocupações e exigências quanto à gerência de arquivos e seus metadados. De fato, fornecer uma abstração de SA como em outros níveis da taxonomia se tornou muito difícil, ou mesmo impraticável, devido à complexidade desses sistemas. Em muitos casos, a semântica de SA é até mesmo imprópria para o tipo de aplicação envolvida.

Começou-se então a projetar sistemas para a gerência de arquivos com objetivos específicos dentro de cada ambiente. Para diminuir a complexidade, a gerência dos metadados foi desacoplada. Apenas um conjunto mínimo de atributos básicos (tais como o nome do arquivo) passou a ser freqüentemente considerado pelo sistema de gerência de arquivos. Pesquisas mais avançadas, efetuadas sobre os metadados, passaram a ser realizadas em sistemas projetados independentemente. As Grades Computacionais e os sistemas P2P são exemplos típicos de ambientes onde acontece a especialização da gerência de arquivos.

2.2 Gerência de Arquivos em Grades

O termo *Computação em Grade* (tradução para o Português do termo original em Inglês *Grid Computing*), ou simplesmente *Grade*, surgiu em meados da década de 1990 como uma metáfora em que recursos de computação seriam acessados de forma universal e transparente, tal como acontece com o uso de recursos da rede elétrica. Sua definição inicial, proposta por Ian Foster e Carl Kesselman no livro "*The Grid: Blueprint for a New Computing Infrastructure*" (FOSTER; KESSELMAN, 1999) era abrangente o suficiente para englobar a grande diversidade de sistemas largamente distribuídos, porém não era precisa o suficiente para deixar claro quais desses sistemas caracterizavam verdadeiramente uma Grade. Posteriormente, os autores (em co-autoria com Steve Tuecke) propuseram sucessivos refinamentos da definição (FOSTER; KESSELMAN; TUECKE, 2001; FOSTER, 2002; FOSTER; TUECKE, 2005) com o objetivo de desfazer a confusão. Segundo a definição mais recente e amplamente aceita, Computação em Grade é:

> "Um sistema que usa protocolos abertos e de propósitos gerais para coordenar o uso de recursos distribuídos e para fornecer qualidades de serviço acima do melhor esforço." (FOSTER; TUECKE, 2005) (tradução do autor)

A tecnologia surgiu com o objetivo de atender à demanda por computação exigida por diversas aplicações científicas, categorizadas em cinco classes principais (FOSTER; KESSELMAN, 1999):

- Supercomputação distribuída: aplicações para solucionar problemas muito grandes, necessitando de muita CPU e memória;
- Alta vazão: aplicações que capturam recursos ociosos com o objetivo aumentar a vazão agregada;
- Sob demanda: aplicações que integram recursos remotos com a computação local por um intervalo de tempo limitado;
- Intensivas em dados: aplicações que buscam a síntese de novas informações através da mineração de grandes quantidades de dados;

- Colaborativas: aplicações para dar suporte à comunicação ao trabalho colaborativo entre múltiplos participantes.

Inicialmente, a infra-estrutura de Grade fornecia a essas aplicações apenas um conjunto mínimo de ferramentas para a transferência dos dados de entrada a elas necessários. A localização desses dados ficava a cargo dos usuários que, *a priori*, sabiam onde obtê-los e para onde enviar seus resultados. Seu gerenciamento era também realizado manualmente ou de forma centralizada. No entanto, com o surgimento de novas aplicações intensivas em dados, o cenário tornou-se complexo demais para que uma abordagem puramente manual fosse utilizada ou simplesmente impraticável devido à queda de desempenho provocada pela necessidade de movimentar grandes quantidades de dados.

Com o objetivo de atender a demanda dessa classe de aplicações, propuseram-se Grades específicas, chamadas de **Grades de Dados** (CHERVENAK et al., 2000; HOSCHEK et al., 2000), que lidam com o fornecimento de serviços e de infra-estrutura para aplicações que precisam acessar, transferir e modificar massivas quantidades de dados armazenados em recursos de armazenamento distribuídos. Segundo Venugopal e outros (VENUGOPAL; BUYYA; RAMAMOHANARAO, 2006), para tirar melhor proveito dessa infra-estrutura é necessário fornecer ferramentas de gerência de dados que: (a) permitam aos usuários encontrar os arquivos com os dados procurados e descobrir recursos de armazenamento adequados para seu acesso; (b) possibilitem a transferência eficiente de grandes quantidades de arquivos; (c) facilitem o gerenciamento de múltiplas cópias dos dados; (d) permitam a identificação de recursos apropriados para o processamento dos dados; e (e) possibilitem a determinação de permissões de acesso aos arquivos.

No mínimo, uma Grade de Dados deve prover duas funcionalidades básicas: um mecanismo de transferência de dados confiável e de alto desempenho (ex.: Globus RFT (ALLCOCK; FOSTER; MADDURI, 2004)), e um mecanismo de gerenciamento e descoberta de réplicas escalável (ex.: Globus RLS e *EU DataGrid WP2 RLS* (CAMERON et al., 2004)). Dependendo das necessidades da aplicação, serviços de gerenciamento de consistência de réplicas, gerência de metadados, entre outros, precisam também ser fonecidos. Exige-se, porém, que todos esses serviços sejam colocados junto a mecanismos de segurança que assegurem a autenticidade dos usuários e seu controle de acesso através mecanismos de autorização.

2.2.1 Caracterização do Ambiente

No ambiente da Grade de Dados, os dados estão tipicamente organizados em coleções de arquivos que são armazenados em sistemas de armazenamento em massa (também chamados de repositórios). Os usuários os acessam de diferentes locais geograficamente distribuídos e podem criar cópias locais, chamadas de **réplicas**, com o objetivo de diminuir as latências envolvidas com a transferência de dados em redes de grandes distâncias, melhorando assim o desempenho da aplicação que processará esses dados. Quando possível, também utiliza-se a abordagem de migrar a aplicação para os locais onde se encontram cópias dos arquivos. Um Sistema de Gerência de Réplicas (*Replica Management System* – RMS) permite aos usuários criar, registrar, gerenciar réplicas e também atualizá-las caso os arquivos originais sejam modificados. Dependendo da aplicação, são feitas diferentes exigências quanto à consistência, segurança no acesso, controle de custo de armazenamento, persistência e disponibilidade das réplicas gerenciadas. O RMS pode ainda criar réplicas automaticamente segundo alguma estratégia de replicação que leva em consideração a demanda atual e futura pelos arquivos, a localidade das requisições e a capacidade de armazenamento dos repositórios.

Outra característica freqüentemente presente nas Grades de Dados é a manutenção dessas coleções de dados distribuídas entre diversos domínios administrativos. A durabilidade dos dados deve ser assegurada independentemente dos sistemas de armazenamento utilizados e deve-se poder agregar com facilidade novos recursos. É necessário que as informações associadas aos dados (tais como os metadados, controle de acesso e mudanças de versão) sejam também preservados mesmo na ocorrência de falhas ou mudanças na plataforma. Sistemas que atendem a essas exigências são conhecidos como **sistemas de arquivamento** (MOORE; RAJASEKAR; WAN, 2005).

Um cenário de Grade de Dados típico consiste de recursos computacionais e de armazenamento localizados em diferentes países e interconectados por redes de alta velocidade. Os grandes centros são interligados por redes de alta vazão, enquanto que os centros subsidiários são interligados por redes de menor capacidade. Os dados gerados por um instrumento, experimento, ou rede de sensores são armazenados nos centros computacionais onde se encontram e são transferidos para outros centros espalhados geograficamente de acordo com a estratégia de replicação adotada. Os usuários consultam então catálogos de réplicas e metadados para localizar os dados que precisam e, caso seja a eles concedida a devida permissão, recupera-se os arquivos a partir dos locais mais próximos a esses usuários. Caso não existam cópias em um local próximo, os dados são buscados nos repositórios.

Os recursos no ambiente de Grade são extremamente heterogêneos no que diz respeito ao *hardware* (CPU, memória e capacidade de armazenamento), ao ambiente operacional (infra-estrutura de rede ao qual estão situados), à plataforma de execução (sistema operacional, *software* de Grade e linguagem de programação utilizada pelas aplicações) e à disponibilidade (probabilidade de o recurso estar pronto para ser utilizado). Eles estão também sobre o controle de diferentes domínios administrativos, cujas entidades responsáveis são autônomas e possuem políticas próprias para permitir o acesso por parte dos usuários. Portanto, Grades lidam com o compartilhamento e gerenciamento de recursos, com a autenticação e autorização de usuários para o seu uso, e com o seu escalonamento eficiente.

As Grades de Dados compartilham essas preocupações quanto ao uso dos recursos, porém precisam também atender a aspectos relativos aos dados. Os arquivos nesse ambiente possuem grande heterogeneidade no que diz respeito ao seu tamanho (variando de algumas dezenas de *Megabytes* até dezenas de *Gigabytes*) e popularidade (com dados recentes muito requisitados enquanto outros já analisados são pouco utilizados). No geral, os dados são imutáveis ou existe uma única entidade que os altera com pouca freqüência (normalmente apenas nos primeiros dias após terem sido gerados), sendo necessário apenas propagar as alterações para as réplicas existentes. Assim como os recursos, cada domínio administrativo pode impor suas políticas próprias de acesso aos dados, autorizando diferentes usuários. Ou seja, alguns arquivos tem caráter confidencial, outros podem ser modificados somente por um pequeno grupo de usuários, enquanto outros são abertamente disponíveis para leitura (VENUGOPAL; BUYYA; RAMAMOHANARAO, 2006; BUNN; NEWMAN, 2003).

2.3 Gerência de Arquivos em Ambientes Par-a-Par

Embora seja possível encontrar referências que remetam ao modelo par-a-par (do original em Inglês *peer-to-peer* – P2P) desde o final da década de 1960 ((EBERSPäCHER; SCHOLLMEIER, 2005; CROWCROFT et al., 2004; MINAR; HEDLUND, 2001; MI-

LOJICIC et al., 2002)), a popularização da Internet e o surgimento de aplicações de compartilhamento de arquivos, como o Napster (NAPSTER FREE, 2008) e o Gnutella (GNUTELLA, 2008), contribuíram consideravelmente para a sua disseminação, tanto no meio comercial quanto acadêmico.

Assim como no caso da Computação em Grade, essa popularização acabou gerando uma certa confusão sobre o que realmente é o modelo P2P. Diversos autores ((MILOJICIC et al., 2002; SHIRKY, 2001; ANDROUTSELLIS-THEOTOKIS; SPINELLIS, 2004; STEINMETZ; WEHRLE, 2005)) propuseram definições com o objetivo de identificar uma classe de sistemas distribuídos com características comuns e, paralelamente, auxiliar a delimitar a área de pesquisa. Entre as quatro definições, a de Theotokis e Spinellis se mostra a mais completa e precisa:

> "Sistemas peer-to-peer são sistemas distribuídos compostos por nós interconectados entre si, capazes de se auto-organizarem em topologias de rede com o propósito de compartilharem recursos tais como conteúdo, ciclos de processador, armazenamento e largura de banda, de se adaptarem a falhas e de aceitarem populações variáveis de nós enquanto mantém conectividade e desempenho satisfatórios, sem necessitarem intermediação ou suporte de um servidor ou autoridade global centralizada." (ANDROUTSELLIS-THEOTOKIS; SPINELLIS, 2004) (tradução do autor)

Em resumo, sistemas considerados P2P são aqueles que, quando observados externamente, dão a impressão de estarem fornecendo interação direta entre os nodos. A idéia é que os nodos assumam as funcionalidades tanto de cliente quanto de servidor, numa alternativa ao tradicional modelo cliente-servidor amplamente empregado em sistemas distribuídos.

Entre as vantagens do modelo P2P, se destacam: (a) maior escalabilidade e flexibilidade, através da incorporação sob demanda de recursos; (b) ganho de desempenho ao agregar recursos potencialmente ociosos; (c) distribuição do custo de propriedade dos recursos entre os participantes; e (d) aumento inerente da confiabilidade do sistema, ao reduzir a dependência em pontos centrais.

Por outro lado, o modelo introduz uma série de aspectos que precisam ser atacados para comprovação dos benefícios que ele potencialmente agrega. A complexidade de projetar e de implantar um sistema totalmente distribuído, seguindo o modelo P2P, é muito maior do que no caso do modelo cliente-servidor. Dentre os principais, está a necessidade de auto-organizar os nodos através de protocolos escaláveis e seguros para roteamento de mensagens, garantir o correto funcionamento dos serviços mesmo na ocorrência de um grande número de falhas e fazer o balanceamento de carga para evitar pontos na rede onde existe uma demanda por serviços desproporcionalmente maior.

Embora os sistemas P2P tenham se popularizado pelas aplicações de distribuição de conteúdo (ex.: Napster (NAPSTER FREE, 2008), Gnutella (GNUTELLA, 2008), Kazaa (CHOON-HOONG; NUTANONG; BUYYA, 2005), OceanStore (RHEA et al., 2003)), seu modelo tem sido utilizado também por outras classes de aplicações. Segundo Theotokis e Spinellis (ANDROUTSELLIS-THEOTOKIS; SPINELLIS, 2004), a arquitetura P2P tem sido empregada também em aplicações de computação distribuída (ex.: SETI@Home (ANDERSON et al., 2002), XtremeWeb (GERMAIN et al., 2000), Computer Power Market (BUYYA; VAZHKUDAI, 2001)), aplicações colaborativas (ex.: mensagem instantânea entre usuários – ICQ, MSN, Jabber (JABBER PROTOCOLS, 2008)), aplicações que dão suporte a serviços na Internet (ex: serviço de *multicast* (RENESSE et al., 2003)) e

sistemas de bancos de dados distribuídos (ex.: PIER (HUEBSCH et al., 2003)).

Mais diretamente relacionado a este trabalho estão as aplicações de distribuição de conteúdo. Essa classe engloba desde simples aplicações de compartilhamento de arquivos até sistemas que fornecem uma mídia de armazenamento para que, de forma segura e eficiente, os usuários possam publicar, organizar, indexar, pesquisar, atualizar e recuperar seus dados. Os motivos para focar essas aplicações é que elas lidam com estratégias eficientes para localizar arquivos, fornecem protocolos confiáveis para movimentação de dados mesmo com a grande volatilidade dos nodos, e propõem mecanismos para lidar com o grande número de requisições causadas pela demanda por arquivos populares.

2.3.1 Caracterização do Ambiente P2P

Em um ambiente P2P, os recursos (nodos) estão presentes em larga escala e estão distribuídos globalmente. Esses recursos entram e saem da rede com grande freqüência (alta volatilidade), e estão também sujeitos a um maior número de falhas, refletindo, portanto, pouca disponibilidade. Basicamente, eles são *desktops* de usuários posicionados em pontos da Internet com baixa conectividade e/ou lentos (com taxas de transmissão na ordem de Kbps a poucos Mbps). Apesar da heterogeneidade, normalmente não há grandes discrepâncias quanto a sua capacidade (CPU, disco, memória).

Os usuários são, em geral, voluntários e cedem seus recursos e dados em troca de algum benefício, tais como prioridade nas transferências de arquivos remotos ou um aumento na cota de armazenamento a ele atribuída. O número de usuários normalmente atinge a escala de milhões e, assim como os recursos, estão fisicamente espalhados de forma global. Não existem usuários com autoridades especiais e os arquivos são normalmente disponibilizados de forma anônima, através de um sistema capaz de protegê-los contra a censura.

Diferentemente das Grades de Dados, onde geralmente os dados são gerados por poucas fontes ou por uma rede de sensores, no ambiente P2P existe um potencial grande número de entidades capazes de gerar dados. Esses dados são, na sua maioria, arquivos pessoais de usuários (documentos, filmes, músicas, etc) que desejam compartilhar com os demais. O tamanho desses arquivos é muito variável, ficando, em sua maioria, na faixa de alguns poucos *MBytes*. Esses dados são geralmente disponibilizados voluntariamente e de forma anônima. Sua característica de mutabilidade depende da aplicação envolvida: em aplicações simples como as de compartilhamento de arquivos, considera-se que os dados são somente leitura e não é dado garantias quanto à consistência de múltiplas cópias; já em aplicações mais sofisticadas como as de armazenamento persistente, permite-se que os dados sejam atualizados através de mecanismos de controle de versões. Os dados podem ainda ser replicados tanto por motivos de desempenho (como nos casos onde existe a necessidade de atender a uma grande demanda por arquivos populares) quanto tolerância a falhas (como nos sistemas que fornecem armazenamento persistente).

Uma propriedade característica de sistemas lançados sobre esse ambiente é a simetria de funções desempenhadas pelos nodos. Ou seja, normalmente não existem nodos com capacidades especiais e que desempenham funções em particular. Sistemas cliente-servidor convencionais são assimétricos pois os servidores geralmente são muito mais capacitados do que os clientes, possuindo funções diferenciadas. Como viabilizam o compartilhamento de recursos e dados sem a necessidade da intermediação de um servidor central, os sistemas P2P, diferentemente de outros sistemas distribuídos tradicionais que normalmente escalam para não mais que poucas centenas ou milhares de nodos, escalam para milhões de nodos e usuários. A administração, a manutenção e a responsabili-

dade de operação do sistema também são distribuídas entre os usuários ao invés de serem controladas por uma única companhia, instituição ou pessoa.

A escala que o sistema assume torna difícil, senão impossível, fazer a auditoria de usuários. Sistemas distribuídos convencionais, por estarem mais próximos de seus usuários, simplificam os protocolos do sistema ao considerar que estes são confiáveis, atribuindo às entidades responsáveis pelo sistema a responsabilidade de controlar as atitudes dos usuários de forma a penalizá-los caso não estejam agindo de acordo com as regras. Um dos grandes desafios no projeto de sistemas P2P é, portanto, fornecer suas funcionalidades através de protocolos que sejam seguros, sem comprometer seu desempenho e escala.

Esses sistemas devem ainda ser capazes de fazer uma boa distribuição dos objetos de dados baseando-se na capacidade e na disponibilidade dos recursos. Ao executar essa tarefa, são necessários mecanismos para evitar que a carga seja desproporcionalmente alta em locais específicos da rede. Uma vez mapeados para os recursos, é desejável que seja feito continuamente o rebalanceamento da carga, tipicamente através da observação de padrões de uso (por exemplo, a popularidade dos arquivos).

Por serem fornecidos voluntariamente pelos usuários, não se pode esperar ou forçar a presença de recursos. Conseqüentemente, exige-se que o sistema seja robusto a ponto de tolerar o grande número de falhas aos quais esses recursos estão susceptíveis e de suportar a remoção desses elementos a qualquer momento. A rápida entrada/saída de nodos ainda pode causar oscilações na rede, resultando em queda de desempenho. Como existe um custo de manutenção e gerenciamento de estruturas e/ou objetos de dados associado à entrada/saída dos nodos, esse efeito pode provocar também problemas de segurança. Nesse caso, as oscilações podem ser intencionalmente provocadas por um invasor a fim de fazer ataques de negação de serviço. O sistema deve então ser projetado para resistir a esse tipo de ataque, tolerando-as sem uma queda significativa no desempenho de suas funcionalidades.

Uma vez que os usuários do sistema não são confiáveis, aplicações que desejam fornecer acesso persistente aos dados precisam fornecer mecanismos que garantam o armazenamento seguro dos dados, sua proteção contra a destruição e sua disponibilização de forma transparente. A criptografia, os esquemas de codificação e a redundância na realização de operações são exemplos de técnicas usadas para atingir esse objetivo. Normalmente se assume, no entanto, que a maioria dos usuários é confiável e segue as regras do sistema.

O mais importante aspecto relacionado ao modelo P2P diz respeito ao método utilizado para localizar recursos ou dados. Uma vez que os recursos de armazenamento estão distribuídos entre uma grande quantidade de nodos, um algoritmo eficiente para localizar dados nesses recursos (ou descobrir os recursos onde os dados devem ser colocados) torna-se um fator decisivo no projeto desses sistemas.

2.3.2 Rede Lógica P2P

É através da uma rede lógica de conexões que os nodos em um sistema P2P são organizados para permitir a localização dos dados. Essa rede lógica, também chamada de rede de sobreposição ou topologia (em inglês, *overlay network*), é independente da rede física que interliga as máquinas. As camadas de sobreposição são classificadas em duas categorias principais: estruturadas ou não-estruturadas (ANDROUTSELLIS-THEOTOKIS; SPINELLIS, 2004; CROWCROFT et al., 2004; BALAKRISHNAN et al., 2003). Por estruturada entende-se que existe uma topologia bem definida onde os objetos

de dados podem ser localizados deterministicamente, diferentemente das redes P2P não-estruturadas, onde a atribuição dos dados aos nodos não depende da rede lógica formada (ANDROUTSELLIS-THEOTOKIS; SPINELLIS, 2004).

Redes P2P Não-Estruturadas

As regras usadas para formação de uma rede não-estruturada são menos rígidas e resultam em uma topologia aleatória, na maioria das vezes sem uma estrutura denida. Os mecanismos de pesquisa por conteúdo nessas redes incluem *fooding*, *random walks* e outros métodos por força bruta. Essas redes suportam bem populações muito voláteis e são simples de serem construídas e mantidas, mas são inecientes para pesquisa quando o tamanho da rede cresce muito (BALAKRISHNAN et al., 2003). O principal exemplo é a rede utilizada pelo sistema Gnutella (GNUTELLA, 2008).

Redes P2P Estruturadas

No tipo de rede lógica estruturada, nodos e objetos de dados compartilham um mesmo espaço de identificadores de m bits de tamanho. Um identificador de um nó é chamado de `NodeID` e um identificador de um objeto de dados é chamado de `chave`. Ambos são gerados através de uma função *hash* (tipicamente SHA1) aplicada a um conteúdo que identifique unicamente o nó ou o objeto de dados. A associação de uma `chave` a um `NodeID` se dá através de uma função distância: diz-se que um nó é **responsável** por responder pelas `chaves` que forem mais próximas a ele do que a qualquer outro nó do sistema, de acordo com a noção de proximidade dada por essa função. Essa mesma função é usada entre `NodeIDs` para determinar a proximidade entre dois nodos e, conseqüentemente, determina quem são seus vizinhos nessa rede lógica.

Cada nó mantém informações sobre alguns dos outros nodos da rede lógica em uma estrutura de dados local, chamada de tabela de roteamento. Um algoritmo P2P usa então essas informações para rotear mensagens referentes a uma `chave` até o nó por ela responsável. Esse processo de encontrar o nó responsável por uma `chave` é chamado de *lookup*, e constitui a principal funcionalidade oferecida por um *middleware* P2P. Seu desempenho é medido em função do número de nodos que são consultados até que o nó responsável seja encontrado. Algoritmos P2P mais conhecidos, tais como Chord (STOICA et al., 2001), Pastry (ROWSTRON; DRUSCHEL, 2001b) e Kademlia (MAYMOUNKOV; MAZIèRES, 2002) concluem um *lookup* em $\mathcal{O}(logN)$ consultas, onde N é o número de nodos que compõem a rede P2P.

Um aspecto muito importante na escolha do algoritmo P2P diz respeito à distribuição do espaço de identificadores entre nodos, que depende da função distância usada em cada algoritmo. Em um algoritmo P2P ideal, os nodos são responsáveis por um número igual de chaves, balanceando a carga das consultas e do armazenamento dos dados referentes às chaves. Por exemplo, quando a `chave` identifica o conteúdo de um arquivo, um nó responsável por muitas chaves vai responder por um grande número de requisições e vai precisar de bastante espaço de armazenamento para gravar os arquivos. Esse problema pode se agravar quando os dados são potencialmente grandes, como no caso da Grade de Dados do HEP, onde o tamanho dos arquivos pode atingir a ordem de *Gigabytes* com facilidade.

2.4 Por que usar a tecnologia P2P em um ambiente de Grade é vantajoso para a gerência de arquivos?

Em um ambiente como o da Grade de Dados, a necessidade das aplicações de acessar grandes quantidades de dados impõe sérias restrições de desempenho. Com o objetivo de diminuir a latência no acesso a esses dados, utiliza-se extensivamente técnicas de replicação de arquivos. A idéia é utilizar os recursos de armazenamento disponíveis nesse ambiente para melhorar o desempenho das aplicações. Entretanto, devido à incapacidade dos serviços de Grade de gerenciar recursos de armazenamento mais voláteis, tais como aqueles encontrados em pequenos centros ou laboratórios onde os cientistas desejam processar os dados, apenas os recursos fortemente controlados são utilizados. Por fortemente controlados entende-se que o recurso é monitorado por pessoas que constantemente fazem sua manutenção, evitando que se torne indisponível com muita freqüência ou por longos períodos, além de garantir seu uso correto evitando desperdícios e penalizando usuários que não estejam agindo de acordo com as regras de uso. Para viabilizar essa solução, o *hardware* de armazenamento ainda precisa ser menos susceptível a falhas que componentes de armazenamento de prateleira. Além do alto custo de gerenciamento em termos de recursos humanos necessários para o seu controle, a exigência por um *hardware* de qualidade eleva o custo de sua aquisição. Esse conjunto de fatores dificulta a contribuição de recursos de armazenamento por parte das entidades, culminando na centralização dos dados em locais onde há verbas para aquisição de sistemas de armazenamento em massa (ex.: *Castor* no Cern, *Enstore* no Fermilab, HPSS no Laboratório de Berkeley, TSM no laboratório DESY, entre outros).

Uma outra abordagem utilizada nesse ambiente é a de migrar as aplicações para os locais da Grade onde se encontram os dados ou para locais próximos a eles. A vantagem dessa estratégia é não exigir a movimentação de grandes quantidades de dados. No entanto, como as aplicações deixam de rodar em recursos computacionais locais, exige-se que eles sejam processados nos lugares onde encontram-se os dados. Devido ao volume excessivo de dados, acabam-se por formar grandes centros onde, além da necessidade por recursos de armazenamento potencialmente caros, torna-se necessário também a presença de grandes quantidades de recursos para processá-los, elevando ainda mais o custo. Outro aspecto importante diz respeito à utilização da técnica de migração em si, que encontra grandes dificuldades de ser aplicada em ambientes heterogêneos como o da Grade. Para facilitar seu uso, as entidades tentam acordar quanto às configurações do *hardware* a ser utilizado, fato que impede que muitas instituições contribuam com seus recursos de processamento já existentes. Por serem configurações de *hardware* de alto desempenho, a abordagem impõe ainda restrições orçamentárias para aquisição de novos desses recursos.

O fato de ambas as técnicas acabarem por centralizar o armazenamento e o processamento em grandes centros computacionais advém da incapacidade da plataforma de Grade de escalar seus serviços de gerenciamento a ponto de capturar recursos menos acoplados. Para lidar com esse problema, seria necessário que os recursos fossem autogerenciáveis e o sistema tolerasse o grande número de falhas ao quais esses recursos estão susceptíveis. É justamente nesses aspectos que o modelo P2P tem muito a oferecer. Assim, justifica-se o uso dessa técnica com o objetivo de possibilitar que recursos menos capacitados sejam agregados com facililidade, possibilitando seu uso no ambiente de Grade e, dessa forma, diminuindo custos. O fato de potencializar a quantidade de recursos disponíveis e diminuir o custo do armazenamento permite também criar um maior número de réplicas para os dados, tornando-os mais próximos das aplicações. Como essas cópias

passam a ser gerenciadas em recursos locais aos usuários, seu processamento pode ser realizado também localmente, evitando assim a dependência por recursos computacionais disponibilizados nos grandes centros.

Enquanto que o modelo P2P lida com os aspectos de auto-gerenciamento, tolerância a falhas e escalabilidade, utiliza-se a infra-estrutura padronizada da Grade para obter segurança permitir a interoperabilidade entre os serviços. Como os aspectos tratados por uma são complementares aos da outra, ainda que ambas tenham objetivos semelhantes (fornecer mecanismos para compartilhamento de recursos em sociedades computacionais de larga escala), a convergência dessas tecnologias já vem sendo prevista pela academia (FOSTER; IAMNITCHI, 2003; ANDROUTSELLIS-THEOTOKIS; SPINELLIS, 2004; HASAN et al., 2005).

O desafio é, no entanto, fornecer uma arquitetura P2P para a gerência de dados que faça um bom uso dos recursos de armazenamento em um ambiente onde pretende-se agregar tanto recursos de pouca capacidade e alta volatilidade (ex.: *desktops*, estações de trabalho e pequenos *clusters*) quanto aqueles de grande capacidade e alta disponibilidade fornecidos pelos grandes centros (ex.: sistemas de armazenamento em massa e grandes *clusters*). Essa arquitetura ainda deve dar suporte eficiente a aplicações que precisem processar grandes quantidades de dados.

3 TRABALHOS RELACIONADOS

Este capítulo apresenta uma síntese dos principais trabalhos relacionados. O critério de seleção utilizado baseou-se em três aspectos fundamentais: (a) deve gerenciar arquivos em um ambiente de larga escala; (b) deve seguir o modelo P2P; e (c) deve gerenciar os recursos de armazenamento. A ordem em que eles aparecem no texto a seguir respeita uma classificação, em termos de requisitos atendidos e ambiente-alvo, por similaridade com este trabalho.

3.1 OppStore

OppStore (CAMARGO; KON, 2007) é um *middleware* que fornece armazenamento de dados somente leitura para Grades de forma confiável. O objetivo é usar a grande quantidade de espaço em disco disponível em Grades formadas por estações de trabalho (chamadas de *Opportunistic Grids* (LITZKOW; LIVNY; MUTKA, 1988)) para fornecer uma solução de baixo custo para o armazenamento de dados. As máquinas nesse tipo de Grade, além de apresentarem grande heterogeneidade, são freqüentemente desligadas e/ou reiniciadas e podem ser usadas apenas quando estiverem ociosas, caracterizando um ambiente bastante dinâmico.

Estrutura Organizacional

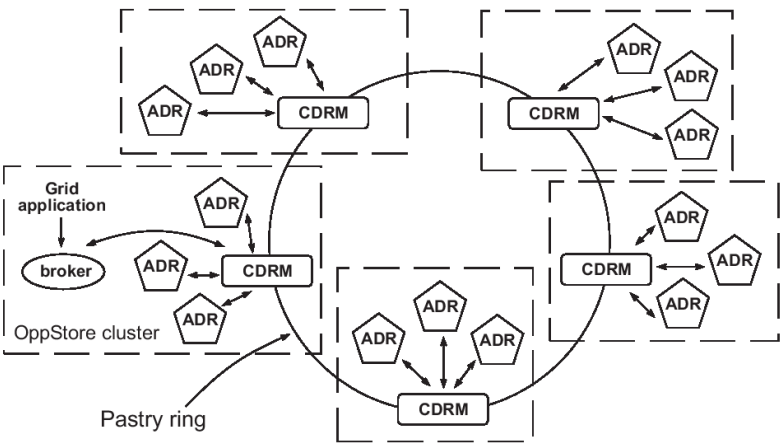

Figura 3.1: A arquitetura do OppStore (CAMARGO; KON, 2007).

O sistema utiliza o algoritmo Pastry (ROWSTRON; DRUSCHEL, 2001b) para criar uma rede P2P estruturada, objetivando a obtenção de escala e auto-organização. Para

lidar com a volatilidade e a instabilidade dos recursos de armazenamento, OppStore os organiza em uma federação de *clusters*. Cada um deles é composto por máquinas de um mesmo laboratório ou departamento dentro de uma instituição. Apenas um de seus nodos, chamado *Cluster Data Repository Manager* (CDRM), ingressa a rede P2P (Fig. 3.1). A cada CDRM é atribuído um identificador Pastry, que caracteriza um trecho do espaço de chaves da rede P2P que deverá ser gerenciado (armazenado) pelo *cluster*. As demais máquinas do cluster são chamadas de *Autonomous Data Repository* (ADR). Para decidir em qual ADR do *cluster* um objeto de dados será armazenado, o CDRM se baseia na **capacidade** de cada nó. Nodos do *cluster* com maior capacidade responsabilizam-se pelo armazenamento de um número maior de objeto de dados.

Tratamento da Heterogeneidade de Recursos: *virtual ids*

Embora os autores afirmem que a capacidade dos nodos possa ser dada em função de diversos fatores, tais como a disponibilidade, espaço em disco fornecido, banda de rede e poder de processamento, apenas a disponibilidade média da máquina é usada. Já a capacidade de um *cluster* é dada pela soma das capacidades das máquinas (ADRs) que o compõem. A idéia é atribuir um tamanho de trecho de identificadores a serem por ele gerenciados que seja proporcional a sua capacidade. Dessa forma, pretende-se corrigir o desbalanço provocado pela heterogeneidade dos recursos (ex.: nodos pouco disponíveis gerenciando a mesma quantidade de dados que nodos que estão quase sempre conectados). Ainda que todos os *clusters* tivessem a mesma capacidade, a forma aleatória de atribuir identificadores no Pastry também provoca diferenças no tamanho dos trechos gerenciados por cada CDRM. Esse desbalanço também é corrigido ao determinar o tamanho dos trechos proporcional à capacidade. No entanto, as sucessivas entradas/saídas de nodos refletem uma variabilidade na capacidade de cada *cluster*, assim como a entrada/saída de CDRMs na rede P2P exige que os trechos sejam redistribuídos de forma a manter o tamanho dos mesmos proporcional às capacidades. Para tratar dessa dinamicidade, OppStore permite que o tamanho dos trechos seja variável através do uso de **identificadores virtuais** (CAMARGO; KON, 2006) (Fig. 3.2).

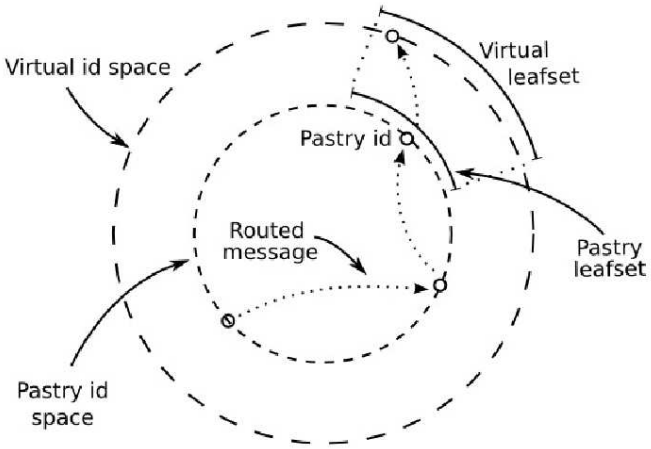

Figura 3.2: A técnica de identificadores virtuais (CAMARGO; KON, 2006).

Nessa técnica, cada nó da rede P2P recebe um identificador adicional, chamado de *virtual Id*, para o qual o identificador do nó no Pastry é mapeado. O espaço virtual tem o mesmo tamanho do espaço do Pastry, e a redistribuição desse espaço virtual se dá apenas entre nodos vizinhos na rede P2P. Para rotear as mensagens ao nó virtual responsável por

um dado procurado, apenas o último passo do esquema de roteamento do Pastry é modificado, fazendo com que a tabela de nós virtuais vizinhos (chamada *virtual leafset*) seja usada ao invés das informações de vizinhança fornecidas pelo Pastry. Conseqüentemente, mantém-se o mesmo número de passos de roteamento do algoritmo do Pastry sem o uso dessa técnica. Os autores ainda afirmam que o sobrecusto de manutenção do espaço virtual (redistribuição e gerenciamento dos *virtual Ids*) é pequeno. Por fim, para evitar que o espaço em disco dos nodos com grande disponibilidade fique lotado rapidamente, o sistema diminui linearmente a capacidade associada a cada nó sempre que o espaço ocupado em disco ultrapassar um certo limiar ($1GB$).

Redundância de Dados

Com o objetivo de tolerar a entrada/saída de máquinas e de melhorar a disponibilidade dos dados, OppStore os divide em fragmentos redundantes durante a etapa de armazenamento e os distribui entre diferentes *clusters*. O procedimento, que combina as técnicas de replicação e fragmentação de dados, é realizado através do uso de um **algoritmo de dispersão de informação** (*Information Dispersal Algorithm* – IDA) (RABIN, 1989). Nesse esquema, um arquivo de tamanho t é codificado em $n + m$ fragmentos de tamanho t/n, dos quais apenas n fragmentos quaisquer são necessários para decodificar e obter o arquivo original. O arquivo poderá então ser reconstituído mesmo caso m dos fragmentos estejam indisponíveis devido a falhas de nodos. Para um mesmo nível de redundância (fator de replicação), estudos de terceiros (WEATHERSPOON; KUBIATOWICZ, 2002; RODRIGUES; LISKOV, 2005) mostram que o uso de um IDA provê uma disponibilidade média dos dados maior do que quando a simples replicação é utilizada.

Operações de Arquivamento

Escrita (inserção): ao inserir um arquivo no sistema, roteia-se a chave que identifica cada fragmento (o *hash* do seu conteúdo) para o CDRM responsável no espaço de identificadores virtual. Esse CDRM escolhe uma máquina do *cluster* e devolve o endereço da mesma para que ela possa ser contactada diretamente para fazer o armazenamento do fragmento. Após descobrir o destino de cada fragmento e concluir suas transferências, monta-se um índice, chamado *File Fragment Index* (FFI), que contém o *hash* de cada um desses fragmentos e o local onde podem ser obtidos. Por fim, o FFI é transferido para o CDRM por ele responsável. Existem dois modos de inserção de um arquivo no sistema: *perennial* e *ephemeral*. O primeiro deles é usado para armazenamento de longo prazo, enquanto que o segundo é usado para dados que requerem armazenamento por apenas algumas horas. No modo *ephemeral*, os dados são armazenados em um único *cluster* para fins de desempenho.

Leitura (recuperação): para recuperar um arquivo, localiza-se através da rede P2P o CDRM responsável por armazenar o FFI. Após obtê-lo, n fragmentos (dos m possíveis) são escolhidos e transferidos. Em seguida, verifica-se a integridade de cada fragmento comparando seu *hash* com aquele presente no FFI e decodifica-se os fragmentos para então obter o arquivo original. Como existem $n + m$ fragmentos, pode-se escolher os n fragmentos que estão em máquinas mais próximas ou em redes mais rápidas. Essas métricas são obtidas através de um histórico de transferências gravado localmente na máquina do usuário.

Remoção: como não existe uma operação de remoção de arquivos associada, o gerenciamento do espaço ocupado é dado através de aluguéis (*leases*). Ao inserir um arquivo, o usuário especifica por quanto tempo o arquivo deverá ser gerenciado (armazenado). Du-

rante esse tempo, o sistema monitora o número de fragmentos disponíveis e os substitui após um limiar de indisponibilidade, garantindo assim que o arquivo possa ser recuperado. Precisa-se então renovar o aluguel do arquivo antes que esse prazo termine para assegurar a durabilidade do mesmo. Caso esse tempo venha a expirar, o sistema não vai garantir que as operações de recuperação de arquivo sejam bem sucedidas, mesmo caso o aluguel seja posteriormente renovado.

Manutenção: para dar garantias quanto à recuperação dos dados, as informações contidas nos CDRMs (dentre as quais, os FFIs) são replicadas em nodos vizinhos da rede P2P. Embora os FFIs continuem sendo localizáveis mesmo quando um CDRM deixa o sistema, a entrada/saída de CDRMs exige a movimentação dos FFIs de uma máquina para outra. Essa necessidade, no entanto, não persiste para os fragmentos de arquivos, pois estes são localizados através de um endereço contido nos FFIs. Já na ocorrência de uma falha em um ADR, precisa-se reconstituir o arquivo por inteiro para que uma nova cópia do fragmento seja gerada. Como essa operação é extremamente custosa, ela só é realizada depois que uma quantidade significativa de fragmentos do arquivo é perdida. Até que os novos fragmentos sejam gerados, os CDRMs marcam nos FFIs aqueles que se encontram indisponíveis.

Resultados Obtidos

Os resultados por simulação (CAMARGO; KON, 2007) mostram que, em configurações realistas de Grade de máquinas não-dedicadas, o uso de identificadores virtuais melhora a disponibilidade dos dados (aumenta as chances de o arquivo poder ser reconstituído). Também é mostrado que, para fatores de replicação maiores, o aumento do número de fragmentos resulta numa maior disponibilidade. Nessas condições, o fato de os *clusters* estarem mais espalhados geograficamente também resulta em um ganho na disponibilidade dos dados, pois diminui a correlação de indisponibilidade de máquinas em diferentes *clusters*. Através de experimentos em uma Grade real, os autores mostram ainda que a possibilidade de escolha dos melhores fragmentos otimiza o tempo de recuperação do arquivo.

3.2 CFS

O *Cooperative File System* (CFS) (DABEK et al., 2001) é um sistema de armazenamento para dados imutáveis baseado em redes P2P. Seu objetivo é capturar os recursos fornecidos por participantes voluntários para fornecer um gerenciamento eficiente, robusto e capaz de balancear a carga do armazenamento e da recuperação de arquivos. Utiliza-se as técnicas de fragmentação, replicação, *caching* e nodos virtuais para atingir esses propósitos.

Estrutura Organizacional

Todas as máquinas que são gerenciadas pelo CFS fazem parte de uma rede P2P estruturada baseada no algoritmo *Chord* (STOICA et al., 2001). Sua organização é, portanto, totalemente descentralizada, não exigindo a presença de recursos mais capacitados para realizar operações específicas. Através dos serviços fornecidos pelo Chord, CFS mapeia **blocos** de dados para as máquinas. Cada um deles é gerenciado independentemente em um nó através de um componente chamado `DHash`, que é responsável por garantir a sua manutenção frente às entradas/saídas de outras máquinas. Os blocos, que podem armazenar tanto dados de arquivos como metadados (ex.: informações de diretório), são então

interpretados pelos usuários (componente FS) de forma semelhante a blocos em disco em um sistema de arquivos convencional (Fig. 3.3).

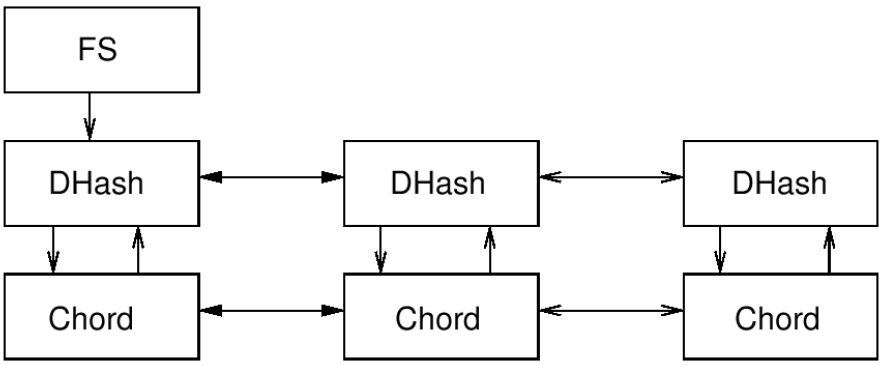

Figura 3.3: A arquitetura do CFS (DABEK et al., 2001).

Distribuição dos Dados

Cada usuário pode inserir dados apenas no seu próprio sistema de arquivos, através da atualização de um bloco especial, chamado bloco raiz. Esse bloco contém o *hash* do conteúdo dos demais blocos, sejam eles diretórios ou arquivos. Ele também é assinado digitalmente com a chave privada do usuário, de forma que toda a integridade e autenticidade do sistema de arquivos pode ser verificada (Fig. 3.4).

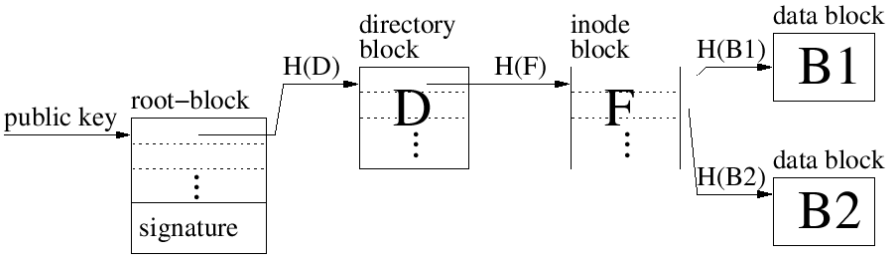

Figura 3.4: Exemplo de estrutura de sistema de arquivos do CFS (DABEK et al., 2001).

Os blocos possuem tamanho fixo e recebem um identificador único, chamado Id. Ele é utilizado pelo sistema para determinar os servidores responsáveis pelo seu gerenciamento. Com exceção do bloco raiz, cujo Id é a própria chave pública do usuário, o Id de cada bloco individual é dado pelo *hash* do conteúdo do bloco. Cada máquina que pertence ao sistema também recebe um identificador no mesmo espaço de Ids dos blocos.

Um bloco é considerado de responsabilidade de um nó se o seu Id é imediatamente sucedido pelo Id do nó no espaço de identificadores. Para saber qual é esse nó, solicita-se uma operação de localização aos serviços P2P, usando como chave o Id do bloco. Durante essa operação, consulta-se nodos cada vez mais próximos ao sucessor da chave procurada, até que ele seja descoberto. Tal aproximação é dada logaritmicamente em relação ao número de nodos presentes na rede P2P. Uma vez localizado o nó responsável, o bloco é diretamente a ele enviado ou dele requisitado. Os autores afirmam que o custo de localização do nó responsável por um bloco é pequeno comparado ao custo de transferência do bloco, embora o tamanho do mesmo seja apenas na ordem de algumas dezenas de *Kilobytes*. É possível ainda fazer *pre-fetching* dos blocos a fim de mascarar

esse custo de localização: enquanto alguns blocos vão sendo transferidos, outros estão sendo localizados em paralelo.

Cada arquivo inserido no sistema é dividido em diversos desses blocos, que são armazenados com o esquema descrito. A idéia da divisão em blocos é balancear a carga de armazenamento: arquivos com diferentes tamanhos não causam discrepâncias quanto ao espaço em disco ocupado em diferentes nodos. O espaço necessário para armazenar um arquivo grande, por exemplo, seria dividido entre um grande conjunto de nodos. Como diversos nodos estão envolvidos no seu gerenciamento, o uso de fragmentação também tem o objetivo de balancear a carga de atendimento à requisições por arquivos populares. Para tratar a popularidade de arquivos pequenos (que são quebrados em poucos blocos), CFS faz *caching* dos mesmos nos nodos anteriores aos nodos responsáveis pelos blocos no espaço de identificadores do Chord. Os nodos que precedem o nó responsável pelo bloco tendem a ser consultados conforme as mensagens vão convergindo até ele na rede P2P.

Redundância de Dados

Como os blocos na *cache* estão sujeitos a serem substituídos a qualquer momento, essa técnica não pode ser também utilizada para garantir tolerância a falhas. CFS lida com as falhas dos nodos replicando os blocos naqueles que sucedem o nó por eles responsáveis. Na presença de falha de um nó, seu sucessor passa automaticamente a ser o responsável pela gerência dos blocos anteriormente gerenciados pelo nó falhado. Como esse sucessor já possui uma cópia dos blocos, ele poderá responder às requisições normalmente. Outro objetivo de replicar os blocos é permitir a escolha do nodo mais rápido na hora de transferir o bloco: o cliente solicita a lista de sucessores do Id juntamente com uma estimativa de latência e escolhe o nó com o menor valor para esse parâmetro. Essa estratégia tem também o efeito de balancear o atendimento às requisições entre os nodos que contém uma de suas réplicas. Caso um nó venha a ser sobrecarregado, automaticamente sua latência aumentará e outros nodos passarão a ser escolhidos para transferir o bloco procurado. O nó cujo Id imediatamente sucede o do bloco é quem deve garantir a existência dessas réplicas conforme outros servidores entram e saem da rede P2P. Se essa máquina vier a cair, seu sucessor assume essa responsabilidade.

Tratamento da Heterogeneidade de Recursos: nodos virtuais

Utiliza-se ainda a técnica de nodos virtuais para tratar das diferenças causadas pela heterogeneidade dos recursos de armazenamento. Cada nó recebe um número de identificadores na rede P2P proporcional ao espaço em disco fornecido. Assim, nodos que forneçam grande quantidade de armazenamento se responsabilizam por um número maior de blocos. Para evitar que nodos maliciosos criem identificadores inadvertidamente a fim de dominar a responsabilidade de blocos específicos, CFS impõe que os Ids atribuídos aos nodos sejam o *hashing* do endereço IP concatenado a um índice virtual. Esses dois campos são usados durante a entrada dos nodos para verificar se o identificador foi criado corretamente.

CFS não possui uma operação explícita para remoção de arquivos. Para gerenciar o espaço ocupado pelos blocos, exige-se que o armazenamento destes seja periodicamente renovado. Os blocos expirados ficam sujeitos a serem removidos, ou seja, o armazenamento dos dados é não durável. Esse esquema serve para o sistema se recuperar de ataques onde grandes quantidades de dados são inseridas a fim de esgotar os recursos de armazenamento. Também como uma tentativa de reter esse tipo de abuso, CFS impõe o

uso de cotas fracas. Cada servidor limita o uso local do disco em $\%0.1$ por parte de uma outra máquina qualquer. Assumindo-se que os usuários insiram arquivos sempre através da mesma máquina, um único usuário não conseguiria ocupar todo o espaço em disco de outra máquina. Seria necessário que ele ingressasse o sistema através de 1000 máquinas diferentes para ter sucesso nesse tipo de ataque. Por outro lado, a quantidade em disco total que um único usuário pode usar está limitado à quantidade total de servidores presentes no sistema. Conforme mais máquinas ingressam, maior é a proporção em disco que cada usuário pode utilizar.

Resultados Obtidos

Resultados de testes com 12 servidores parcialmente espalhados geograficamente mostram que, usando-se a técnica de *pre-fetching* com uma janela de $40KBytes$, o CFS é capaz de entregar dados aos clientes tão rapidamente quanto serviços de FTP convencionais. Observou-se também que fazer a escolha do melhor servidor para baixar os blocos otimiza consideravelmente o tempo de transferência para tamanhos de janela de *pre-fetching* pequenos. Outros testes, realizados por simulação, comprovam que a técnica de nodos virtuais melhora a distribuição dos blocos entre os nodos que compõem o sistema. Verifica-se também que o espaço em disco ocupado nos servidores é proporcional ao número de nodos virtuais gerenciados por cada nó. Com relação à técnica de *caching*, simulações simples indicam uma redução significativa no número de nodos médio contactados para concluir operações de localização na rede P2P, porém nada é relatado com relação ao tempo para concluir a operação como um todo (que precisaria levar em conta o tempo para transferir os blocos a serem colocados na *cache*). Ainda através de simulações, comprovou-se que a replicação dos blocos junto ao Chord permite que eles sejam localizados e obtidos com alta probabilidade mesmo quando uma grande quantidades de nodos falha simultaneamente, embora nada tenha sido testado quanto à disponibilidade dos arquivos por inteiro na ocorrência dessas falhas.

3.3 PAST

PAST (ROWSTRON; DRUSCHEL, 2001a; DRUSCHEL; ROWSTRON, 2001) é um serviço de armazenamento para arquivos imutáveis que visa fornecer forte persistência, alta disponibilidade, escalabilidade e segurança para os dados gerenciados. Para obter essas características, emprega-se o uso das técnicas de nodos virtuais, replicação e *caching*.

Estrutura Organizacional

O sistema é composto por máquinas conectadas à Internet que se auto-organizam através de uma rede P2P construída pelo algoritmo Pastry (ROWSTRON; DRUSCHEL, 2001b). Esses nodos **não** são confiáveis, possuem diferenças quanto aos recursos fornecidos e estão sob a tutela de diferentes domínios administrativos. Assim como no CFS, não há a necessidade de recursos mais capacitados para a realização de operações diferenciadas, característica que lhe garante uma organização totalmente descentralizada.

Segurança

O modelo de segurança do PAST recai sobre uma infra-estrutura de chave pública. Tanto nodos quanto usuários recebem um par de chaves (pública e privada) que ficam armazenadas em um *smartcard*, sem o qual não podem ingressar o sistema. A chave

pública armazenada em um cartão é assinada por uma entidade confiável para fins de certificação. Esses cartões são usados para gerar e verificar documentos digitais usados para garantir a segurança das operações solicitadas ao sistema. Eles também armazenam informações de cotas de disco que os usuários tem direito de usar. Assim, sempre que um usuário tem a confirmação da inserção de um arquivo, sua cota é devidamente debitada no cartão. Percebe-se a necessidade de os *smartcards* serem confiáveis: mesmo que um usuário possa controlar uma máquina, ele não pode vir a controlar o comportamento do seu cartão. Os documentos digitais gerados pelos cartões permitem aos nodos e usuários verificar a integridade e a autenticidade dos arquivos armazenados.

Um tipo de ataque muito comum à rede P2P consiste em simular a entrada de máquinas fictícias através da criação ilegal de novos identificadores. Conseqüentemente, o invasor pode vir a dominar a responsabilidade pelo gerenciamento de objetos de dados, possibilitando que ele os remova do sistema. Para evitar esse tipo de ataque, PAST exige que para cada identificador de máquina exista uma chave pública associada. Como apenas a entidade confiável pode emitir novas chaves públicas, evita-se que os identificadores possam ser criados inadvertidamente. É preciso, no entanto, que as entradas das tabelas de roteamento sejam verificadas pelo nodos para validar esses identificadores.

Redundância de Dados: replicação

Utiliza-se a técnica de replicação para garantir a durabilidade dos dados e aumentar a disponibilidade dos mesmos. Ao inserir um arquivo, o usuário indica um número k de réplicas que devem ser geradas. PAST armazena uma cópia do arquivo por inteiro em cada um dos k nodos cujos *nodeIds* (identificadores) estão mais próximos ao *fileId* que identifica o arquivo (os k nodos responsáveis). A invariante de k réplicas é mantida mesmo na presença de desconexões e/ou ingresso de nodos. A replicação também tem o efeito de balancear a carga das requisições pelo arquivo e reduzir latências de acesso devido à forma como o Pastry roteia as mensagens, sempre priorizando a requisição aos nodos que estão mais próximos na rede física que os interconecta. Entretanto, a replicação potencializa problemas devido à heterogeneidade dos recursos de armazenamento e das diferenças nos tamanhos dos arquivos armazenados.

Balanceamento do Espaço de Armazenamento

Como os arquivos são armazenados por inteiro, um dos k nodos por ele responsáveis pode não ter espaço em disco suficiente para gerenciá-lo. Quando isso ocorre, o nó que não pôde armazenar a réplica procura outro com espaço suficiente dentre os seus vizinhos na rede P2P que ainda não estejam armazenando uma cópia do arquivo. Uma espécie de ponteiro é criada para esse nó vizinho, de forma que as requisições pelo arquivo sejam para ele encaminhadas. O objetivo desse esquema é balancear o espaço ainda disponível entre um conjunto de nodos vizinhos, devido às diferenças nos tamanhos dos arquivos gerenciados por cada nó, à heterogeneidade dos recursos de armazenamento e à variação na quantidade de *fileIds* gerenciados por cada nó. Caso não seja possível inserir as k réplicas, mesmo com o uso desses ponteiros, a operação é rejeitada e um novo conjunto de nodos é sorteado. Assim, procura-se balancear o espaço em disco global ainda disponível.

Tratamento da Heterogeneidade de Recursos: nodos virtuais

PAST ainda garante que os nodos que fazem parte de um grupo de vizinhos não apresentem mais do que duas ordens de magnitude de diferença no espaço em disco fornecido,

pois exige que nodos com grande capacidade requisitem mais de um *nodeId*. No entanto, o uso de mais de um *nodeId* requer outras chaves públicas e, conseqüentemente, outros *smartcards*.

Tratamento da Popularidade dos Dados: *caching*

Para tratar a popularidade dos arquivos, PAST faz cópias adicionais em uma *cache*. Dessa forma, existem temporariamente mais do que k cópias para um mesmo arquivo. Ao envolver um maior número de nodos no seu gerenciamento, a carga de requisições por um arquivo tende a ser balanceada. Outra vantagem do uso da *cache* é que cópias mais próximas fisicamente aos nodos requisitantes podem ser criadas, diminuindo latências de acesso. Se um arquivo for popular apenas entre um aglomerado de clientes, então é vantajoso criar uma cópia do mesmo perto desse grupo. A *cache* é criada na parte do disco disponibilizada pelo nó e que ainda esteja livre. Sempre que um nó precisar de espaço para gravar uma nova réplica, ele pode apagar arquivos da *cache*. O objetivo é usar o espaço livre em disco para melhorar o desempenho. Arquivos que deixam de ser populares com o tempo, tendem a ter suas cópias de *cache* substituídas pela de outros arquivos, fazendo com que o sistema apresente característica adaptativa. PAST cria cópias de *cache* em nodos que tenham sido contactados pelo Pastry durante uma requisição por um *fileId*, seja ela tanto na recuperação quanto na inserção dos arquivos.

Resultados Obtidos

Resultados por simulação com 2250 nodos mostram que, ao usar as estratégias de balanceamento de armazenamento entre vizinhos e entre grupo de vizinhos, é possível melhorar o armazenamento global de cerca de 60% para em torno de 95%, com um sobrecusto de gerenciamento aceitável. Observou-se também que as falhas de inserção de arquivos são maiores quando estes são grandes. A técnica de *caching* se mostrou eficaz para diminuir o número de nodos que precisam ser requisitados durante o roteamento feito pelo Pastry, mesmo quando o tamanho da *cache* diminui significativamente conforme a utilização do sistema chega ao seu limite. Diminuir o número de nodos usados significa que um maior número de nodos está respondendo por requisições de um arquivo e que cópias mais próximas puderam ser encontradas, reduzindo latências de acesso e custos para obtenção dos dados. Nesses testes, foram utilizados traços de sistemas de *web proxy* e traços de sistemas de arquivos em uma mesma instituição. Entretanto, não são mostrados resultados com o impacto das transferências dos arquivos.

3.4 Resumo e Análise Comparativa

Para fins de comparação, a Tab. 3.1 mostra resumidamente as principais características apresentadas pelas soluções descritas e pelos ambientes para os quais foram projetadas.

3.5 Necessidade de uma nova proposta

Embora as soluções descritas anteriormente atendam a diferentes exigências feitas por ambientes de larga escala, as características específicas dos ambientes de Grade de Dados como o do HEP as tornam ineficientes ou mesmo inviáveis. Nesses cenários de Grade, existe a necessidade de lidar com massivas quantidades de dados e arquivos de tamanho muito grande, tornando operações que movimentam dados extremamente custosas e

Tabela 3.1: Resumo comparativo entre as soluções analisadas.

	OppStore	**CFS**	**PAST**
Algoritmo P2P	Pastry	Chord	Pastry
Ambiente de Execução	Grade	P2P	P2P
Organização dos Nodos	Misto P2P e centralizado (federação de *clusters*)	P2P (descentralizado)	P2P (descentralizado)
Tipo de Recurso	Estações de trabalho e *Desktops* em laboratórios com boa conectividade	*Desktops* com baixa conectividade	*Desktops* com baixa conectividade
Tipo de Dados	Imutáveis	Imutáveis	Imutáveis
Tipo de Armazenamento	Temporário (*leasing*)	Temporário (*leasing*)	Permanente
Item Armazenado	Fragmento	Bloco (fragmento de tamanho fixo)	Arquivo Completo
Mecanismo de Redundância	IDA para os arquivos e replicação para os metadados	Replicação	Replicação
Controle de Cotas	Não	Parcial (cota fixa para todos)	Sim (uma cota para cada usuário)
Tratamento da Hetoregeneidade de Recursos	Espaço de identificadores virtual	Nodos virtuais	Nodos virtuais e ponteiros para arquivos
Segurança		PKI (distribuição de chaves manual) e documentos digitais	PKI (chaves distribuídas em *smartcards*) e documentos digitais

dificultando o seu gerenciamento. É possível anotar diversos problemas decorrentes da modelagem do CFS, PAST e OppStore caso fossem aplicados nesse ambiente. Por exemplo, o fato de o CFS quebrar os arquivos em fragmentos de tamanho único e pequenos (de alguns pouco *Kilobytes*), implicaria em um potencial grande número de blocos a serem gerenciados pelo sistema para arquivos da ordem de *Gigabytes*, exacerbando os custos de gerenciamento e manutenção para esses arquivos. PAST, por não fragmentar os arquivos, inviabilizaria o seu armazenamento em recursos menos capacitados, além de subutilizar o espaço de armazenamento global. Já OppStore, devido à ausência de replicação de fragmentos, precisaria reconstituir o arquivo por inteiro para simplesmente substituir o fragmento gerenciado por um nó que tivesse falhado, envolvendo assim a movimentação de grandes volumes de dados em suas operações de manutenção. Dessa forma, além de dever ser projetado para evitar a movimentação de dados, o modelo de gerência deve permitir uma boa otimização das operações envolvendo transferências de arquivos grandes.

Existem ainda outras exigências feitas pelo ambiente da Grade de Dados que não são tratadas na íntegra pelas soluções analisadas:

- Necessidade de interoperar com os outros serviços de Grade (por exemplo, ferramentas de monitoramento, serviços para submissão e execução de *jobs*, acesso a certificados e serviços de distribuição de chaves, entre outros);
- Necessidade de usar protocolos de transferência apropriados para redes de grandes distâncias e alta latência (tais como o *GridFTP* (ALLCOCK et al., 2005) e o *bbcp* (HANUSHEVSKY; TRUNOV; COTTRELL, 2001));
- Necessidade de gerenciar metadados na mesma infra-estrutura, de forma a prover eficientemente mecanismos de busca sofisticados baseados nesses atributos (ex.: busca por intervalos);
- Necessidade de mecanismos de proteção quanto a falhas e ataques, implicando em serviços de gerenciamento de dados seguros.

Por fim, existe a necessidade por uma nova proposta que seja capaz de gerenciar simultaneamente e eficientemente recursos típicos de Grade (grande capacidade, alta estabilidade) e recursos típicos de ambientes P2P (pequena capacidade, baixa estabilidade).

4 JAVARMS: CONCEPÇÃO E MODELAGEM

4.1 Introdução

O presente capítulo apresenta a descrição do modelo de um sistema de gerência de dados voltado para aplicações intensivas em dados, dentro do contexto da Computação em Grade. Esse sistema, batizado de JavaRMS (Java *Replica Management System*), tem como objetivo principal reduzir os custos para a construção de Grades de suporte a essa classe de aplicações.

A idéia chave para atingir esse objetivo é agregar a grande quantidade de recursos menos capacitados e de baixo custo ao ambiente de Grade, ainda que utilizando de forma eficiente os recursos mais capacitados que estiverem disponíveis. Como foi mostrado no capítulo anterior, as aplicações intensivas em dados apresentam características únicas que tornam pouco eficientes, ou mesmo inviáveis, as soluções de gerenciamento de dados existentes para esse ambiente-alvo.

Dentre os requisitos funcionais indentificados para um sistema de gerência de dados em Grades, este trabalho foca principalmente nas questões de gerenciamento dos recursos de armazenamento e de distribuição e manutenção dos dados. Em relação aos requisitos não-funcionais, uma maior atenção é destinada a pontos envolvendo segurança, interoperabilidade, escalabilidade e persistência (durabilidade dos dados). Essas escolhas foram feitas com a intenção de limitar o escopo do trabalho e, dessa maneira, possibilitar que os resultados alcançados atendessem aos objetivos estabelecidos inicialmente. Os requisitos escolhidos constituem o núcleo de uma arquitetura proposta para o gerenciamento de dados.

A principal contribuição deste capítulo é a descrição detalhada do modelo do sistema de gerência de dados JavaRMS. O modelo inclui uma estratégia P2P de distribuição e replicação de dados e uma arquitetura de gerenciamento de dados e recursos orientado a serviços interoperáveis (entre si, e entre outros serviços de Grade). Além do modelo, as contribuições originais incluem: um mecanismo através do qual os usuários podem fazer escolhas quanto ao desempenho, disponibilidade e custo de armazenamento para seus arquivos; um esquema de manutenção dos dados baseado num modelo de estados de disponibilidade das máquinas; e algoritmos para inserção, remoção e recuperação de dados que protegem os recursos de armazenamento contra fraudes.

O texto deste capítulo está organizado da seguinte forma: na seção seguinte, apresentam-se as principais exigências feitas pelo ambiente e as técnicas usadas para tratá-las; na seqüência, propõe-se a arquitetura de gerenciamento de dados; posteriormente, cada seção descreve detalhadamente um dos componentes dessa arquitetura. A modelagem do problema de gerenciamento de dados discorre durante o desenvolvimento desses componentes.

4.2 Visão Geral do Sistema

Como a escala do número de recursos da plataforma-alvo é potencialmente grande (da ordem de centenas de milhares a dezenas de milhões), é preciso que o sistema seja auto-organizável e auto-reparável. A fim de ser assim caracterizado, os recursos de armazenamento foram organizados por meio de uma rede P2P estruturada. Entretanto, as funcionalidades P2P foram fornecidas como um componente independente para não quebrar a compatibilidade com os demais serviços de grade. A compatibilidade se faz necessária pois o *middleware* de Grade já fornece parte dos serviços necessários para a concepção do sistema. Pode-se citar a infra-estrutura de segurança, os sistemas de monitoramento e as ferramentas para transferência de dados como exemplos desses serviços.

O sistema utiliza a técnica de nodos virtuais (STOICA et al., 2001) para melhorar a distribuição dos dados entre os nodos e para permitir um esquema simples e eficaz no controle de uso dos recursos de disco e rede. O número de identificadores virtuais que cada máquina recebe é proporcional à quantidade de espaço fornecido com o objetivo de lidar com a grande heterogeneidade de recursos de armazenamento. Os arquivos são quebrados em fragmentos para balancear a carga do atendimento às requisições de transferência e para permitir um maior grau de paralelismo na concepção das mesmas, melhorando assim seu desempenho. Esses fragmentos são também replicados de forma a garantir aos usuários que os arquivos possam ser reconstituídos integralmente, mesmo quando alguns recursos de armazenamento da Grade se encontram indisponíveis.

O modelo contempla ainda diversos aspectos relacionados à segurança, que vão desde comunicação segura até a proteção contra fraudes na realização das operações fornecidas. Há também a exigência do ambiente de Grade em controlar os recursos utilizados por cada usuário. Para atender a esse requisito, JavaRMS fornece um mecanismo de contabilização do uso de disco baseado em cotas. O cadastro dos usuários em si também é gerenciado pelo sistema, pois as ferramentas típicas de gerenciamento de usuários para Grade não atingem a escala desejada ou não fazem o uso de protocolos abertos e interoperáveis com os demais serviços de Grade (BAKER; YU; WLODEK, 2003).

4.2.1 Segurança

Uma grande exigência feita por sistemas distribuídos e que normalmente é negligenciada pelas propostas de gerência de dados é a segurança. Embora muitos desses projetos se limitem a fornecer comunicação segura, a segurança como um todo envolve outros aspectos que também precisam ser considerados num ambiente de Grade como o do HEP (FOSTER et al., 1998). O controle de acesso dos usuários aos recursos, a proteção dos recursos contra fraudes e a proteção das funcionalidades do sistema são exemplos de alguns desses aspectos. De fato, considerar segurança no projeto de um sistema eleva sua complexidade e muitas vezes impõe fortes restrições devido à queda de desempenho provocado por seus protocolos ou a sua dificuldade de gerenciamento (AZZEDIN; MAHESWARAN, 2002; FOSTER et al., 1998).

JavaRMS propõe uma solução segura para a gerência de dados que é baseada na infra-estrutura de chave pública e certificação X509 (ITU, 2000) já fornecida pelo *middleware* de Grade. Essa infra-estrutura já está presente e solidificada em *toolkits* para Grade e em outros ambientes distribuídos, tais como o *Globus Security Infrastructure* (GSI) (GLOBUS, 2008; KANASKAR; TOPALOGLU; BAYRAK, 2005). Ela é usada com o objetivo de atender às seguintes exigências de segurança:

- **Controle de Usuários**: Somente usuários previamente autenticados poderão fazer

o uso dos recursos de armazenamento;

- **Proteção de Dados**: nenhum usuário pode modificar ou remover os dados de outros usuários sem que estes tenham lhe concedido autorização;
- **Proteção de Operações**: sempre que uma operação requisitada por um usuário for validada, o sistema deve garantir que ela seja concluída com êxito, mesmo na ocorrência de falhas de alguns recursos ou de erros provocados por um invasor;
- **Proteção de Recursos**: os recursos de armazenamento devem ter proteção contra fraudes. Ou seja, o sistema não deve permitir que usuários tirem proveito dos serviços fornecidos para ganhar vantagem no uso de recursos da Grade ou prejudicar outros usuários.

Entretanto, assume-se que:

- Serviços básicos fornecidos pelo *middleware* de Grade são também seguros;
- Os algoritmos criptográficos usados são confiáveis e não podem ser violados com o poder computacional hoje disponível.

4.2.2 Balanceamento de Carga

Algumas características da plataforma-alvo podem provocar grandes diferenças na carga dos recursos (de rede ou armazenamento). Quando essas características não são consideradas na solução de gerência de dados, subutilizam-se recursos, degrada-se o desempenho das operações, e pode-se não conseguir comportar um grande número de máquinas. No projeto do JavaRMS, identificou-se como as principais causas de desbalanceamento:

- **Heterogeneidade de Recursos de Armazenamento**: os recursos de armazenamento fornecidos ao sistema possuem grandes diferenças quanto à capacidade, podendo variar de alguns poucos *Gigabytes* até centenas de *Terabytes*;
- **Heterogeneidade de Recursos de Rede**: a conectividade das máquinas pode variar drasticamente, com alguns recursos conectados por redes de baixa latência e alta vazão, enquanto outros estão conectados por redes de baixa velocidade como aquelas encontradas na borda da *Internet*. Ou ainda, os recursos se situam em boas redes mas estas se encontram congestionadas;
- **Tamanho dos Dados**: o tamanho dos arquivos pode variar de algumas dezenas de *Megabytes* até algumas dezenas de *Gigabytes*;
- **Popularidade dos Dados**: a demanda por alguns arquivos pode ser muitas vezes maior que a de outros. Essa demanda pode ainda variar significativamente com o tempo até mesmo para um único arquivo. Tipicamente, dados recentemente coletados pelos experimentos físicos apresentam grande popularidade nos primeiros dias/semanas e muitos meses depois são raramente solicitados (BUNN; NEWMAN, 2003).

O fato de organizar as máquinas através de uma rede estruturada P2P também provoca diferenças quanto à distribuição dos dados (CAI; CHERVENAK; FRANK, 2004).

Essas diferenças são provocadas pelo espaçamento irregular dos identificadores dos nodos, fazendo com que máquinas similares sejam responsáveis por gerenciar proporções diferentes de número de arquivos.

JavaRMS emprega o uso da técnica de **nodos virtuais** (STOICA et al., 2001) para lidar com a heterogeneidade dos recursos de armazenamento e as diferenças provocadas pela estrutura P2P, com o objetivo de melhorar a distribuição dos dados entre as máquinas. Entretanto, para obter um bom balanceamento da carga de armazenamento, ainda se faz necessário usar **fragmentação** de dados (TANENBAUM, 2001; DABEK et al., 2001). Dessa forma, um arquivo muito grande acaba sendo distribuído entre um maior número de máquinas, evitando grandes diferenças quanto ao espaço ocupado em nodos com configurações semelhantes. Como normalmente as características de rede das máquinas são compatíveis com as de armazenamento (ou seja, nodos que fornecem pouco espaço em geral têm baixa conectividade), a fragmentação e o uso de nodos virtuais também balanceiam a distribuição da carga de rede entre os nodos (DABEK et al., 2001). Entretanto, a heterogeneidade de rede é tratada de fato pelos algoritmos de transferência dos arquivos, que requisitam mais dados de máquinas capazes de entregar os dados mais rapidamente. Essa solução só é possível porque os fragmentos são replicados para garantir a sua durabilidade mesmo quando ocorrem falhas permanentes de recursos de armazenamento. Por fim, utiliza-se *caching* para tratar das diferenças de popularidade dos dados, criando-se cópias temporárias dos fragmentos conforme a demanda observada.

4.2.3 Replicação de Dados

A replicação de dados é uma técnica básica de redundância que consiste em criar cópias (réplicas) adicionais dos dados gerenciados (WEATHERSPOON; KUBIATOWICZ, 2002; RODRIGUES; LISKOV, 2005). Seu emprego se faz necessário no JavaRMS por diversos motivos:

- Permitir o acesso aos dados mesmo na ocorrência de falhas temporárias. Ou seja, o sistema quer dar garantias quanto à disponibilidade dos dados gerenciados;
- Prover a durabilidade (persistência) dos dados gerenciados, não permitindo a sua extinção devido à ocorrência de falhas permanentes;
- Viabilizar a construção de protocolos seguros que realizam as operações do sistema;

Embora se utilize *replicação* por motivos de segurança, disponibilidade e durabilidade dos dados, seu emprego tem impacto também no desempenho. Ao gerar novas cópias para os arquivos, aumenta-se o número de máquinas envolvidas no seu gerenciamento e, dessa forma, viabiliza-se o atendimento a uma maior demanda pelos dados. Maiores níveis de paralelismo nas transferências dos dados também são possíveis pelo uso dessa técnica, pois as réplicas são colocadas em máquinas diferentes. A replicação também diminui latências de acesso aos dados, devido a maior probabilidade de alguma cópia do arquivo estar próxima ao cliente (BUNN; NEWMAN, 2003).

Com o objetivo de determinar aonde as réplicas dos dados serão criadas e como elas serão posteriormente localizadas, utiliza-se uma estratégia simples baseada nas informações de roteamento P2P. Para todos os objetos de dados gerenciados pelo JavaRMS existe uma `chave` associada que é usada como entrada para a operação de localização (*lookup*) fornecida pelo *middleware* P2P. Relembrando que essa operação retorna a lista dos k nodos mais próximos à `chave`, a estratégia de replicação adotada consiste em criar cópias nos R primeiros nodos dessa lista, onde R é o número de réplicas desejado. Diz-se

que os nodos dessa lista formam um **grupo responsável** pelo gerenciamento do dado em questão.

Embora existam outras estratégias de replicação que resultem numa melhor distribuição dos dados entre as máquinas que compõem a rede P2P (BYERS; CONSIDINE; MITZENMACHER, 2003; GOPALAKRISHNAN et al., 2004), optou-se por uma estratégia que ainda permita o emprego de outros mecanismos para lidar com a distribuição dos dados. Esses mecanismos consideram a heterogeneidade dos recursos e o tamanho dos dados, fatores tipicamente negligenciados por estratégias de replicação P2P mais sofisticadas que lidam apenas com parâmetros de composição da rede lógica P2P.

A grande desvantagem de usar replição é o uso adicional dos recursos de armazenamento. Outras técnicas de redundância mais sofisticadas, tais como os Algoritmos de Dispersão de Informação (*Information Dispersal Algorithms* – IDA) (PLANK, 1997; RABIN, 1989), conseguem fornecer uma melhor disponibilidade dos dados ainda que usando menos espaço de armazenamento. Embora seu emprego seja desejável, existem custos associados à codificação dos dados que não são desejáveis quando o tamanho dos mesmos é pequeno. Ainda assim, tanto o uso de IDA quanto o de técnicas de compactação poderiam ser empregadas ortogonalmente ao modelo de replicação aqui utilizado.

No JavaRMS, o modelo de replicação proposto é usado tanto para os fragmentos quanto para as informações que descrevem os arquivos propriamente ditos (seus metadados principais). O sistema de gerenciamento de usuários e cotas também utiliza replicação para dar persistência aos objetos de dados que representam os usuários e suas credenciais da infra-estrutura de segurança da Grade.

4.3 A Arquitetura

A Fig. 4.1 mostra como a arquitetura orientada a serviços está organizada. As camadas mais de baixo estão mais próximas ao *hardware*, enquanto que as de cima lidam com serviços de mais alto nível, voltadas diretamente para o usuário. Um componente em uma determinada camada apenas utiliza serviços fornecidos por componentes da mesma camada e/ou de camadas inferiores.

A camada mais de baixo contém componentes diretamente ligados ao *hardware*, onde os serviços são freqüentemente fornecidos pelo sistema operacional. A segunda camada, chamada de Serviços de Base, lida com os serviços básicos fornecidos pelo *middleware* de Grade e pelo *software* que organiza os nodos em uma rede lógica P2P. A camada intermediária (Serviços de Gerenciamento) contém o núcleo da arquitetura para gerência de dados, sendo o principal foco de projeto deste trabalho. Ela inclui, por exemplo, o gerenciador de arquivos e o de informações sobre usuários. Por fim, a camada superior visa fornecer serviços voltados ao usuário final e suas aplicações, tais como abstrações para sistemas de arquivos.

Embora essa arquitetura envolva a concepção de um grande número de componentes, este trabalho tem o foco principal em apenas um subconjunto dos mesmos: a Gerência de Arquivos, a Gerência de Usuários e Cotas, a Gerência de Recursos e os Serviços de Comunicação. Entretanto, um conjunto mínimo de funcionalidades de alguns dos outros componentes precisou ser projetado para viabilizar a concepção dos componentes principais. De fato, o *middleware* P2P, presente na camada de Base, precisou passar por alterações para fornecer as funcionalidades necessárias para os demais componentes. Quanto à Gerência das Transferências, um modelo simples foi concebido para viabilizar um conjunto mínimo de operações de movimentação de dados. O Serviço de Metadados e os

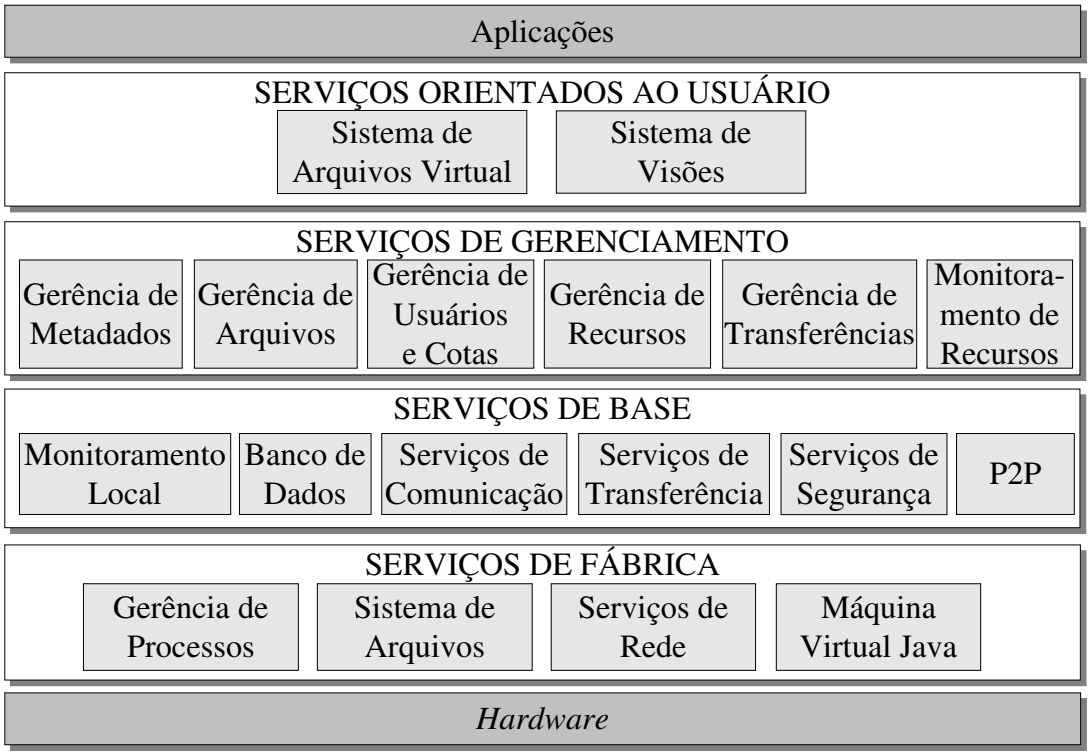

Figura 4.1: A arquitetura do JavaRMS orientada a serviços.

serviços da camada superior não foram modelados e, assim como os serviços de gerência de transferências, fazem parte do trabalho do doutorando Marko Petek (membro do grupo de pesquisa do qual o autor faz parte no GPPD). A arquitetura em si também foi projetada em conjunto com ele, onde a versão proposta neste trabalho representa uma evolução de trabalhos anteriores (PETEK et al., 2006).

Após um resumo das funcionalidades de cada serviço a seguir, apresenta-se apenas os serviços principais desenvolvidos neste trabalho ou versões simplificadas dos outros serviços a eles necessários.

Serviços de Base

- **Monitoramento Local**: fornece informações básicas de monitoramento no nó, tais como uso de disco e rede;

- **Banco de Dados**: responsável por fornecer persistência local à informações de gerenciamento utilizadas pelos demais componentes. Essas informações vão desde metadados até certificados que identificam os usuários da grade. Os fragmentos que compõem os arquivos em si **não** são armazenados neste componente;

- **Serviços de Comunicação**: também chamados de serviços de interoperabilidade, fornece comunicação segura através de *web services*;

- **Serviços de Transferência**: constitui-se das ferramentas básicas para movimentação de dados presente no *middleware* de Grade. Exemplos dessas ferramentas são o *Globus GridFTP* e o *bbcp*;

- **Serviços de Segurança**: contém a infra-estrutura de segurança baseada em certificação X509 e chave pública. Constitui-se das ferramentas da Grade usadas para gerenciar certificados e chaves criptográficas.

- **P2P**: estrutura as máquinas em uma rede lógica P2P e fornece uma operação (*lookup*) para localizar as máquinas responsáveis pelo gerenciamento de objetos de dados. É sua função também disparar eventos conforme os nodos entram/saem dessa rede.

Serviços de Gerenciamento

- **Gerência de Metadados**: tem o objetivo de fornecer meios pelos quais os usuários possam identificar os arquivos que contêm os dados por eles procurados. Gerencia um conjunto de informações sobre os arquivos (metadados) de forma distribuída e permite a realização de consultas baseadas nesses atributos;

- **Gerência de Arquivos**: uma vez que os usuários sabem quais arquivos contêm os dados procurados, este componente fornece meios para localizá-los, reconstituí-los e/ou removê-los. Também permite que novos arquivos sejam armazenados no sistema. Dá garantias quanto à disponibilidade, persistência e segurança dos arquivos gerenciados;

- **Gerência de Usuários e Cotas**: cria um cadastro distribuído dos usuários da Grade, atribuindo a eles cotas individuais de uso de disco. Também contabiliza o espaço utilizado por cada um desses usuários;

- **Gerência de Recursos**: permite o controle do uso dos recursos (rede e disco) segundo políticas locais. Responsável também pelo gerenciamento dos nodos virtuais e pelo fornecimento de uma interface para o uso das funcionalidades de localização providas pelo componente P2P;

- **Gerência de Transferências**: recebe um conjunto de arquivos e locais da grade onde existem cópias dos fragmentos que os constituem e organiza as transferências de forma a otimizá-las. Responsável também por acordar um protocolo para movimentação dos dados entre os nodos;

- **Monitoramento de Recursos**: fornece uma interface única para informações sobre recursos locais e também sobre a entrada/saída de nodos na rede P2P. Permite ainda que outros serviços sejam avisados quando determinadas condições de monitoramento são atingidas.

Serviços Orientados ao Usuário

- **Sistema de Arquivos Virtual**: mapeia as operações típicas fornecidas por sistemas de arquivo convencionais em um conjunto de operações de arquivamento. O objetivo é permitir a compatibilidade entre aplicações já existentes e facilitar a navegação do usuário entre os arquivos gerenciados pelo sistema.

- **Sistema de Visões**: fornece um mecanismo de alto nível de abstração para a busca de dados baseado no conceito de *visões* (SLAZINSKI, 2001), que advém de trabalhos da área de bancos de dados. A idéia é permitir que os usuários procurem pelos dados como se estivessem fazendo consultas SQL. Também permite a compatibilidade com aplicações que gerenciam dados através de bancos de dados.

4.4 Serviços de Comunicação

O componente de comunicação tem o objetivo de viabilizar a interoperabilidade entre os serviços situados em nodos distintos. Ele fornece uma *interface* baseada em *web services* para as funcionalidades dos diferentes componentes que compõem a arquitetura. Ou seja, é este componente quem caracteriza o sistema como sendo orientado a serviços. Também é sua função prover mecanismos para que o nó em si possa contactar serviços remotos. A comunicação se dá de forma segura: há garantias quanto à autenticidade e à integridade das informações que estão sendo trafegadas.

Todos os serviços da arquitetura que são disponibilizados por meio desse componente são chamados de **serviços RMS**. Este termo será utilizado para diferenciar os serviços daqueles fornecidos pelo *middleware* de Grade e P2P. A Fig. 4.2 mostra como se dá a comunicação entre esses diferentes tipos de serviço.

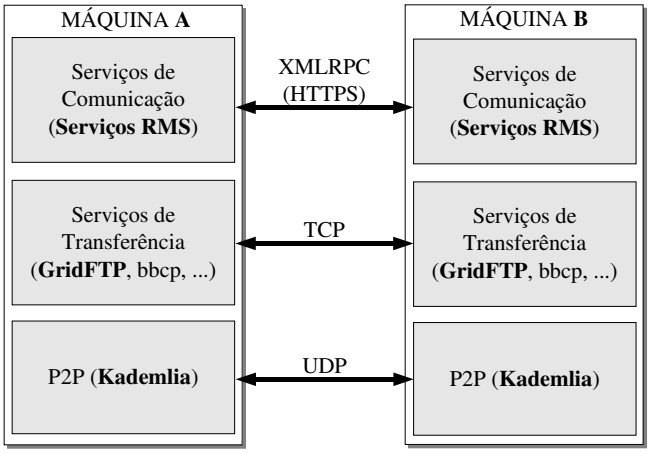

Figura 4.2: Protocolos usados para comunicação entre um par de nodos, de acordo com o tipo de serviço oferecido.

Nota-se que a comunicação envolvendo transferências de arquivos e envolvendo a rede P2P **não** é realizada pelo protocolo usado nos serviços RMS. Os motivos para essa escolha são:

- A necessidade de manter compatibilidade com os serviços de Grade (ex.: serviços básicos de segurança e serviços de transferência de arquivos) e P2P já existentes para que possam ser utilizados pela arquitetura;
- Usar os serviços RMS para transferir os arquivos propriamente ditos seria inviável devido ao baixo desempenho do *web services* (CHEN et al., 2006);
- Uma vez que as operações P2P são tipicamente realizadas por troca de mensagens pequenas que exigem um tempo de resposta pequeno, o uso de *web services* para realização dessas operações impactaria significativamente no seu desempenho.

O objetivo de assim estruturar a comunicação é atender a um bom compromisso entre interoperabilidade e desempenho. Quando um nó deseja, por exemplo, transferir um arquivo para outra máquina, primeiramente utilizam-se os serviços RMS para negociar um protocolo de movimentação de dados (ex.: *GridFTP*). Uma vez negociado, os dados são transferidos diretamente por esse protocolo, sem ter que arcar com o sobrecusto dos *web*

services. Nota-se que essa negociação pode inclusive levar em consideração a existência de *firewalls*. Nesse caso, os nodos escolhem um protocolo para transferência que não esteja sendo filtrado. Caso nenhum seja possível, um serviço RMS próprio poderia ser utilizado para este fim, porém com a desvantagem de perder em desempenho. Protocolos para *web services* são, em geral, bem aceitos por *firewalls*.

A segurança da comunicação entre serviços RMS se dá através do uso do protocolo TLS (DIERKS; RESCORLA, 2006) (do Inglês, *Transport Layer Security*), também conhecido pela sigla SSL devido aos protocolos que o antecederam. Garante-se assim a integridade e a autenticidade dos dados trafegados. O canal de comunicação seguro se faz necessário pois, do contrário, os pacotes poderiam ser forjados por um terceiro elemento a fim de quebrar as funcionalidadades dos serviços RMS. Por exemplo, ao receber uma solicitação de armazenamento de um arquivo, o nó precisa requisitar a cota do usuário para saber se tem direito de gravar tal arquivo. Caso essa informação fosse interceptada e o valor da cota disponível alterado, o nó poderia armazenar o arquivo de um usuário mesmo quando sua cota fosse insuficiente. As credenciais usadas pelas entidades que estão tentando se comunicar são as mesmas usadas na Grade, ou seja, os certificados X509 criados para usuários e serviços. Portanto, para que um elemento qualquer utilize serviços RMS, ele antes precisa obter credenciais junto à Autoridade Certificadora da Grade (*Certificate Authority* – CA).

Apesar de a comunicação entre serviços RMS ser segura, ela não tem influência sobre o protocolo de comunicação usado entre serviços de transferência e entre serviços P2P. Se a ferramenta escolhida para a transferência for o FTP, por exemplo, não há garantias quanto a autenticidade e integridade dos dados transferidos. No entanto, JavaRMS calcula o *hashing* do conteúdo do arquivo transferido e o confere com o valor informado através de serviços RMS. Embora essa solução ainda esteja sujeita a ataques onde diferentes arquivos mapeiam para um mesmo valor de *hash*, o fato de usar uma função de *hashing* dita segura (nesse caso, SHA-256) dificulta em muito esse tipo de ataque. Pode-se ainda utilizar o canal seguro RMS para negociar uma chave a ser usada junto à função de *hashing* (por exemplo, utilizar HMAC256 (KRAWCZYK; BELLARE; CANETTI, 1997)), garantindo assim a autenticidade e integridade do arquivo tranferido. De forma análoga, a comunicação entre os serviços P2P pode ser interceptada e alterada por um invasor. Entretanto, mecanismos de segurança foram modelados junto ao algoritmo P2P escolhido para garantir seu funcionamento com segurança.

4.5 Serviços P2P

É através dos serviços P2P que a arquitetura para gerência de dados emprega um modelo P2P para a organização dos nodos e para a localização de objetos de dados. JavaRMS utiliza Tabelas Hash Distribuídas (*Distributed Hash Table* – DHT) (BALAKRISHNAN et al., 2003) para construir uma rede de sobreposição estruturada.

Utilizou-se como `NodeID` o *hashing* do resultado da concatenação do endereço IP do nó a um contador. Já uma `chave`, é sempre o *hashing* de alguma informação que identifique unicamente o objeto de dados. No JavaRMS, um objeto de dados pode ser um documento que descreve um arquivo, um fragmento de um arquivo ou um documento que descreve um usuário. Respectivamente, a informação que identifica unicamente esses objetos de dados é: a concatenação do nome do arquivo (*Logical File Name* – LFN) ao nome do usuário que o registrou no sistema, o conteúdo do fragmento de arquivo referenciado, e o nome do usuário.

A fim de identificar qual algoritmo P2P proporcionaria uma melhor distribuição para os dados, comparou-se Chord, Pastry e Kademlia através de simulações. Essas simulações consideraram também a possibilidade de os dados estarem replicados segundo o modelo de replicação adotado no JavaRMS. Os resultados obtidos (GOMES; PETEK; GEYER, 2007) levaram à escolha do algoritmo Kademlia (para detalhes de funcionamento do Kademlia, ver o Apêndice A).

4.6 Serviços de Monitoramento

Os Serviços de Monitoramento tem o objetivo de capturar e formatar informações básicas de monitoramento, disponibilizando-as através de serviços RMS ou permitindo que eventos sejam disparados quando condições específicas forem observadas. A idéia é complementar os serviços de monitoramento já existentes na Grade disponibilizando informações específicas necessárias para a gerência de dados.

Essas necessidades envolvem um serviço para monitoramento da disponibilidade dos nodos, necessários para o sistema poder dar garantias quanto à disponibilidade dos dados gerenciados, e um serviço para monitoramento da troca de estados dos nodos remotos que são vizinhos na rede P2P, para viabilizar a execução de operações de manutenção. As informações de monitoramento básicas da Grade relevantes para o JavaRMS são o uso do disco e da rede, usados para o controle desses recursos por parte de seus administradores. Complementam ainda os serviços de monitoramento o Serviço de Consulta, que é uma *interface* RMS para as informações disponibilizadas; o Serviço de Aviso, que dispara eventos quando condições específicas são monitoradas; e o Serviço de Histórico, que compila as informações monitoradas e as armazena para consulta posterior. A arquitetura desse componente é mostrada na Fig. 4.3.

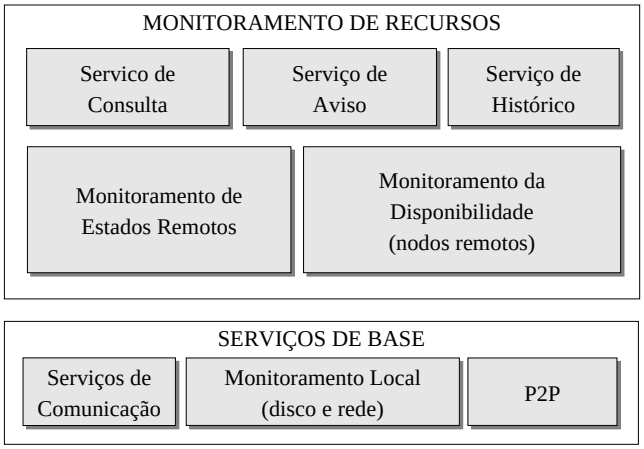

Figura 4.3: Arquitetura dos Serviços de Monitoramento.

4.6.1 Monitoramento de Estados Remotos

O Serviço de Monitoramento de Estados Remotos surge da necessidade que os outros serviços da arquitetura têm de executar procedimentos de manutenção devido à entrada/saída de nodos que gerenciam dados em comum. Por exemplo, quando é detectado que um nó falhou permanentemente, precisa-se que sejam criadas novas cópias para os arquivos anteriormente gerenciados pelo nó falhado para assegurar sua durabilidade.

Como essas operações potencialmente envolvem a movimentação de grandes quantidades de dados, a entrada/saída de nodos com alta freqüencia impactaria significativamente no desempenho do sistema.

Com o objetivo de suportar falhas temporárias para diminuir o impacto da volatilidade das máquinas sobre os procedimentos de manutenção, propõe-se um modelo de falhas baseado em estados.

Modelo de Estados

Nesse modelo, cada nó pode assumir um dos 3 estados possíves (Fig. 4.4):

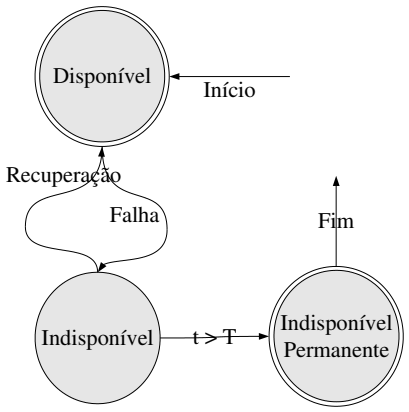

Figura 4.4: Modelo de estados possíveis para um nó conforme a sua disponibilidade.

Quando o nó ingressa o sistema pela primeira vez, ele inicialmente vai para o estado `disponível`. Esse estado indica que o nó está ativo e apto a gerenciar dados. Caso o nó venha a falhar, ele passa para o estado `indisponível`, indicando que o nó saiu temporariamente do sistema e que não poderá atender a requisições pelos seus dados. Se ele se recuperar da falha dentro de um tempo T, ele volta para o estado `disponível`. Caso contrário, ele vai para o estado `indisponível permanente`. Esse estado significa que o nó estará impossibilitado de gerenciar dados por tempo indeterminado, exigindo então que outro nó do sistema assuma a responsabilidade de gerenciamento dos seus dados.

O parâmetro T deve ser ajustado para refletir a volatilidade do nodos. Em uma Grade mais controlada, onde a probabilidade de ocorrer uma falha é menor, deve-se ajustá-lo para valores pequenos (ex.: $T = 12h$). Quando a ocorrência de falhas é grande, no entanto, precisa-se que T assuma valores maiores (ex.: $T = 48h$). A idéia é que se a ocorrência de falhas é muito incomum, o simples fato de um nó ficar indisponível por poucas horas significa que o motivo de sua indisponibilidade não se deu devido a um procedimento de *reboot* ou outra operação administrativa, e sim porque algo de errado realmente aconteceu e o incapacitará de atender a resquisições por um longo período. Analogamente, tolera-se um tempo maior de indisponibilidade quando saídas de nodos são mais comuns para que seja possível identificar falhas maiores.

Com a modelagem de falhas através de estados é possível que os nodos mantenham a invariante de R cópias para os dados da seguinte forma. Sempre que o nó entrar no estado de disponível, ele primeiramente verifica junto a seus vizinhos quais os dados que precisa gerenciar, requisitando-os caso ele ainda não possua uma cópia. Seus vizinhos também

verificam se deixam de fazer parte do grupo de R nodos caso o nó esteja ingressando pela primeira vez, habilitando-os a remover sua cópia caso essa saída do grupo se confirme. Quando o nó se torna indisponível, nenhuma outra cópia é gerada pois espera-se que a máquina retorne dentro de um tempo (T) previsto. Entretanto, ao perceber que o nó não mais retornará, uma nova cópia deverá ser feita pelo nó que tomar o seu lugar no grupo dos R nodos. Garante-se assim a durabilidade dos dados ainda que suportando indisponibilidades temporárias para evitar o alto custo dos procedimentos de manutenção.

É importante observar que são os **vizinhos** de um nó na rede P2P que controlam o seu estado, disparando os eventos de manutenção somente caso determinadas mudanças de estado ocorram. Caso o próprio nodo controlasse seu estado, ele poderia forjá-lo a ponto de sobrecarregar os nodos vizinhos ou prejudicar usuários. Por exemplo, ele poderia forçar a troca rápida de estados para fazer os demais nodos dispararem os procedimentos de manutenção repetidamente. Outro problema é que uma máquina desconectada não tem como avisar seu estado para seus vizinhos, que não teriam como discernir entre o estado `indisponível` e `indisponível permanente`. O fato de os vizinhos controlarem o estado de um nó também evita a troca de mensagens. Somente esses vizinhos na rede P2P tem interesse em conhecer as trocas de estado do nó em questão, pois nodos distantes não gerenciam dados em comum.

Por fim, observa-se também que existe o controle de estado por parte dos vizinhos para cada **nó virtual**. Isso se faz necessário pois os administradores dos recursos podem retirar provisoriamente alguns de seus identificadores virtuais para controlar o uso de seus recursos. Por exemplo, se o administrador do recurso observa que está com pouco espaço em disco, ele pode desabilitar vários identificadores virtuais por um longo período até que novamente consiga liberar espaço suficiente. Assim, nodos virtuais vizinhos identificam a ausência permanente dos identificadores e requisitam a criação de novas cópias para os dados.

4.6.2 Monitoramento da Disponibilidade

Uma vez que JavaRMS quer dar garantias quanto à disponibilidade dos dados gerenciados, precisa-se de um mecanismo de monitoramento dessa característica para os nodos do sistema. Assim como no monitoramento dos estados, cada nó monitora a disponibilidade de seus vizinhos na rede P2P. Esses vizinhos são os nodos que de fato gerenciarão dados em comum. O serviço recai novamente sobre as informações dos nodos na rede P2P para modelar a disponibilidade desses vizinhos.

As mensagens recebidas dos vizinhos são tratadas como um sinal de *keep-alive*. Em períodos onde nenhuma mensagem é recebida, atribui-se ao vizinho o estado de *indisponível*. Nos demais períodos, considera-se que o nó vizinho encontra-se no estado *disponível*. Casos extremos onde nenhum nó envia mensagens para os demais vizinhos são impossíveis pois os protocolos de manutenção da rede P2P já obrigam a troca de mensagens em intervalos fixos.

A *disponibilidade* de um nó é dada por $\alpha = \frac{tempo\_disponivel}{T}$, onde T é um intervalo para o qual se calculará a disponibilidade (tipicamente, $T = 24$ horas). Valores de T grandes demais (ex.: $T = 1$ semana) não capturam as variações bruscas da disponibilidade, porém dão uma idéia geral da disponibilidade do nó durante uma fase completa. Ou seja, dão a disponibilidade durante o período de um ciclo a partir do qual suas características tendem a se repetir. Já quando T é pequeno demais (ex.: $T = 1h$), consegue-se ter uma idéia da disponibilidade "instantânea" do nó, porém esse valor tende a ter grande variabilidade de uma amostragem para a outra.

Relembrando que o monitoramento de estados remotos armazena o tempo total que o nó vizinho esteve em cada estado desde o momento em que entrou no sistema, para calcular a sua disponibilidade basta requisitar essa informação a cada período T, assim: $tempo\_disponivel = soma\_tempo\_disp_t - soma\_tempo\_disp_{t-1}$. Na prática, os serviços de monitoramento da disponibilidade e de monitoramento de estados remotos poderiam ser concebidos como um serviço único.

4.6.3 Serviço de Aviso

Com o objetivo de fornecer uma ferramenta de monitoramento assíncrona, o Serviço de Aviso permite que outros componentes do sistema registrem eventos a serem disparados sempre que determinadas condições forem observadas. Ele é a base para que os protocolos de manutenção sejam executados. Por exemplo, quando um nó vizinho falha permanentemente, o serviço de aviso dispara um procedimento de manutenção que se encarregará de criar uma nova cópia para os dados anteriormente gerenciados pelo nó que falhou.

As informações básicas monitoradas e que podem ser usadas para determinar quando um evento será disparado são:

- Ocupação de disco;
- Carga de ocupação da rede;
- Disponibilidade de algum nó vizinho;
- Alguma mudança de estado de vizinhos.

O serviço pode ainda ser usado para disparar eventos quando as informações básicas monitoradas se encontrarem abaixo ou acima de um limiar. Por exemplo, registra-se um evento a ser disparado quando a disponibilidade do i-ésimo vizinho ficar abaixo de 90%. A única exceção desse tipo de evento é para mudanças de estado de vizinhos.

4.6.4 Serviço de Consulta

O Serviço de Consulta tem o objetivo de fornecer uma *interface* para que as informações monitoradas pelos demais serviços de monitoramento e pelas ferramentas da Grade possam ser disponibilizadas através de serviços RMS. Futuramente, este serviço também pode agrupar informações, executar filtros e comprimir os dados.

4.6.5 Serviço de Histórico

O objetivo do Serviço de Histórico é permitir que informações antigas monitoradas possam ser resgatadas e disponibilizadas. Por exemplo, ele poderia ser utilizado por uma aplicação que precisasse analisar as disponibilidades das máquinas para fazer alguma previsão. Assim, os valores de disponibilidade medidos nas últimas semanas poderiam ser utilizados para prevê-la na semana seguinte.

4.7 Gerência de Transferências

Os Serviços de Gerência de Transferências têm como objetivo principal gerenciar a movimentação de dados entre os nodos de forma a melhorar seu desempenho. É também sua função viabilizar essa movimentação de dados fazendo com que as máquinas

envolvidas negociem um protocolo de transferência. A arquitetura é composta de dois componentes (Fig. 4.5): o Serviço de Negociação de Protocolo e o Serviço de Transferências.

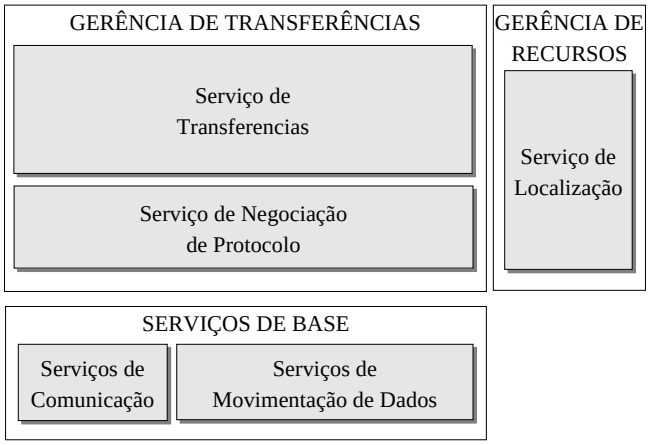

Figura 4.5: Arquitetura dos Serviços de Gerenciamento de Transferências

4.7.1 Serviço de Negociação de Protocolo

Como no ambiente de Grade existe uma grande diversidade de protocolos usados para transferir arquivos, torna-se necessário que as entidades envolvidas no procedimento de movimentação acordem sobre qual desses protocolos utilizar. Mesmo que uma única ferramenta fosse utilizada por todas as máquinas de Grade (ex.: *GridFTP*), seria ainda preciso que as máquinas trocassem outras informações para permitir o seu uso e/ou otimizá-lo. Por exemplo, os nodos precisariam saber em que porta o servidor do *GridFTP* aguarda por conexões. Ou ainda, que tamanho dos *buffers* e número de conexões paralelas cada um deve utilizar (parâmetros tipicamente suportados pelas ferramentas de transferência de arquivos para Grades).

O uso de protocolos específicos para movimentação de dados se faz necessário pois uma solução baseada em serviços RMS teria um baixo desempenho devido ao sobrecusto da comunicação via *web services*. Uma vez que existe uma diversidade de ferramentas já consolidadas na Grade para transferir arquivos, não se justifica também impor outra ferramenta não baseada em *web services*. Mesmo que um protocolo próprio fosse forçado para tirar vantagem de uma característica não adequadamente explorada, os outros já existentes continuariam a ser usados pelos demais serviços da Grade.

O Serviço de Negociação de Protocolo fornece operações para detecção e cadastro de protocolos de movimentação juntamente com uma operação específica para negociar um dos protocolos com uma entidade remota. Com relação a esta operação de negociação, o resultado pode ainda ser armazenado em uma *cache* para fins de otimização pois a probabilidade de a ferramenta de movimentação de dados fornecida por uma máquina mudar em um curto intervalo de tempo é muito pequena. Consegue-se assim manter compatibilidade com ferramentas já existentes ainda que garantindo a interoperabilidade entre as entidades remotas sem implicar uma queda de desempenho significativa.

4.7.2 Serviço de Transferências

O tempo necessário para transferir arquivos entre as máquinas domina o tempo total da maioria das operações fornecidas pela arquitetura de gerência de dados. Especificamente no caso da Grade do HEP, onde o tamanho dos arquivos freqüentemente atinge a ordem de *Gigabytes*, as operações que envolvem a movimentação de arquivos se tornam críticas. Com o objetivo de otimizar essas transferências, o Gerente de Transferências se faz valer do fato de os arquivos serem fragmentados e replicados em diferentes nodos da Grade para transferir diversos trechos do arquivo em paralelo. O paralelismo das transferências compensa a heterogeneidade dos recursos de rede, maximizando a utilização desses recursos e viabilizando uma maior vazão de dados.

O Serviço de Transferências fornece uma única operação: `transfer(arqid, fontes, arqlocal)`. Essa operação recebe como entrada um identificador de um arquivo (sua chave) e uma lista de fontes (endereços de máquinas) que contêm cópias de seus fragmentos, transferindo os trechos em paralelo para um arquivo local também informado como parâmetro. Cabe ao modelo de transferência adotado decidir quando e como cada uma das fontes será utilizada para maximizar a vazão dos dados. Deseja-se também que o algoritmo baseado nesse modelo considere as variações de carga da rede que podem ocorrer nas fontes durante as transferências. Assim, caso uma fonte se torne lenta durante uma transferência, o tamanho do seu trecho seria reajustado para que a transferência terminasse junto com as demais fontes. A idéia do algoritmo de transferência é, portanto, requisitar o *download* de trechos maiores para fontes capazes de dar maior vazão aos dados, adaptando-os às variações da rede.

Tal algoritmo poderia ainda levar em conta o fato de diversas máquinas estarem requisitando a transferência de um mesmo arquivo simultaneamente para otimizar ainda mais o procedimento. Assim, as máquinas contribuiriam com as demais disponibilizando para *download* os trechos recentemente por elas transferidos, relembrando o que acontece no protocolo *BitTorrent* (COHEN, 2003). Entretanto, a elaboração de tal modelo e a especificação de um algoritmo para transferência com esse nível de sofisticação está fora do escopo deste trabalho, sendo tratado na tese do doutorando Marko Petek (membro do grupo de pesquisa do qual o autor faz parte). Planejou-se então um algoritmo mais simples a fim de viabilizar o Serviço de Transferências.

Inicialmente, obtém-se junto a cada uma das fontes os dados necessários para poder utilizá-las, tais como a ferramenta de transferência negociada e o endereço de onde pegar o arquivo. Num segundo momento, inicia-se a transferência em paralelo de todos os fragmentos do arquivo. Caso os fragmentos estejam replicados, eles são divididos em tantos trechos quantas forem as fontes contendo uma de suas réplicas. Cada um desses trechos é também transferido em paralelo, porém nenhum ajuste é feito quando termina a transferência de um deles primeiro que a de outro. A terceira e última etapa consiste em aguardar o término dessas transferências.

O desempenho do algoritmo proposto fica limitado ao tempo necessário para transferir o trecho da fonte mais lenta. No entanto, poderia-se utilizar os mecanismos de monitoramento da Grade para dividir o tamanho dos trechos proporcionalmente à capacidade da fonte em dar vazão aos dados, fazendo com que as fontes lentas recebam trechos menores e as fontes rápidas recebam trechos maiores. Assim, conseguiria-se aproveitar melhor a ociosidade das fontes para diminuir o tempo total de transferência. Caso não ocorressem variações na carga da rede das fontes, as transferências tenderiam a ser finalizadas ao mesmo tempo.

4.8 Serviços de Gerência de Recursos

O componente de Gerência de Recursos tem o objetivo de tratar da heterogeneidade dos recursos (rede e disco) fornecidos ao sistema para melhorar seu aproveitamento, ainda que sem desobedecer às políticas locais determinadas por seus administradores. A idéia é fazer com que recursos semelhantes recebam cargas semelhantes, e recursos diferentes recebam cargas proporcionais à capacidade fornecida. É também sua função dar mecanismos pelos quais os administradores possam controlar o uso de seus recursos. Para atingir esses objetivos, o componente utiliza os serviços P2P para descoberta e localização de recursos juntamente com o uso da técnica de nodos virtuais.

A Fig. 4.6 mostra como o componente está estruturado. Ele é formado por três unidades menores: o Gerente de Nodos Virtuais, o Serviço de Localização e o Controle de Recursos Locais. O primeiro deles lida diretamente com os serviços P2P e é usado exclusivamente pelos demais serviços da gerência de recursos. Ele tem como função prover a funcionalidade de nodos virtuais não fornecida pelo *middleware* P2P. O Serviço de Localização é uma *interface* para o uso da operação de *lookup*, fazendo a tradução de objetos de dados para chaves a serem procuradas pelos serviços P2P. Já o Controle de Recursos Locais propõe meios através dos quais o administrador do recurso pode estabelecer políticas de uso do mesmo. Tanto o Serviço de Localização quanto o de Controle de Recursos Locais expõem suas operações através de serviços RMS e, portanto, utilizam-se dos serviços de comunicação da camada de Base.

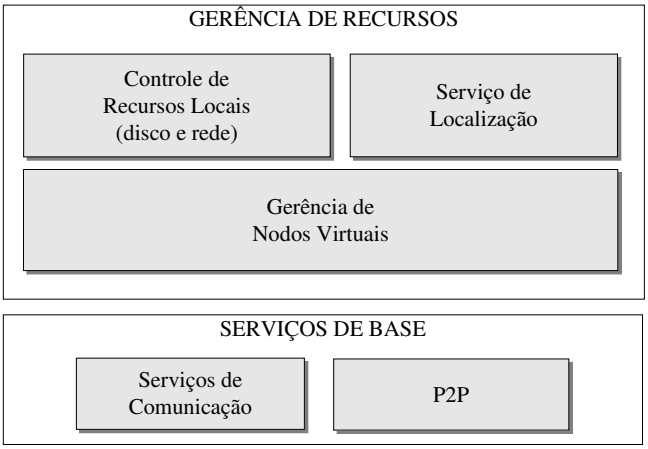

Figura 4.6: Arquitetura dos Serviços de Gerência de Recursos

4.8.1 Gerência de Nodos Virtuais

Num ambiente de Grade como o do HEP, podem existir recursos de armazenamento de alguns poucos *Gigabytes* até sistemas de armazenamento em massa da ordem de *Terabytes* ou até mesmo *Petabytes* de dados. As redes que os interconectam são também bastante heterogêneas, sendo comum encontrar conexões de grandes distâncias onde o produto *latência*$\times$*banda* é elevado (BUNN; NEWMAN, 2003).

Caso cada nó do sistema gerenciasse uma mesma quantidade de objetos de dados, os recursos de armazenamento maiores certamente seriam subutilizados. Mesmo que esses recursos fossem semelhantes, o fato de usar uma rede lógica P2P para mapear os objetos aos recursos cria diferenças quanto à quantidade de dados gerenciados por cada um. Nodos que gerenciam maiores quantidades de objetos (arquivos/fragmentos e usuários),

além de diretamente precisarem gastar mais espaço em disco precisam também atender a um maior número de requisições e devem ser capazes de dar vazão aos dados, exigindo assim mais de sua rede.

Na concepção do JavaRMS, tais diferenças foram tratadas através do uso da técnica de **nodos virtuais** (STOICA et al., 2001) juntamente com um **modelo de capacidade** para recursos. A técnica consiste em atribuir mais de um identificador P2P para cada nó do sistema, de forma que ele se responsabilize por mais de um conjunto de chaves do espaço de nomes P2P. Quanto mais chaves um nó gerencia, maior a quantidade de objetos de dados a ele mapeados. O número de objetos gerenciados por um nó é proporcional à quantidade desses identificadores virtuais a ele atribuídos (STOICA et al., 2001). A técnica de balanceamento aqui empregada consiste então em atribuir uma quantidade de identificadores virtuais para cada nó que seja proporcional à sua **capacidade**, fazendo com que nodos mais capacitados se responsabilizem por uma maior quantidade de objetos de dados.

Devido à diversidade de características dos recursos de rede, chegar ao equacionamento para um valor único que indique sua capacidade não é uma tarefa trivial. Ainda que fosse simples, existem paralelamente ao RMS outros programas ou até mesmo outros serviços da Grade que a utilizam. A rede pode ser, inclusive, um recurso compartilhado por diversas máquinas, muitas das quais o administrador pode não ter o controle. Sendo assim, apenas o recurso de disco será considerado para a capacidade de um nó, ficando o balanceamento do uso dos recursos de rede a cargo dos algoritmos usados para transferir os dados.

A capacidade de um nó é dada como sendo a quantidade de espaço de armazenamento por ele fornecido dividido por uma quantidade mínima exigida para que uma máquina faça parte do sistema:

$$Capacidade = \frac{armazenamento\_fornecido}{min\_armazenamento} \tag{4.1}$$

Essa unidade mínima deve ser determinada pela CA da Grade, de forma a refletir as características das máquinas que a compõem. Um exemplo típico para esse valor seria $40GB$, que é a metade da quantidade de armazenamento fornecida pelos discos rígidos mais comuns existentes atualmente no mercado. Se esse valor for pequeno demais em relação à quantidade de armazenamento fornecido pelas máquinas, elas terão que gerenciar um potencial grande número de identificadores, impactando significativamente no uso de memória e CPU. Por outro lado, se essse valor for grande demais, máquinas fornecendo pouco espaço poderão ter uma carga semelhante àquelas que fornecem muito espaço.

Caso o valor de $40GB$ fosse utilizado, por exemplo, um nó que fornecesse $160GB$ de espaço receberia 4 identificadores virtuais, enquanto que um dispositivo de armazenamento fornecendo $5TB$ receberia $\frac{5000}{40} = 125$ identificadores.

4.8.2 Controle de Recursos Locais

Em sistemas distribuídos, é comum os recursos estarem sob a tutela de diferentes domínios administrativos. Seus administradores se dispõem a compartilhá-los somente se for possível chegar a um acordo a respeito do seu uso. Ou seja, existem políticas de uso do recurso que precisam ser satisfeitas ou, do contrário, seu administrador recusará disponibilizá-lo junto à Grade. Um exemplo simples seria o de um usuário que deseja contribuir com sua estação de trabalho apenas no horário fora do expediente de trabalho. Ou ainda, um conjunto de máquinas de um laboratório que só podem ser disponibilizadas

em horários onde nenhum professor esteja lecionando uma aula. Ainda é possível que o administrador do recurso queira apenas diminuir a quantidade de dados gerenciados pelo sistema de forma a não impactar significativamente no seu ambiente de trabalho.

É através do serviço de Controle de Recursos Locais que JavaRMS fornece mecanismos para que os administradores dos recursos estabeleçam essas políticas de uso, ou mesmo manualmente controlem o impacto do sistema. Os recursos passíveis de controle são a rede e o disco, embora o consumo de memória esteja fortemente relacionado ao uso dos primeiros. Já o recurso CPU é pouco utilizado, mesmo sobre grande demanda por arquivos.

O mecanismo de controle de armazenamento se dá pelo gerenciamento dos identificadores virtuais atribuídos ao nó. Uma vez que a quantidade desses identificadores é proporcional ao espaço em disco fornecido, ao diminuir o número de identificadores ativos o administrador limita diretamente o uso desse recurso, pois um menor número de arquivos será gerenciado. O fato de esses identificadores serem desabilitados temporariamente simula a saída do nó virtual da rede P2P, fazendo com que as demais máquinas que compõem a rede deixem de rotear as mensagens para esse nó "indisponível". Dessa forma, diminui-se também o número de requisições por dados que serão repassadas ao recurso em questão, atenuando também o uso dos recursos de rede.

Entretanto, para controlar o uso da rede, a diminuição do número de nodos virtuais ativos não é suficiente. O simples fato de um único objeto de dados muito popular ser atribuído a um dos identificadores ativos já pode fazer com que um grande número de requisições sejam feitas ao nó. Ou ainda, mesmo que todos os dados fossem igualmente populares, uma única transferência de arquivo poderia consumir na totalidade o uso dos recursos de rede. Para lidar com esses problemas, JavaRMS utiliza mecanismos simples de limitação de banda.

O Controle de Recursos Locais utiliza as técnicas de nodos virtuais e de limitação de banda para fornecer o seguinte conjunto de operações:

- Consultar o número de identificadores virtuais ativos;
- Solicitar o uso explícito de um certo número de identificadores virtuais;
- Agendar a solicitação de um certo número de identificadores virtuais para uma data e horário específicos;
- Consultar o limite de banda ativo;
- Solicitar o limite de banda para um valor específico;
- Agendar a solicitação de um valor específico para a banda de rede a ser usada.

Embora os serviços listados sejam simples, é possível que políticas mais sofisticadas de controle de recursos sejam disponibilizadas futuramente. Um exemplo seria o de fornecer um controle adaptativo dos identificadores virtuais e banda de rede conforme um limiar de uso desses recursos. Ou seja, o usuário especifica que $50GB$ de disco poderão ser utilizados e o sistema diminui automaticamente o número de nodos virtuais ativos caso o uso de disco supere esse valor.

4.8.3 Serviço de Localização

O Serviço de Localização tem como única função localizar o grupo de nodos responsáveis por gerenciar um determinado objeto de dados. O mecanismo de localização por ele utilizado é determinístico pois, do contrário, um dado armazenado não poderia ser recuperado. Como os serviços P2P por si só já fornecem essa funcionalidade, o Serviço de Localização nada mais é do que uma simples *interface* para disponibilizar a operação de *lookup* como um serviço RMS. Ao receber a solicitação de localização para um objeto de dados, calcula-se a chave correspondente e envia-se uma requisição de *lookup* para o nodo virtual que possua o identificador mais próximo a ela. Iniciar a consulta nos nodos mais próximos otimiza a operação pois estes tendem a localizar o grupo de nodos responsáveis pela chave enviando um menor número de mensagens.

4.9 Serviços de Gerência de Usuários e Cotas

Os Serviços de Gerência de Usuários e Cotas surgem da necessidade de controlar o acesso aos dados por parte dos usuários, bem como de proteger os recursos de armazenamento contra fraudes. É também sua função fornecer meios para o cadastro de usuários, já que a gerência de arquivos precisa de um serviço igualmente escalável para obter informações dos usuários. Caso essa proteção contra fraudes não fosse fornecida, os usuários poderiam usar indiscriminadamente os recursos de armazenamento, fazendo com que se esgotassem rapidamente. Ou ainda, o sistema estaria sujeito a ataques onde todos os recursos seriam ocupados para evitar que outros usuários inserissem seus arquivos. Sem o controle de acesso, seria também possível que um usuário removesse os arquivos de outro sem ter sido autorizado.

A Fig. 4.7 mostra como o componente está organizado e sua relação com demais serviços da arquitetura. Os Serviços de Comunicação são utilizados para que os serviços de usuários e cotas sejam apresentados como serviços RMS. Já o Serviço de Localização, fornecido pelo Gerente de Recursos, se faz necessário para determinar quais nodos serão responsáveis por manter informações de cada usuário. O Serviço de Gerência de Usuários e Cotas em si é subdividido em quatro categorias de serviços: o Serviço de Criação e Registro de Usuários, que determina que usuários da Grade poderão usar o sistema e que cota cada um receberá; o Serviço de Informações de Usuários, que fornece dados a respeito dos usuários, tais como a cota ainda disponível; o Serviço de Contabilidade, que recebe informações sobre os arquivos inseridos/removidos do sistema para contabilizar o quanto cada usuário já utilizou da sua cota; e o Serviço de Manutenção de Usuários, que garante a disponibilidade de informações de cada usuário gerenciado na ocorrência de falhas dos nodos. Existe ainda, como entidades externas, os próprios Usuários e a Autoridade Certificadora.

A CA é a entidade da Grade com a função de dizer quais usuários tem permissão de usar o sistema e qual a cota de armazenamento a eles concedida. A atribuição de cotas a usuários deve ser feita segundo alguma política de uso da Grade. Por exemplo, a CA poderia determinar que cada usuário tem o direito de usar uma quantidade de disco proporcional àquela por ele fornecida ao sistema.

4.9.1 Serviço de Criação e Registro de Usuários

Independentemente de os serviços do JavaRMS serem usados ou não, cada usuário da Grade recebe um documento digital, assinado pela CA, que faz a ligação entre o seu nome

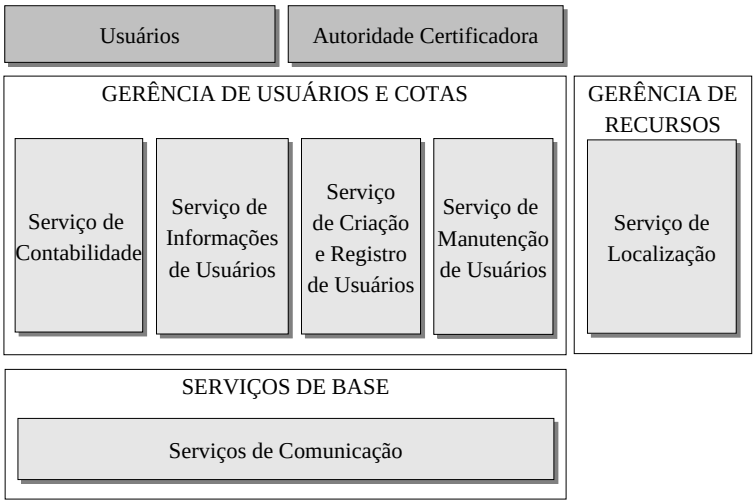

Figura 4.7: Arquitetura dos Serviços de Gerência de Usuários e Controle de Cotas.

e uma chave pública. Esse documento (um certificado X509) é utilizado pelo usuário para se autenticar frente aos serviços fornecidos pela Grade. O usuário também utiliza a chave privada correspondente à chave pública do certificado como parte desse procedimento de autenticação. Dependendo do serviço acessado, esses objetos podem ainda ser utilizados para garantir a integridade e a confidencialidade das informações trocadas.

Ambos os documentos (certificado e chave privada) são também utilizados pelo JavaRMS para garantir a segurança de uso dos serviços prestados e para viabilizar o controle de acesso. Juntamente com eles, cada usuário recebe um documento digital, também assinado pela CA, que liga o nome do usuário da Grade a um valor de cota. Esse documento também pode conter informações adicionais usadas para o controle de acesso, tais como os grupos da Grade aos quais o usuário pertence. No JavaRMS, tal documento chama-se **pedido de cota**. Considera-se que um usuário existe apenas quando um pedido de cota for a ele emitido pela autoridade da Grade.

Para que um usuário esteja apto a usar os serviços RMS, ele encaminha seu certificado X509 e o seu pedido de cota ao Serviço de Criação e Registro de Usuários. Ao recebê-los, esse serviço os envia para o grupo de nodos responsáveis por gerenciar o usuário. Tal grupo é descoberto usando-se o Serviço de Localização, onde o objeto de dados procurado é a identificação do usuário na Grade (seu *Distinguished Name* – DN). Ao receber esses documentos, os nodos pertencentes ao grupo verificam a assinatura dos mesmos para validá-los e passam a contabilizar os arquivos do referido usuário para o controle de cotas.

O armazenamento do certificado X509 se faz necessário para poder verificar as assinaturas de outros documentos digitais emitidos pelo usuário. Mesmo que existisse um repositório na Grade com esses documentos, seria necessário que a sua localização e descoberta fossem também escaláveis de acordo com a quantidade de nodos presentes no sistema. Ambos os documentos, por serem assinados pela mesma entidade, poderiam ainda ser concatenados num só. Entretanto, eles não foram assim concebidos para não quebrar com o padrão X509. Caso o fosse, os usuários não poderiam utilizar seus certificados junto a outros serviços da Grade.

A exigência de um grupo de nodos para manter as informações de cada usuário se dá por motivos de desempenho, de segurança e também para suportar indisponibilidades

temporárias dos nodos. Como a infra-estrutura P2P é utilizada pelo Serviço de Localização para encontrar os nodos que compõem o grupo, cada usuário é mapeado para um grupo de nodos distinto e que está espalhado pela Grade. Distribui-se, assim, a carga de gerenciamento de usuários entre todas as máquinas do sistema, obtendo-se escalabilidade.

Uma vez que as máquinas desses grupos estão espalhadas pela Grade e estão potencialmente sobre a vigência de domínios administrativos diferentes, é muito improvável que um usuário consiga ter controle sobre elas para emular uma cota ilimitada. De forma análoga, um nó do grupo que tenha tido sua segurança comprometida não é capaz de forjar informações sobre um usuário pois estas dependem da maioridade de votos do grupo.

Devido à volatilidade dos nodos que compõem a Grade, caso o usuário fosse gerenciado por um único nó, o fato de ele se tornar indisponível tornaria também o usuário inativo. Ao utilizar um grupo de nodos, consegue-se estabelecer protocolos que garantam o gerenciamento do usuário mesmo frente a essa situação. Para esse propósito, os integrantes de cada grupo se responsabilizam por replicar os documentos em novas máquinas caso algum nó do próprio grupo se tornar indisponível. Para que um usuário seja indisponibilizado nessa proposta, seria necessário que todas as máquinas do grupo falhassem simultaneamente, sem que houvesse tempo para que novas réplicas fossem feitas. Como não há relação administrativa entre os nodos do grupo, é muito improvável que exista também uma correlação de falhas entre eles. Para que um usuário seja de fato removido, a CA deve emitir um pedido de cota para ele especificanto um valor de cota igual a zero.

4.9.2 Serviço de Contabilidade

Sempre que um usuário insere ou remove um arquivo do sistema, o grupo de nodos por ele responsável deve atualizar seus registros para refletir o espaço de armazenamento total utilizado pelo usuário. Essa atualização se dá sobre a responsabilidade do Serviço de Contabilidade, que tem o objetivo de divulgar os arquivos do usuário entre os nodos que compõem o grupo. Consegue-se, assim, fazer com que cheguem a um consenso sobre a quantidade de espaço utilizado.

Cada arquivo do sistema é referenciado por um documento digital, assinado pelo usuário, que indica que o usuário especificou uma operação de inserção ou remoção. Dentre outras informações presentes nesse documento, consta o tamanho do arquivo e o fator de replicação utilizado. Como parte do protocolo de operações sobre arquivos, esse documento chega até um ou mais dos nodos do grupo. Cada máquina que o recebe atualiza localmente o espaço usado pelo usuário somando $tamanho\_arquivo \times fator\_replicacao$, caso o documento indique inserção, ou subtraindo esse mesmo valor caso indique remoção. Na seqüência, uma referência ao documento é armazenada para que o nó saiba que ele já foi processado. Por fim, a máquina o encaminha para seus vizinhos sistematicamente, de forma que todos os nodos do grupo obtenham uma cópia do documento e o processem.

4.9.3 Serviço de Informações de Usuários

O Serviço de Informações de Usuários tem como simples função fornecer uma *interface* RMS para que outros componentes do sistema possam obter dados a respeito dos usuários. As informações disponibilizadas através dessa *interface* são:

- Data de criação do usuário, nome de Grade da CA que o cadastrou, a cota a ele atribuída e demais informações contidas no pedido de cota;
- O espaço utilizado pelo usuário até o momento.

Toda vez que um nó se defronta com requisições de armazenamento do usuário, por exemplo, consulta-se o serviço de Informações de Usuários para saber se a cota desse usuário ainda não foi estourada.

4.10 Gerência de Arquivos

O componente de Gerência de Arquivos constitui um sistema de arquivamento escalável, robusto e seguro para dados imutáveis ou com poucas exigências de escrita, tais como aqueles encontrados na Grade da Física de Altas Energias. O sistema garante a durabilidade dos arquivos mesmo na ocorrência de falhas dos recursos de armazenamento, garante que somente usuários autorizados poderão acessá-los ou modificá-los, e realiza auditoria para controlar o espaço utilizado. Os protocolos para a realização dessas operações são seguros, garantindo assim seu correto funcionamento mesmo na presença de entidades mal intencionadas. O gerenciamento e o armazenamento dos dados é distribuído entre as máquinas que compõem a Grade segundo um modelo P2P e aliado a técnicas de balanceamento de carga, permitindo atingir grande escala no número de nodos gerenciados.

A arquitetura da Gerência de Arquivos é composta por quatro unidades menores (Fig. 4.8): os Serviços de Arquivamento, que fornecem serviços RMS para inserção, remoção, localização e reconstituição de arquivos; o Serviço de Criação de Arquivos, que auxilia os usuários na escolha de parâmetros para a criação dos arquivos de acordo com suas expectativas de uso; o Serviço de Manutenção, que garante que os arquivos poderão ser obtidos mesmo na ocorrência de falhas dos recursos de armazenamento; e os Serviços de Controle de Popularidade, que monitoram a demanda pelos arquivos e empregam o uso de *caching* para lidar com as diferenças de popularidade dos dados.

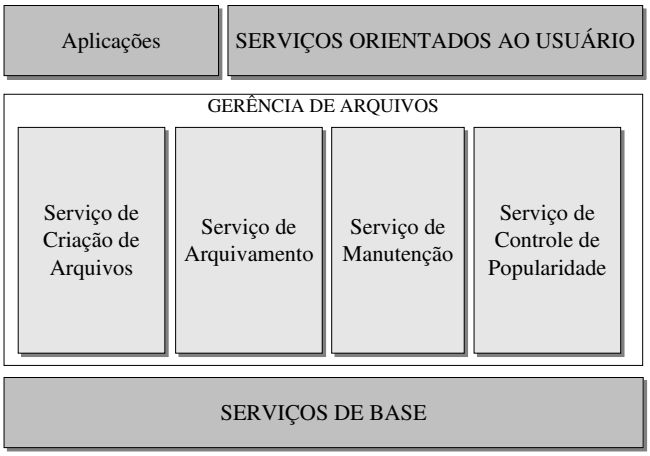

Figura 4.8: Arquitetura dos Serviços de Gerência de Arquivos.

4.10.1 Fragmentação

Mesmo que o sistema distribuisse equalitariamente os arquivos entre os nodos que compõem o sistema, o fato de os arquivos poderem assumir tamanhos variados geraria diferenças quanto ao uso de recursos de armazenamento com configurações semelhantes. No caso da Grade do HEP, onde o tamanho dos arquivos pode variar de algumas dezenas de *Megabytes* até dezenas de *Gigabytes*, grandes discrepâncias quanto ao uso de disco nas máquinas seriam evidentes.

Outro problema que surge quando o tamanho dos arquivos assume essas características é que recursos menos capacitados poderiam não conseguir armazenar um arquivo por inteiro. Ainda que fosse possível, um único arquivo poderia monopolizar praticamente todo o espaço de armazenamento nesse recurso. Por exemplo, um arquivo de $70GB$ poderia ser alocado a um *desktop* que fornece apenas $80GB$ de disco. Nesse caso, inclusive, existiria um custo muito alto associado à transferência do arquivo para o nó que, para piorar o problema, poderia ter uma conectividade de rede ruim.

JavaRMS trata desses problemas através do uso de **fragmentação** de dados (TANENBAUM, 2001; DABEK et al., 2001). Essa técnica consiste em dividir os arquivos em unidades menores, chamadas de fragmentos, pedaços ou blocos, de forma a gerenciá-los independentemente como se fossem arquivos distintos. Possibilita-se então o uso de nodos menos capacitados sem exaurir com seus recursos de rede e disco. Consegue-se também balancear a carga de armazenamento, fazendo com que nodos que armazenem uma mesma quantidade de arquivos não apresentem grandes diferenças quanto ao espaço em disco ocupado.

Um terceiro motivo importante para usar fragmentação é que ela potencializa o número de máquinas envolvidas no gerenciamento de um arquivo. Ou seja, apesar de ter o mesmo custo de armazenamento (a soma do tamanho dos blocos é igual ao tamanho total do arquivo quando não fragmentado), consegue-se que uma maior quantidade de nodos possam responder pelo arquivo. Em resumo, as vantagens de aumentar o número de máquinas envolvidas no gerenciamento de um arquivo são:

- Por distribuir a carga de gerenciamento entre um conjunto de nodos, consegue-se atender a um maior número de requisições pelo arquivo num mesmo intervalo de tempo;

- Um único cliente pode aumentar o paralelismo das transferências fazendo o *download* simultâneo de cada trecho do arquivo de uma máquina diferente;

- Apesar de falhas e ataques a máquinas individuais, o arquivo não se torna totalmente inacessível, permitindo que os clientes tenham uma maior probabilidade de conseguir transferir algum dos seus trechos.

Entretanto, a probabilidade de o arquivo poder ser reconstituído como um todo é menor, pois precisa-se que um maior número de máquinas esteja disponível. Existe também um custo associado à localização e à transferência de cada fragmento do arquivo. O custo de localização seria o mesmo daquele para localizar um arquivo inteiro caso a fragmentação não estivesse sendo usada. Para as transferências, há um custo inicial dos protocolos de movimentação de dados até que o primeiro *byte* do fragmento seja recebido. Como existem n fragmentos, esses custos se darão n vezes. Entretanto, esses custos podem ser mascarados por um ganho no desempenho obtido por transferir em paralelo vários trechos do arquivo provindos de fontes (máquinas) diferentes. Para que isso seja possível, no entanto, é preciso que o tamanho e número de pedaços seja escolhido adequadamente.

Existem diversas estratégias para dividir um arquivo em pedaços. Pode-se usar um tamanho de bloco fixo para todos os arquivos (como acontece em sistemas de arquivos convencionais), um tamanho de bloco fixo por arquivo, tamanhos de blocos diferentes para um mesmo arquivo, um número fixo de blocos para todos os arquivos, um número fixo de blocos por arquivo, entre outros. De fato, muitos fatores influenciam na escolha de uma melhor estratégia de fragmentação, que por sua vez depende se o objetivo é priorizar aspectos relativos ao desempenho ou ao tratamento de erros. Mesmo quando o objetivo é

bem definido (no caso do HEP, usar a alta disponibilidade de dados para fins de desempenho), a dinamicidade da Grade exige uma estratégia de fragmentação adaptativa. Por exemplo, em um dado instante, usar um fragmento de tamanho de $5Mb$ em uma transferência quando a rede apresenta congestionamentos pode ser vantajoso, porém quando a mesma não encontra-se congestionada, o sobre custo de iniciar a trasferência pode não valer a pena. Nesse caso, seria mais vantajoso transferir um único fragmento de $10Mb$ do que dois de $5Mb$.

JavaRMS não impõe qualquer estratégia de fragmentação, apenas fornece mecanismos para que o usuário escolha a estratégia que lhe pareca mais favorável ao registrar um arquivo. Assim, permite-se que os arquivos sejam fragmentados com tamanho e número de blocos variável. Ou seja, os blocos de um mesmo arquivo podem possuir tamanhos variáveis, e diferentes arquivos podem possuir diferentes números de blocos. Além da flexibilidade na escolha da estratégia de fragmentação, esse esquema apresenta uma grande vantagem: a possibilidade de os arquivos registrados serem modificados. Quando na ocorrência de uma modificação, apenas os fragmentos que foram alterados precisam ser reinseridos no sistema. Embora os arquivos do HEP sejam imutáveis, eles passam por um período inicial onde modificações sucessivas são feitas. Na prática, esses arquivos acabam sendo reinseridos no sistema com um nome diferente. Já no JavaRMS, a característica de mutabilidade para os dados permitiria que o mesmo arquivo fosse mantido e apenas os fragmentos alterados fossem reinseridos. Essa característica permite também que o sistema seja usado em grades de dados de outras outras áreas da Ciência onde a modificação de arquivos se faz necessária. Para evitar problemas com atualizações concorrentes, permite-se que apenas o usuário que criou o arquivo faça as modificações. Caso ele faça modificações concorrentemente, valerá aquela com a data mais atual.

4.10.2 Serviços de Arquivamento

Constituem os Serviços de Arquivamento as seguintes operações:

- *Inserção de Arquivo*: torna um arquivo gerenciável pelo sistema;
- *Localização de Arquivo*: permite a identificação das máquinas que contêm cópias do arquivo ou de seus fragmentos;
- *Reconstituição (ou Recuperação – Retrieve) de Arquivo*: remonta o arquivo a partir das cópias de suas partes que estão espalhadas pela Grade e o entrega ao usuário;
- *Remoção de Arquivo*: apaga as cópias do arquivo do sistema e o torna ingerenciável.

Inserção de Arquivo

Para colocar um arquivo sob o controle do sistema, o usuário deve emitir um documento, chamado **pedido de arquivo**, através do qual o arquivo é descrito e utilizado para posterior reconstituição. Esse documento contém uma lista de pedaços (fragmentos) que compõem o arquivo, um fator de replicação R, usado para dizer quantas réplicas do arquivo devem ser geradas, e um conjunto de atributos básicos usados para identifição e posterior localização do arquivo no sistema. Esses atributos são:

- Nome do arquivo na Grade (*Logical File Name* – LFN);
- Identificação do usuário dono do arquivo;

- Fator de replicação;
- Lista de fragmentos;
- Data de criação (data de emissão do pedido);
- Endereço onde a cópia primária do arquivo pode ser obtida (*Physical File Name* – PFN);
- Operação: 0 indica inserção, 1 indica remoção.

O pedido de arquivo é então assinado digitalmente pelo usuário dono a fim de garantir sua autenticidade e para garantir a integridade do seu conteúdo. Uma vez com ele em mãos, o usuário o encaminha para o serviço de arquivamento para que o arquivo seja inserido no sistema. Ao receber esse pedido, o sistema primeiramente localiza um grupo de nodos que serão responsáveis pelo seu gerenciamento. Para tal, solicita-se aos serviços de Localização para que procurem pela chave que resulta do *hashing* do LFN concatenado ao nome do usuário que o criou. Depois de descobrir os nodos que compõem o grupo, realiza-se um *multicast* do pedido para os mesmos. Ao recebê-lo, as máquinas desse grupo verificam se já não existe um arquivo com o mesmo nome e usuário. Caso já exista, a data de criação é comparada ao do pedido já existente e, se for posterior, a operação prossegue. Questiona-se então a cota do usuário dono ao grupo de nodos responsáveis por contabilizar seus arquivos. Caso ele não tenha cota suficiente, os nodos se recusam a gerenciar o arquivo e a operação é cancelada. Caso contrário, um ou mais desses nodos informam os nodos responsáveis pelo usuário dono que o arquivo deve ser contabilizado, enviando a eles o pedido de arquivo. Nesse momento, o pedido de arquivo já fora aceito pelo grupo de nodos responsável pelo seu gerenciamento e o espaço ocupado pelo arquivo já fora contabilizado.

Inicia-se então a etapa de transferência, que consiste em distribuir cópias dos fragmentos entre máquinas espalhadas pela Grade. Para cada fragmento da lista contida no pedido de arquivo, o serviço de Arquivamento solicita uma operação de localização, utilizando como chave o *hashing* do conteúdo do fragmento (retirados do próprio pedido de arquivo). Como resultado dessas operações de localização, tem-se uma lista que identifica as máquinas que devem armazenar cada um dos fragmentos. Realiza-se então um *multicast* do pedido de arquivo para as R (onde R é o fator de replicação) primeiras máquinas de cada grupo. Ao recebê-lo, as máquinas verificam se realmente pertencem ao grupo dos R primeiros nodos responsáveis pelo armazenamento do fragmento, consultam a cota do usuário dono e depois iniciam a transferência do fragmento, usando a URL para a cópia primária especificada no pedido (o PFN). Assim, após as transferências existirão R cópias de cada fragmento e a operação é dada como concluída.

Existe uma redundância em alguns dos procedimentos, tais como verificação da cota do usuário por parte de cada grupo de máquinas. Essa redundância se dá por motivos de segurança. Um nó dominado por um usuário poderia, por exemplo, ser modificado para apenas realizar a segunda etapa do procedimento de inserção, fazendo com que as máquinas armazenassem os fragmentos mesmo sem o usuário ter cota suficiente. Ao requisitar as mesmas informações a mais de um nó, consegue-se atribuir segurança às operações pois a vizinhança na rede P2P não possui relação com os domínios administrativos onde os nodos se encontram. Tal redundância poderia ser retirada a fim de otimizar essas operações, porém as verificações de segurança a ela associadas deveriam ser verificadas em protocolos de manutenção posteriores. Algumas outras operações de segurança são ainda

realizadas durante o procedimento descrito, tal como a verificação da assinatura do pedido por parte de cada nó que o recebe, porém esses testes foram suprimidos com o objetivo de simplificar a descrição.

Remoção de Arquivo

O processo de remoção de um arquivo no JavaRMS é semelhante ao processo de inserção. Emite-se um **pedido de arquivo**, porém ao campo `operação` é atribuído o valor 1, que identifica que o arquivo deve ser removido. O pedido de arquivo deve conter uma data mais recente do que o pedido de arquivo usado para inserí-lo, pois do contrário a operação de remoção será desconsiderada. Com o pedido em mãos, o serviço de Arquivamento o repassa aos nodos responsáveis pelo seu gerenciamento, aos nodos responsáveis por cada pedaço e aos nodos responsáveis por fazer a contabilização do espaço utilizado pelo usuário dono do arquivo. Assim como na inserção, cada um desses conjuntos de nodos é localizado através dos serviços de Localização do módulo de Gerência de Recursos. Ao receber um pedido de remoção de arquivo, o nó responsável por mantê-lo simplesmente substitui o pedido que representava a inserção do arquivo. Dessa forma, caso o arquivo seja posteriormente a ele solicitado, retorna-se o documento que garante que o arquivo fora removido. Esse esquema também garante que um nó, ao reingressar no sistema, possa ser informado corretamente a respeito dos arquivos anteriormente por ele armazenados e que foram removidos enquanto estava fora. Cada nó responsável por armazenar uma cópia de um pedaço apaga o mesmo ao receber o pedido de remoção, de forma a liberar espaço. Por fim, todos os nodos que processam o pedido de remoção antes verificam a assinatura do mesmo para ter certeza de que o pedido não foi forjado, ou seja, verificam a autenticidade e integridade do pedido para evitar que um invasor remova do sistema os arquivos de outros usuários.

Reconstituição de Arquivo

Reconstituir um arquivo significa montar o arquivo a partir dos pedaços espalhados pelos nodos que compõem o sistema. Primeiramente o cliente RMS localiza os nodos responsáveis pelo gerenciamento do arquivo e obtém uma cópia do pedido de arquivo correspondente. Com o pedido em mãos, requisita-se uma operação de localização para cada pedaço descrito na lista de fragmentos. Uma vez com a lista de R fontes de cada pedaço em mãos, o usuário as repassa ao Serviço de Transferências, informando um arquivo local praonde ele será gravado.

Inicia-se então a transferência de cada pedaço, onde vários deles podem ser transferidos simultaneamente. Todas as R fontes de um pedaço são usadas para transferí-lo segundo o algoritmo de transferência adotado pelo JavaRMS. Assim, consegue-se um alto nível de paralelismo nas transferências, otimizando o processo de reconstituição do arquivo. Como existe a informação de *offset* associada a cada pedaço no pedido de arquivo, não há a necessidade de executar outros procedimentos para reconstituí-lo após o término das transferências. Ou seja, conforme vão sendo transferidos, os dados vão sendo gravados nos trechos correspondentes em um arquivo local. Ao término da transferência de cada pedaço, calcula-se o *hash* do seu conteúdo para verificar se bate com o descrito no pedido de arquivo. Essa verificação de integridade precisa ser realizada por motivos de segurança, pois do contrário um nó comprometido por um invasor poderia retornar um conteúdo que não corresponde ao do arquivo solicitado.

Modificação de Arquivo

Modificar um arquivo significa reinserí-lo no sistema. O procedimento é o mesmo da inserção, porém os pedaços do arquivo antigo que apresentaram modificações precisam ser removidos para liberar espaço. Como no JavaRMS a chave usada para localizar um pedaço é o *hash* do seu conteúdo, ao efetuar uma modificação em um ou mais pedaços, alguns nodos deixarão de ser responsáveis pelos pedaços onde ocorreram modificações, assim como outros nodos passarão a ser responsáveis pelos novos pedaços e precisarão transferí-los. Quando um nó responsável pelo pedaço antigo recebe uma cópia do novo pedido de arquivo, ele identifica que não é mais responsável pelo pedaço e o descarta. Já os nodos responsáveis pelo novo pedaço percebem o ocorrido e fazem a transferência a partir do PFN, que é a URL para a cópia primária especificada no documento. Para os demais pedaços onde não ocorreram modificações os nodos simplesmente substituem o pedido antigo pelo mais recente. Os nodos responsáveis pelas informações do dono do arquivo também farão as contabilizações de cota necessárias.

Caso um novo valor para o fator de replicação R seja especificado no novo documento, o grupo de nodos é informado para refletir a nova configuração. Dessa forma, novas cópias devem ser geradas caso o novo valor de R seja maior. Como sempre os R nodos são comunicados como parte do protocolo de inserção, os novos nodos responsáveis copiam o arquivo como se ele estivesse sendo inserido pela primeira vez. No entanto, caso o novo valor de R seja menor que o antigo, alguns nodos deixam de ser um dos R responsáveis e precisam ser comunicados. Para evitar que um procedimento adicional tenha que ser efetuado para esse caso específico, deixa-se que, provisoriamente, os nodos armazenem cópias relativas ao valor de R antigo. Como parte dos procedimentos de manutenção devido à entrada/saída de nodos, em algum momento o novo pedido de arquivo chegará a esses nodos e as cópias excedentes de fragmentos serão removidas.

4.10.3 Serviço de Criação de Arquivos

O Serviço de Criação de Arquivos tem o objetivo de auxiliar os usuários na escolha dos parâmetros que constituem um pedido de arquivo. Eles são:

- Fator de replicação (R);
- Número de fragmentos (n).

A escolha desses parâmetros é importante pois tem impacto significativo na disponibilidade do arquivo, no custo de armazenamento, e no desempenho das funções que operam sobre o arquivo. A seguir, apresenta-se uma discussão quanto à determinação desses valores.

Disponibilidade

Considera-se a **disponibilidade** de um arquivo como sendo a probabilidade de ele poder ser reconstituido, ou seja, a probabilidade de que pelo menos uma cópia de cada um dos seus fragmentos estar disponível para *download*. A disponibilidade dos dados está relacionada aos fatores:

- *Disponibilidade dos nodos*: é a probabilidade de um nó estar presente, permitindo acesso aos arquivos (fragmentos) por ele gerenciados. Devido à heterogeneidade dos nodos no ambiente de Grade, existe uma grande diferença na disponibilidade de cada nó. Por exemplo, um recurso de armazenamento em um grande centro

computacional estaria praticamente 100% do tempo disponível, enquanto que um *desktop* de um físico poderia ter disponibilidade baixa, por volta de 20%. Assim, dados gerenciados por nodos pouco disponíveis apresentam uma menor disponibilidade;

- *Fragmentação*: ao fragmentar um arquivo, é necessário contactar um maior número de nodos para poder reconstituí-lo, tornando-se susceptível a um maior número de falhas. O simples fato de um único fragmento não estar disponível já inviabiliza a construção do arquivo como um todo;

- *Replicação*: o fato de criar cópias de um mesmo fragmento em diferentes nodos aumenta a probabilidade de pelo menos um deles estar disponível para *download*.

Adicionalmente, as políticas locais de uso dos nodos virtuais por parte dos administradores dos recursos faz com que os arquivos gerenciados se tornem indisponíveis mesmo quando não ocorrem falhas reais. No entanto, para contornar esse problema basta considerar a disponibilidade individual de cada nó virtual ao invés da disponibilidade real das máquinas. Dessa forma, equaciona-se a disponibilidade de um arquivo (λ) em função da indisponibilidade do nó virtual (γ), do número de fragmentos (n) e do fator de replicação (R):

$$\lambda = \prod_{i=1}^{n}(1 - \prod_{j=1}^{R} \gamma_{ij}) \tag{4.2}$$

onde:

- γ_{ij}: é a probabilidade de a j-ésima réplica do i-ésimo fragmento **não** estar disponível;

O termo $\prod_{j=1}^{R} \gamma_{ij}$ representa a probabilidade de nenhuma das R réplicas de um fragmento i estar acessível para *download*, ou seja, é a **indisponibilidade** do fragmento. Logo, $(1 - \prod_{j=1}^{R} \gamma_{ij})$ dá a **disponibilidade** para o fragmento i. Como um arquivo só está disponível se todos os seus fragmentos forem acessíveis, a probabilidade de ele poder ser reconstituído (λ) é o produto entre as disponibilidades dos fragmentos, resultando na Eq. 4.2.

Número de fontes

Conceitua-se o **número de fontes** para um arquivo como sendo a quantidade de nodos envolvidos no seu gerenciamento. Como existem R réplicas de cada fragmento, calcula-se o número de fontes F para um arquivo:

$$F = nR \tag{4.3}$$

O número de fontes reflete o **desempenho** das operações realizadas sobre um arquivo. Quanto maior for o número de fontes disponíveis, maior é o paralelismo possível nas transferências. As requisições pelo arquivo são também distribuídas entre os F nodos envolvidos, de forma que esse parâmetro reflete a demanda que o sistema pode suportar. Um arquivo muito popular precisa de um grande número de fontes, mesmo que um único usuário não vá utilizar todas elas durante uma operação de reconstituição.

Devido à entrada/saída de nodos, a equação 4.3 pode não refletir na prática o número de fontes para um arquivo existentes em um dado momento. No momento que um nó

virtual sai da rede P2P, leva-se um tempo até que uma nova réplica possa ser criada. Entretanto, para simplificar a análise a seguir considerar-se-á o número de fontes como sendo constante, como se as máquinas pudessem gerar rapidamente novas cópias de fragmentos quando um nó se torna indisponível.

Custo de Armazenamento

O custo de armazenamento de um arquivo está unicamente relacionado ao fator de replicação (R) escolhido. Esse custo é indiferente quanto à disponibilidade dos nodos envolvidos no gerenciamento e reflete o espaço total que o arquivo ocupará em disco enquanto estiver sendo gerenciado. O custo de armazenamento (A) é dado por:

$$A = sR \tag{4.4}$$

Onde s é o tamanho do arquivo. De fato, o custo de armazenamento é a **moeda** usada pelos usuários no JavaRMS, limitando-se no máximo ao valor da cota atribuída ao usuário. É como se o usuário estivesse pagando pelo serviço. Naturalmente, deseja-se minimizar os custos de armazenamento.

Discussão

Em uma análise direta da eq.4.2, percebe-se que ao aumentar o número de fragmentos, a disponibilidade do arquivo diminui. Por outro lado, ao aumentar o número de réplicas, aumenta-se sua disponibilidade. Na eq. 4.3, consegue-se aumentar o número de fontes tanto através do número de fragmentos quanto pelo fator de replicação. Como é desejável sempre minimizar o custo de armazenamento, deve-se escolher valores para n e R de forma a atender um bom compromisso entre disponibilidade (λ), desempenho (F) e custo de armazenamento (A).

Se o objetivo do usuário é economizar espaço de armazenamento, ele deve escolher um fator de replicação baixo. Ao fazê-lo, ele deve decidir se o segundo critério mais importante para seus propósitos é garantir uma maior disponibilidade dos dados ou um maior desempenho das operações. Caso seja a disponibilidade, ele terá que usar um pequeno número de fragmentos n. Caso opte pelo maior desempenho, deverá utilizar um valor maior para n a fim de aumentar o número de fontes. Neste caso, porém, a probabilidade de o arquivo poder ser reconstituído diminui.

Por outro lado, se o desempenho for o aspecto principal, o usuário deve primeiro escolher um valor de F que vá lhe permitir maiores taxas de transferência durante as operações de reconstituição do arquivo. Num segundo momento, ele deve procurar valores de n e R que satisfaçam o F e maximizem a disponibilidade. Se ele usar um valor de R grande demais para atingir a disponibilidade desejada, terá de arcar com um maior custo de armazenamento.

Já quando a prioridade do usuário é a disponibilidade, ele deve primeiro especificar o valor para λ (por exemplo, $\lambda = 0,999$ para $99,9\%$ de disponibilidade). Posteriormente, deve escolher o par n e R que satisfaçam a λ e maximizem o número de fontes gastando o menos possível de armazenamento.

Percebe-se que o usuário deve, portanto, escolher os valores de n e R de forma a obter um bom compromisso entre *disponibilidade*$\times$*desempenho*$\times$*custo de armazenamento*. O Serviço de Criação de Arquivos tem justamente a função de auxiliar o usuário nessa escolha.

Análise

Se na equação da disponibilidade (Eq. 4.2) for considerado um mesmo valor de indisponibilidade (γ) para todos os nodos, obtém-se:

$$\lambda = (1 - \gamma^R)^n \tag{4.5}$$

Isolando-a para a variável n, tem-se:

$$n = \frac{log(\lambda)}{log(1 - \gamma^R)} \tag{4.6}$$

Da equação do número de fontes (Eq. 4.3), observa-se que $n = \frac{F}{R}$. Substituindo n na Eq. 4.6 e isolando F, obtém-se:

$$F = R\frac{log(\lambda)}{log(1 - \gamma^R)} \tag{4.7}$$

Observa-se que essa equação relaciona F, R e λ de acordo com a indisponibilidade γ dos nodos que compõem o sistema. A Fig. 4.9 mostra o resultado dessa equação, para um valor de $\gamma = 0.2$. O eixo do fator de replicação R reflete o custo de armazenamento, enquanto que o eixo do número de fontes F reflete o desempenho.

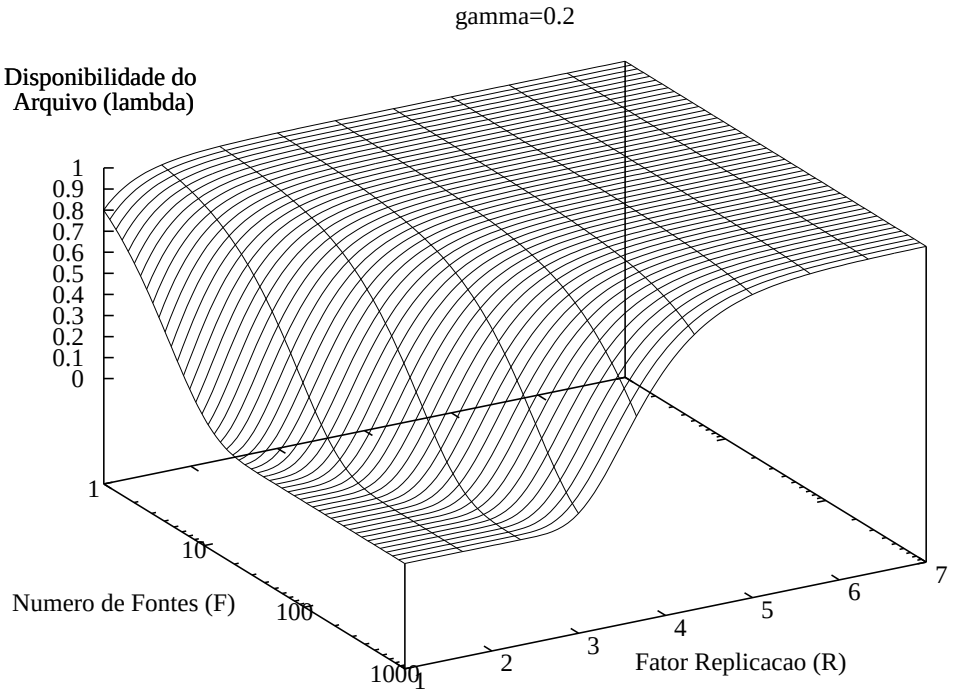

Figura 4.9: Disponibilidade obtida para os arquivos resultante do emprego das técnicas de fragmentação e replicação.

Para facilitar a visualização, a Fig 4.10 mostra o resultado da mesma equação, porém para um subconjunto de valores para a disponibilidade λ. Como o eixo do número de fontes está em escala logarítmica, percebe-se que, para um mesmo valor de λ, o número de fontes possíveis cresce exponencialmente com o aumento do custo de armazenamento (indicado pelo fator replicação R).

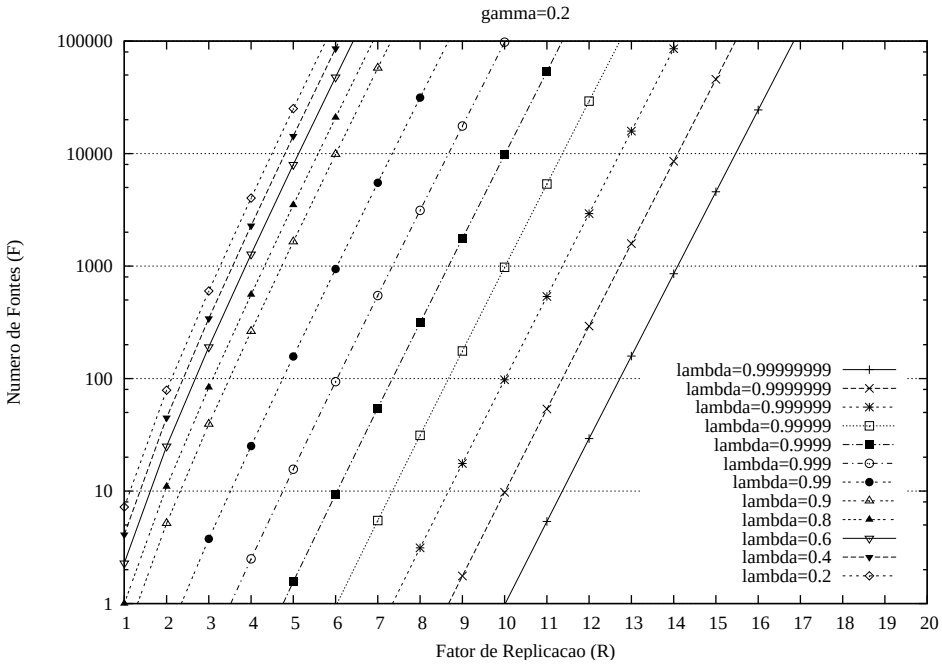

Figura 4.10: Combinações possíveis de fragmentos e de réplicas para valores fixos de disponibilidade para um arquivo.

O quão exponencial se dará esse crescimento depende do valor da indisponibilidade dos nodos γ. A Fig. 4.11 mostra qual é o impacto da indisponibilidade dos nodos virtuais para uma disponibilidade de 99% ($\lambda = 0.99$). Percebe-se que a indisponibilidade tem grande influência no custo de armazenamento para que uma mesma disponibilidade do arquivo seja mantida. Por exemplo, para usar 1000 fontes com $\gamma = 20\%$, cerca de 6 réplicas são necessárias para cada fragmento para manter os 99% de disponibilidade para o arquivo. No entanto, com $\gamma = 50\%$, são necessárias 13 réplicas para obter essa mesma disponibilidade.

Embora a equação da disponibilidade tenha sido simplificada, é possível concluir que, caso a indisponibilidade dos nodos variasse entre 20 e 50%, escolher um valor de $R = 13$ asseguraria que a disponibilidade do arquivo seria de, no mínimo, 99% quando $F = 1000$. Por outro lado, se o fator de replicação escolhido fosse $R = 6$, a disponibilidade do arquivo seria de, no máximo, 99%.

Com isso, é possível fornecer um primeiro serviço de auxílio na criação de arquivos: dado um valor de disponibilidade (λ) e o número de fontes (F) desejado, retorna-se os valores de R que darão, respectivamente, no mínimo, no máximo e na média a disponibilidade λ desejada. Esse serviço recai sobre as informações de monitoramento dos nodos virtuais para saber qual a faixa de valores para a indisponibilidade que os nodos virtuais da Grade assumem. É importante frizar também que assume-se uma distribuição de indisponibilidade (γ) para os nodos virtuais que é homogênea na faixa de valores monitorado. Na prática, essa distribuição pode não ter essa forma. Nesse caso, o serviço poderia ser alterado para calcular intervalos de confiança para os valores de R retornados. Ou seja, calcularia-se uma faixa de valores de R onde existe uma grande probabilidade de a disponibilidade exigida ser a obtida.

Caso o custo de armazenamento seja mais importante para o usuário do que o desempenho das transferências, pode-se também utilizar a Eq. 4.7 para fornecer um segundo serviço: dados λ e R, calcula-se os valores F_{min}, F_{max} e F_{medio} para se obter, respecti-

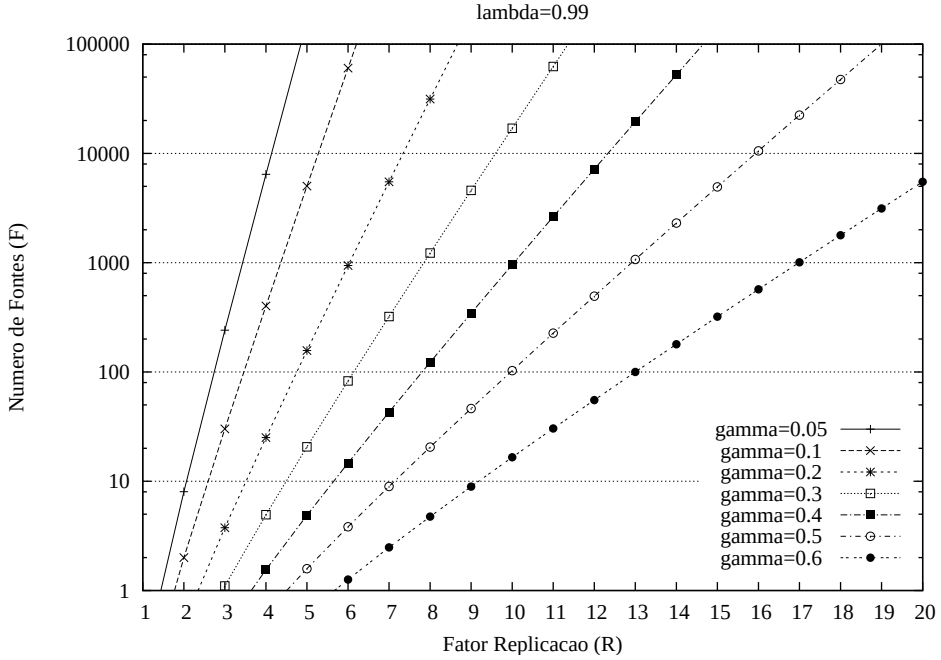

Figura 4.11: Efeito da indisponibilidade dos nodos na disponibilidade do arquivo.

vamente, no mínimo, no máximo e na média a disponibilidade λ exigida. Por exemplo, para $lambda = 99\%$, $R = 10$ e uma variação de γ entre 40 e 50%, o serviço responderia que, no máximo $F_{min} = 100$ fontes (ou seja, $n = \frac{F}{R} = \frac{100}{10} = 10$ fragmentos) poderiam ser utilizadas para garantir no mínimo 99% de disponibilidade.

Determinando o Número de Fontes

No primeiro dos serviços oferecidos pelo Serviço de Criação de Arquivos, assume-se que o usuário já sabe qual o melhor número de fontes a ser usado. Já no segundo, embora seja retornado uma faixa de possíveis valores para esse parâmetro, não se sabe qual o valor de F otimizaria mais as operações sobre o arquivo. Em ambos os casos, se faz necessário auxiliar o usuário na escolha desse parâmetro.

Todas as operações que envolvem a transferência de um arquivo apresentam um comportamento semelhante. Para cada fragmento, se faz necessário (i) localizar os nodos que armazenam uma de suas cópias, (ii) negociar um protocolo para movimentação dos dados e obter informações necessárias para fazer o *download* de cada fonte, e (iii) fazer a transferência do fragmento propriamente dito. Como os fragmentos são gerenciados independentemente uns dos outros, é possível obtê-los em paralelo. Para um mesmo fragmento, pode-se transferir um trecho diferente a partir de cada réplica existente.

Define-se o **custo de transferência** do i-ésimo fragmento (C_i) como sendo a soma do tempo necessário para realizar cada uma dessas três etapas: $C_i = C_{loc_i} + C_{negoc_i} + C_{transf_i}$. Como as etapas (ii) e (iii) podem ainda ser realizadas em paralelo para cada cópia (R) existente, o custo C_i é dado pela transferência do trecho mais demorado (proveniente da fonte mais lenta):

$$C_i = C_{loc_i} + max\{C_{negoc_{ij}} + C_{transf_{ij}},\ 0 \leq j < R\} \tag{4.8}$$

Como os fragmentos são transferidos em paralelo, o custo para obter o arquivo como um todo (C) é dado pelo tempo para concluir o fragmento mais demorado. Ou seja:

$$\begin{aligned} C &= max\{C_i,\ 0 \leq i < n\} \\ C &= max\{C_{loc_i} + max\{C_{negoc_{ij}} + C_{transf_{ij}},\ 0 \leq j < R\},\ 0 \leq i < n\} \end{aligned} \tag{4.9}$$

Ao analisar as Eqs 4.8 e 4.9, percebe-se que, para um mesmo número de fontes F, o fato de usar um fator de replicação maior resulta em um menor número de operações de localização (C_{loc}) e, conseqüentemente, o custo para obter o arquivo tende a ser ligeiramente menor. Por exemplo, para $F = 10$, $n = 1$ e $R = 10$ o custo fica:

$$C = C_0 = C_{loc_0} + max\{C_{negoc_{0j}} + C_{transf_{0j}},\ 0 \leq j < 10\} \tag{4.10}$$

Já quando, para o mesmo número de fontes ($F = 10$), $n = 10$ e $R = 1$, tem-se:

$$C = max\{C_{loc_{i0}} + C_{negoc_{i0}} + C_{tranf_{i0}},\ 0 \leq i < 10\} \tag{4.11}$$

Ou seja, enquanto que na Eq. 4.10 é realizado uma única operação de localização (C_{loc}), na Eq. 4.11 são realizadas 10 vezes essa mesma operação, ainda que mantendo o mesmo número de operações de negociação (C_{negoc}) e transferência (C_{transf}). Para fins de análise, será então considerado sempre o pior caso, ou seja, quando $F = n$.

Uma vez que a operação de localização é realizada por um *lookup* na rede P2P, seu custo depende do número de nodos virtuais (N) presentes no sistema. Ao utilizar algoritmos como o Kademlia, Chord e Pastry para prover o *lookup*, esse custo é dado de forma logarítmica ($C_{loc_i} = \mathcal{O}(\log N)$). No entanto, pode-se considerar C_{loc_i} constante após a rede P2P ter atingido um número de nodos suficientemente grande e estagnado seu crescimento.

Por envolver sempre a troca de um mesmo número de mensagens, a operação de negociação (C_{negoc_i}) impõe um custo fixo. Basicamente, ao solicitar um endereço para o *download* do fragmento, anexa-se uma pequena lista com os protocolos suportados, recebendo-se como resposta uma URL que identifica o protocolo de movimentação de dados escolhido e o endereço para *download* do fragmento.

Por fim, o procedimento de transferência (C_{transf_i}) é constituído de duas etapas: na primeira ($W_{connect_i}$), estabelecem-se canais de comunicação e os nodos envolvidos realizam a autenticação mútua. Na segunda ($W_{download_i}$), ocorre a transferência propriamente dita:

$$C_{transf_i} = W_{connect_i} + W_{download_i} \tag{4.12}$$

Os custos C_{loc_i}, C_{negoc_i} e $W_{connect_i}$ representam o sobrecusto de gerenciamento para um determinado fragmento. Considerando-se o arquivo como um todo, seu **custo de gerenciamento** (G) é o tempo necessário para executar as operações de gerenciamento de todos os seus fragmentos (fontes). Quanto maior é o número de fragmentos (fontes) escolhido, maior é o custo total G. Mesmo que os fragmentos sejam obtidos em paralelo, estima-se que tal proporção continue existindo pois as operações em questão competem pelo uso dos mesmos recursos. Neste caso, a constante de proporcionalidade (K) seria ligeiramente menor do que no caso sequencial.

Em resumo, o custo de gerenciamento (G) é dado por:

$$G \propto F \tag{4.13}$$

$$G = F \times K \tag{4.14}$$

Já o custo $W_{download_i}$ depende do tamanho do trecho a ser transferido (s_i) e da vazão de rede atingida (v_i). Caso uma única fonte fosse utilizada (caso seqüencial), teria-se:

$$W_{download_i} = \frac{s_i}{v_i} \tag{4.15}$$

$$W_{download_0} = \frac{s_0}{v_0} \tag{4.16}$$

$$W_{download} = \frac{s}{v} \tag{4.17}$$

Onde s é o tamanho do arquivo e v a vazão utilizada na transferência.

No entanto, no JavaRMS os F trechos são obtidos em paralelo. Considerando que esses trechos tenham um mesmo tamanho e que as transferências iniciem simultaneamente, o tempo para concluir a transferência ($W_{download}$) pode ser dado por:

$$W_{download} = \frac{s}{\sum_{0 \leq i < F} v_i} \tag{4.18}$$

$$W_{download} = \frac{s}{v} \tag{4.19}$$

Ou seja, a quantidade de dados transferida é sempre a mesma, mas pode-se conseguir aumentar a vazão agregada v, desde que a rede apresente uma configuração assim favorável. Por exemplo, quando:

- Existem congestionamentos no caminho de rede que liga a origem e o destino dos dados;
- A taxa de *upload* dos dados na origem é inferior a de *download* no destino;
- Outras aplicações concorrem pelo uso dos recursos de rede.

É devido ao aumento da vazão que se consegue melhorar o desempenho da transferência. Embora as etapas $C_{loc_i} + C_{negoc_i} + W_{connect_i}$ de um determinado trecho sejam realizadas em paralelo com a etapa $W_{download_i}$ de outro, há pouca concorrência no uso de recursos. Enquanto $C_{loc_i} + C_{negoc_i} + W_{connect_i}$ é sensível à latência de rede (necessidade de trocar algumas poucas mensagens) e à CPU (procedimentos de autenticação e de instanciação de clientes do protocolo de transferência escolhido), $W_{download_i}$ é sensível à largura de banda da rede. Logo, a execução de uma não influencia significativamente no tempo para a conclusão da outra quando o tamanho do arquivo é suficientemente grande. Portanto, o tempo para concluir a transferência do arquivo é dado pelo maior dos dois custos:

$$C = max\{G,\ W_{download}\} \tag{4.20}$$

$$C = max\left\{F \times K,\ \frac{s}{v}\right\} \tag{4.21}$$

Como não há influência significativa de um sobre o outro, não há problemas em escolher o maior número de fontes possível. Quanto maior for o número de nodos fornecendo dados, maior serão as chances de conseguir atingir uma maior vazão agregada, diminuindo-se então o tempo de $W_{download}$. No entanto, se o número de fontes é grande demais, o custo de gerenciamento (G) ultrapassa o de $W_{download}$ e, neste caso, gasta-se

mais tempo com o gerenciamento dos fragmentos do que com a sua transferência propriamente dita. Logo:

$$G \leq W_{download} \tag{4.22}$$

$$F \times K \leq \frac{s}{v} \tag{4.23}$$

$$F \leq \frac{s}{K \times v} \tag{4.24}$$

Como é difícil calcular qual a provável vazão agregada v para um determinado número de fontes, propõe-se que F seja determinado de acordo com o menor valor de $W_{download}$ possível. Tal valor ocorre quando a vazão máxima é utilizada, ou seja, quando toda a banda de rede disponível ao nó que faz o *download* está em uso:

$$F \leq \frac{s}{K \times v_{max}} \tag{4.25}$$

Considerando que se deseja o maior número de fontes possível, obtém-se:

$$F = \frac{s}{K \times v_{max}} \tag{4.26}$$

Esse valor de F dado para o $W_{download}$ mínimo pode, no entanto, limitar a vazão agregada em outras configurações de rede. Por exemplo, se a banda máxima de *dowload* do nó é de $32Mbps$ (4 MB/s), K é $0,5$ segundos e $s = 50$ MB, a Eq. 4.26 resulta em $F = 25$. Como 25 máquinas espalhadas geograficamente na Grade podem não ser capazes de fornecer dados para ocupar todos os $32Mbps$ de banda do nó, este valor de F seria um limitante. Se fossem utilizadas $F = 30$ fontes, por exemplo, teria-se 5 máquinas a mais fornecendo o arquivo e portanto a probabilidade de atingir a vazão máxima seria maior. Por outro lado, este caso necessitaria no mínimo $G = 30 \times 0,5 = 15$ segundos para concluir a transferência, mesmo que toda a banda disponível fosse utilizada ($W_{download} = \frac{50}{4} = 12,5$ segundos). É necessário, portanto, achar um bom compromisso entre o aumento do número de fontes F e a vazão v que ela vem a agregar. O objetivo desse ajuste é minimizar o custo da transferência (minimizar C na Eq. 4.21).

Relembrando que no JavaRMS podem ainda existir réplicas dos fragmentos, tem-se $n \leq F \leq nR$ fontes disponíveis para o ajuste. Como n define o número mínimo de fontes possíveis de serem usadas, propõe-se que:

$$n = \frac{s}{K \times v_{max}} \tag{4.27}$$

No exemplo anterior, se existissem $R = 2$ cópias de cada fragmento, um nó teria que obrigatoriamente arcar com os custos para acessar os $n = 25$ fragmentos ($12,5$ segundos). Caso a vazão máxima não fosse atingida com essas 25 fontes, teria-se ainda disponível outras 25 para encontrar um bom compromisso entre o número de fontes e o tempo de $W_{download}$. Dessas 25 fontes adicionais, o nó poderia utilizar, por exemplo, apenas 3. Caso a rede viesse a apresentar um congestionamento, o nó poderia aumentar este número a fim de aumentar a vazão agregada e minimizar novamente a Eq. 4.21. Fazer o ajuste adaptativamente durante a transferência é tema para pesquisa futura.

A análise acima é feita para a transferência partindo de um nó em específico. No entanto, como no ambiente de Grade os arquivos estão disponíveis globalmente, um mesmo arquivo pode ser requisitado de diferentes máquinas. Como pode existir heterogeneidade

de *hardware* e de rede entre os nodos, estes podem apresentar diferentes valores para os parâmetros K e v_{max}.

Propõe-se então que sejam usados valores de K e v_{max} de nodos onde o tempo de transferência é crítico. Tal caso ocorre quando a máquina apresenta pouca CPU e está colocada em uma rede com alta latência e vazão. Ou seja, quando K e v_{max} apresentam valores elevados. Casos onde v_{max} é baixo não são críticos pois, como a rede é lenta, o tempo para concluir $W_{download}$ tende a ser elevado. Mesmo que se tenha pouca CPU disponível, há tempo para iniciar a transferência dos n fragmentos.

No caso crítico, existe um custo K elevado para iniciar o *download* a partir de uma fonte. Simultaneamente, a transferência é rápida demais (muita banda disponível) para dar tempo de iniciar várias fontes em paralelo. Nessa configuração, não há vantagens em utilizar um grande número de fragmentos. Portanto, o número de fragmentos F dado pela Eq. 4.26 tende a ser pequeno. Como mostrou-se na seção anterior, essa decisão também ajuda a manter uma alta disponibilidade para o arquivo a um baixo custo de armazenamento.

Apesar de os arquivos serem inseridos visando as máquinas nesse cenário crítico (e, portanto, evitando que esses nodos tenham um desempenho pior), essa decisão não chega a ser um limitante para o bom desempenho em outras configurações. Como os fragmentos são também replicados para manter uma boa disponibilidade para o arquivo, a tendência é que exista um bom número de fontes, mesmo que o número de fragmentos escolhido seja pequeno.

4.10.4 Serviços de Controle de Popularidade

Outro problema importante que ocorre em sistemas de armazenamento distribuídos diz respeito à popularidade dos arquivos. Em uma Grade de dados como a do HEP, um novo arquivo registrado no sistema com os últimos dados coletados de um experimento, ou mesmo um arquivo com resultados de pesquisa importantes gerados por um físico, tende a ser requisitado com uma maior freqüência. Com o tempo, ainda podem surgir outros dados que venham a ser mais populares, exigindo que o sistema se adapte para atender à demanda sem desperdiçar recursos.

O emprego da técnica de nodos virtuais não resolve esse problema, pois um objeto de dados muito popular estaria sempre sob responsabilidade de um mesmo nó, independentemente do número de identificadores a ele atribuído (CAI; CHERVENAK; FRANK, 2004). Ao fragmentar o arquivo, no entanto, consegue-se distribuir a carga de gerenciamento entre um conjunto de nodos distintos. Entretanto, a alta fragmentação incorre em uma baixa na disponibilidade do arquivo como um todo. Ainda que essa disponibilidade não fosse prejudicada, usar fragmentação para adaptar o sistema à demanda exigida pelo arquivo necessitaria ter que redimensionar e redistribuir os fragmentos. Além de ser um procedimento demorado, precisaria-se que os nodos fossem autorizados pelos usuários a mudar os pedidos de arquivo por eles emitido.

Assim como a fragmentação, a técnica de replicação também distribui a carga de gerenciamento entre um conjunto maior de nodos. O número de réplicas R deve possuir um valor suficientemente grande para atender à demanda pelo arquivo. Entretanto, como a popularidade do arquivo pode apresentar grande variabilidade, usar um valor elevado para R justamente para atender a demanda em momentos de pico desperdiçaria recursos de armazenamento durante os períodos de baixa popularidade. Como esse custo de armazenamento ainda é debitado da cota dos usuários, os usuários não se sentiriam motivados a inserir no sistema arquivos com alta popularidade. O aumento do número de réplicas (R)

causaria então desvantagens para o dono do arquivo a fim de beneficiar outros usuários.

Como não é justo "cobrar" do usuário que disponibilizou o arquivo, utiliza-se as técnicas de ***caching*** e **replicação automática** (GEELS; KUBIATOWICZ, 2002) para atender às diferenças de popularidade dos dados. Por replicação automática se entende o processo de decidir quando e onde (em qual máquina) criar novas réplicas para um determinado arquivo. Já o *caching* teria o objetivo de definir como a nova réplica seria criada em uma máquina, ou seja, como seria utilizado o espaço de armazenamento disponível e como as cópias na *cache* seriam substituídas. Assim, novas réplicas do arquivo são criadas automaticamente conforme a demanda aumenta e podem ser removidas quando a demanda diminui. Essas cópias adicionais são criadas apenas se as máquinas tiverem espaço de armazenamento ocioso. Ou seja, aproveita-se o armazenamento disponível para otimizar o acesso aos dados.

Como existe um custo associado à criação de uma nova réplica, precisa-se antes modelar a popularidade dos dados a fim de saber quando efetivamente a criação de uma nova réplica é vantajosa, ou qual a demanda por um arquivo que justificaria a criação de novas cópias. Para alimentar esse modelo de popularidade, é necessário também prover ferramentas que monitorem a demanda pelos arquivos. Uma vez decidido que uma nova réplica deve ser criada, surge o problema de decidir onde criá-la. Diversos trabalhos sobre replicação automática em redes P2P (GEELS; KUBIATOWICZ, 2002; SIDELL et al., 1996; RHEA et al., 2001) discutem heurísticas e metodologias para tomar essa decisão, porém este trabalho propõe um modelo simples onde as novas réplicas são sempre criadas nos nodos do grupo dos k responsáveis pelo objeto de dados. Enquanto que os R primeiros nodos desse grupo seriam obrigados a armazenar uma réplica do arquivo, os demais $k - R$ poderiam armazenar cópias opcionalmente. Dessa forma, evita-se que sejam necessários mecanismos adicionais para poder localizar as cópias de *cache*. Por fim, precisa-se empregar alguma política para substituição dos dados nessa *cache*.

Como pode se observar, existe uma grande quantidade de aspectos a considerar ao fornecer um serviço para o controle da popularidade dos dados. Embora uma abordagem simplificada possa ser aplicada, esse serviço de fato não será desenvolvido no presente trabalho.

4.10.5 Serviços de Manutenção

Conforme as máquinas da Grade falham permanentemente (ou, por motivos adversos, perdem seus arquivos), o número de réplicas para um determinado fragmento diminui. Pode-se então chegar ao ponto onde não existem mais cópias do bloco em questão, inviabilizando a reconstrução do arquivo. É função dos Serviços de Manutenção criar novas cópias desses fragmentos gerenciados, de forma a manter o número de réplicas R exigidos pelos usuários. O serviço deve também dar manutenção a outras informações necessárias para que os arquivos possam ser reconstituídos (ex.: os pedidos de arquivo). Em resumo, esse componente tem o objetivo de garantir a durabilidade dos dados gerenciados.

Os protocolos de manutenção são executados em três situações básicas:

- Quando é detectado a falha permanente de alguma máquina;
- Quando um nó reingressa o sistema;
- Quando um nó até então desconhecido entra no sistema (ingresso de um novo nó).

Essa terceira situação exige manutenção pois um novo nó será responsável por um subconjunto de objetos da rede P2P, e portanto deverá obter cópias dos arquivos que são

referenciados por esses objetos. Também existe o fato de que, com a entrada dessa nova máquina no sistema, alguma outra máquina deixe o grupo das R máquinas responsáveis por manter arquivos ou fragmentos.

Como a execução das operações de manutenção envolve a transferência de uma potencial grande quantidade de dados (para criar uma nova réplica para cada fragmento anteriormente gerenciado pelo nó que falhou), existe um alto custo a elas associado. Caso as máquinas falhassem com uma alta freqüência, provavelmente não seria possível concluir as transferências antes que a máquina falhada retornasse ao sistema. Assim, o sistema admite que as máquinas se tornem indisponíveis temporariamente. Caso elas venham a ficar nesse estado por um longo período, considera-se que elas falharam permanentemente e só então os protocolos de manutenção são disparados. Por esses mesmos motivos, a máquina que deixa o grupo das R responsáveis só remove fragmentos no momento que o novo nó tiver permanecido no sistema por um intervalo prolongado.

Para detectar as falhas ou ingressos permanentes, o sistema recai sobre o **modelo de estados** para os nodos, que por sua vez é baseado em informações de entrada/saída obtidas junto à rede P2P.

Manutenção na Ocorrência de Falhas Permanentes

Quando ocorre uma transição de estado `indisponível` para o estado `indisponível permanente`, precisa-se gerar uma nova réplica para cada arquivo (fragmento) antes gerenciado pelo nó que falhou. Para cada arquivo (fragmento) existe um grupo de k nodos vizinhos responsáveis pelo seu gerenciamento, onde os R primeiros desses nodos armazenam uma cópia do arquivo (fragmento). Bastaria, portanto, fazer com que o $(R+1)$-ésimo nodo dessa lista viesse a gerenciar uma nova réplica do arquivo.

Entretanto, como o $(R+1)$-ésimo nó do grupo não tem conhecimento do arquivo (fragmento), ele precisa ser informado por seus vizinhos. Dessa forma, JavaRMS propõe que os α primeiros dos R nodos enviem uma mensagem para o antigo $(R+1)$-ésimo nó informando que ele deve obter uma cópia do arquivo (fragmento). O parâmetro α indica o nível de tolerância a ataques, onde tipicamente $\alpha = 3$, podendo chegar ao máximo de $\alpha = R - 1$. Caso apenas o primeiro dos R nodos ficasse responsável por avisar o $(R+1)$-ésimo nó, um invasor poderia extinguir arquivos do sistema simplesmente deixando de enviar as mensagens aos $(R+1)$-ésimos nodos dos arquivos dos quais ele é o primeiro dos R nodos. Já com $\alpha = 2$, um invasor deveria tomar conta dos 2 primeiros dos R nodos. Como existe uma baixa probabilidade de dois nodos vizinhos estarem sob a mesma vigência administrativa e sujeitos a ataques semelhantes (assumindo-se que existe uma grande quantidade de nodos no sistema), um tipo de ataque como esse é bastante improvável. Ao usar $\alpha = 3$, as chances são potencialmente menores.

Na proposta acima, são enviadas α mensagens para cada um dos x arquivos/fragmentos anteriormente armazenados pelo nó que falhou permanentemente. Como na prática a maioria dos x arquivos deve ser informado para os vizinhos mais próximos do nó falhado, JavaRMS propõe que os arquivos/fragmentos sejam agrupados e enviados de uma única vez para cada um dos $(R+1)$-ésimos nodos. O pior caso continua sendo aquele onde cada um dos x arquivos/fragmentos contém um valor para R diferente. Com o agrupamento, porém, o número de mensagens seria reduzido para $x + \alpha$ mensagens ao invés das $x \times \alpha$ mensagens necessárias no esquema sem o agrupamento.

Manutenção no Ingresso de um Novo Nó

O procedimento de manutenção no ingresso de um novo nó é realizado sempre que um nó até então desconhecido entra no sistema. Ele serve para que o novo nó tenha conhecimento dos arquivos (fragmentos) que deverá gerenciar. Esses arquivos serão aqueles cujo o novo nó é um dos R primeiros que compõem o grupo de gerenciamento. Como pelo menos 1 dos vizinhos mais próximos é também um dos R nodos responsáveis por um arquivo, basta o novo nó requisitar a lista de arquivos que deverá armazenar para cada um dos seus k vizinhos mais próximos (onde k é o número de nodos retornados pela operação de localização). Uma vez que vizinhos mais distantes têm menor probabilidade de serem o únicos vizinhos a armazenarem uma das R réplicas em comum com o novo nó, apenas os 3 primeiros vizinhos são de fato consultados pelo novo nó. Caso algum arquivo acabe por não ser informado (seja devido a essa limitação ou a devido um desses nodos estarem agindo de má fé), não existirá problema pois os nodos que deixam de ser um dos R primeiros nodos não removem suas cópias enquanto o novo nó estiver nesse estado. De fato, não existe nenhuma garantia de que o novo nó vá fazer o *download* dos arquivos que estarão sob sua responsabilidade de gerenciamento.

Manutenção no Reingresso de um Nó

Quando ocorre uma transição para o estado `disponível`, faz-se necessário que o nó que reingressou a rede receba informações a respeito dos arquivos (fragmentos) que passaram a ser de sua reponsabilidade no período em que esteve fora. Nesse caso, o nó que entrou deve requisitar a lista de arquivos (fragmentos) para os quais ele é um dos R nodos mais próximos para cada um dos seus k vizinhos. Assim que as possíveis novas transferências são concluídas, o nó em questão envia uma mensagem a cada um desses vizinhos informando que já atualizou sua lista de réplicas. A partir de então, os vizinhos verificam se ainda são um dos R primeiros nodos para cada arquivo e, caso deixem de ser, podem apagar as réplicas correspondentes.

Caso, no entanto, o nó que ingressou tenha mentido quanto à informação de que já fez o *download* do arquivo, então uma réplica a menos do arquivo deixará de ser gerenciada. Porém, no momento em que o nó deixar de ser um dos R primeiros nodos responsáveis por aquele fragmento, a réplica será novamente criada (assumindo que o novo nó que ingressou o grupo dos R primeiros nodos não esteja também agindo de má fé).

5 PROTOTIPAÇÃO

Embora o modelo proposto no JavaRMS compreenda uma grande quantidade de serviços, apenas um subconjunto dos mesmos foi implementado. A idéia do protótipo é validar os serviços que foram considerados parte do núcleo da arquitetura, e cujo comportamento influencia diretamente no bom desempenho dos demais.

O protótipo tem cerca de 5 mil linhas de código, das quais 95% são código Java e os demais 5% são *scripts* para facilitar seu uso por linha de comando. Embora outras linguagens de programação pudessem resultar num desempenho superior, optou-se por usar Java pelos seguintes motivos:

- *Menor tempo de desenvolvimento*: a linguagem apresenta características que permitem o desenvolvimento do protótipo em um tempo menor. As características principais são: (i) a existência de um coletor de lixo, evitando que o programador tenha que gerenciar a remoção de objetos da memória; (ii) a inexistência de ponteiros e (iii) a partir da versão 5, as facilidades para programação concorrente (tipos atômicos, semáforos, primitivas de sincronização nativas a quaisquer objetos, etc);

- *Suporte à certificação X509 (ITU, 2000) e criptografia de chave pública*: a linguagem dá suporte nativo às funcionalidades de segurança necessárias para utilizar os serviços da Grade, tais como assinaturas digitais e funções *hash* com chave – HMAC (KRAWCZYK; BELLARE; CANETTI, 1997);

- *Aplicação I/O bound*: como a execução das operações mais importantes envolve um grande número de operações de entrada e saída (I/O), o tempo de execução depende muito mais da velocidade dos dispositivos de rede e disco do que propriamente do tempo de processamento ou memória;

- *Portabilidade*: existe máquina virtual Java escrita para a grande maioria das plataformas de execução encontradas no ambiente de Grade.

5.1 Visão Geral do Protótipo

O código Java compreende 37 classes que realizam desde operações junto aos serviços P2P até clientes a serem usados por usuários que utilizam os serviços RMS. Cada uma delas foi cuidadosamente desenvolvida para que fossem *thread-safe*, característica que lhe permite executar várias tarefas concorrentemente sem correr o risco de levar o sistema a estados inválidos de operação. As principais classes são:

- `AbstractClient`: inicializa credenciais e instancia um cliente XMLRPC (WINER, 1999) seguro, fornecendo ainda métodos básicos para fazer chamadas remotas;

- `DataBaseService`: contém métodos para acesso a um banco de dados local;

- `LookupService`: fornece meios de acesso para a localização de dados na rede P2P e gerencia os nodos virtuais;

- `MainServer`: é o *daemon* XMLRPC modificado para aceitar conexões seguras (SSL);

- `MaintenanceService`: monitora os eventos de entrada e saída de nodos na rede P2P e dispara procedimentos de manutenção para diversos serviços da arquitetura;

- `QuotaManager`: contém os principais serviços de Gerência de Usuários e Cotas, fornecendo operações para registro, contabilização de espaço ocupado e consulta da cota disponível dos usuários gerenciados;

- `ReplicaManager`: fornece serviços básicos necessários para a Gerência de Arquivos, com operações para registro de pedidos de arquivo e fragmentos e operações para consulta de seus dados;

- `RMS`: é a classe responsável por inicializar e finalizar todos os serviços da arquitetura;

- `ServerRMSInterface`: contém a *interface* RMS que é exposta pelo servidor XMLRPC, fornecendo métodos para acesso os serviços da arquitetura;

- `TransferManager`: faz a função dos serviços de Gerência de Transferências, fornecendo métodos para realizar o *download* em paralelo de diferentes trechos de um arquivo, calcular o *hash* SHA1 do conteúdo de um fragmento e remontar os arquivos a partir de seus fragmentos;

- `UserServices`: fornece métodos para facilitar o uso dos serviços de arquivamento e de gerência de usuários por parte dos clientes RMS;

- `AbstractPetition`: contém funcões para gerar, assinar, codificar, carregar, decodificar e verificar documentos digitais. É usado como base para os pedidos de arquivo e pedidos de cota para usuários e nodos;

- `UserClient`: é uma biblioteca a ser usada no desenvolvimento de clientes que acessam as funções de arquivamento;

- `CAClient`: é uma biblioteca a ser usada no desenvolvimento de clientes que acessam as funções de criação e registro de usuários.

5.2 Suporte à Rede Par-a-Par

Para prover as funcionalidades P2P, utilizou-se a versão 0.3.7 do *toolkit OverlayWeaver* (SHUDO; TANAKA; SEKIGUCHI, 2008; SHUDO, 2008). Esse *software*, escrito na linguagem Java, é um *framework* para o desenvolvimento de aplicações P2P que implementa diversos algoritmos baseados em DHT's, dentre os quais está o Kademlia. Embora existam outras implementações do Kademlia escritas em Java, optou-se pelo *OverlayWeaver* devido à familiaridade já existente com o seu código. De fato, esse *toolkit* já havia sido utilizado com o objetivo de comparar a distribuição dos dados entre diferentes algoritmos P2P, culminando na escolha do Kademlia. Durante essa etapa, alguns dos *bugs* que inviabilizavam o término das simulações foram corrigidos e, posteriormente, enviados aos mantenedores do projeto.

Apesar de fornecer uma grande variedade de opções no projeto de DHT's, o *framework* não fornece suporte a nodos virtuais. Embora fosse desejável que essa técnica estivesse implementada junto ao *middleware* P2P para possibilitar a otimização da comunicação entre os nodos virtuais e diminuir o uso de memória, ela foi implementada junto à Gerência de Recursos. Assim, evita-se fazer alterações substanciais no código do *framework*, de forma a não dificultar atualizações para versões mais recentes.

A criação dos identificadores no *OverlayWeaver* se dá de forma aleatória. Esse procedimento foi então ligeiramente modificado para poder receber o valor do identificador virtual durante a instanciação do DHT por parte da Gerência de Recursos. O valor do identificador é baseado em informações contidas no pedido de cota para nodos, que é um documento digital assinado pela CA que autoriza o uso do identificador por uma máquina específica.

Modificou-se também a operação de *lookup* para retornar não apenas o nó raiz, mas também todos os $k-1$ vizinhos com identificadores mais próximos à chave procurada. A operação também precisou ser registrada em uma *interface* Java usada pelo *middleware* para poder ser chamada externamente por quem o instancionou, pois a biblioteca apenas utilizava o *lookup* internamente para prover operações mais sofisticadas (funções `get` e `put`).

OverlayWeaver não fornece também serviços para a obtenção dos eventos de entrada/saída de nodos. Na versão considerada, a única opção possível nesse sentido é a de gravar `logs` em disco com essas informações. Dessa forma, precisou-se modificá-lo para que as notificações de entrada/saída pudessem ser compartilhadas com as aplicações que o utilizam. Implementou-se então uma estratégia do tipo produtor/consumidor (TANENBAUM, 2001). Nela, cada item do *buffer* compartilhado representa um evento de entrada ou saída de um nó na rede P2P. Além da descrição do nó causador do evento (um par <`NodeID, Endereço`>), esses itens contêm um *timestamp* e o tipo da operação (entrada ou saída). Um evento de entrada é gerado sempre que uma mensagem é recebida de um nó até então desconhecido. Um evento de saída é disparado sempre que uma entrada das tabelas de roteamento é substituída, ou seja, quando o nó por ela referenciado não estiver respondendo a mensagens, sendo considerado indisponível. Os consumidores dos itens são *threads* Java dos Serviços RMS que são notificadas sempre que um item é colocado no *buffer*. Após consumí-los, decidem se procedimentos de manutenção devem ou não ser disparados.

Por fim, precisou-se reescrever o *driver* utilizado no *toolkit* para fazer a comunicação entre os nodos na rede P2P. Tanto a versão nova como a original utilizam o protocolo UDP para troca de informações. Os motivos que levaram à reescrita do código foram:

- Dificuldade para correção de um problema de comunicação cruzada (*crosstalk*), onde o gerenciamento incorreto dos *sockets* e das mensagens UDP fazia com que uma *thread* recebesse respostas expiradas referentes à requisições feitas por outras *threads*;
- Dificuldade para correção de erros de concorrência (identificação das seções críticas de código);
- Necessidade de tornar o *driver* de comunicação de acordo com o especificado para o Kademlia, pois o original do *toolkit* era genérico para ser usado junto a outros algoritmos (Chord, Pastry, etc) e apresentava um comportamento ligeiramente diferente;
- Necessidade de implementar mecanismos de tratamento de erros de comunicação mais confiáveis, tais como a conclusão do *lookup* mesmo quando respostas inválidas são recebidas de alguns nodos. O mecanismo original para reenvio das mensagens também não considerava o fato de as mensagens estarem expirando devido a congestionamentos da rede (controle de fluxo).

Dentre as configurações possíveis pelo *toolkit*, utilizou-se o protocolo UDP para a comunicação, onde o *timeout* para o envio dos pacotes foi fixado em 2 segundos. Um *pool* contendo um máximo de 20 *sockets* por instância DHT foi configurado junto às funcionalidades para o envio de mensagens UDP. Apenas uma única *thread* é usada para consultar outros nodos durante a operação de *lookup*, embora o artigo original que descreve o Kademlia sugira o uso de 3 *threads*. Cada nó consultado retorna sempre uma lista com os $k = 5$ nodos por ele conhecidos que se encontram mais próximos à chave. Os *k-buckets* no Kademlia podem receber até $k = 5$ entradas e os identificadores dos nodos foram truncados em $m = 40$ *bits* para facilitar a análise dos testes com o protótipo.

5.3 Serviços de Comunicação

O Serviços de Comunicação, usados para fornecer Serviços RMS, foram implementados em cima da biblioteca Apache XMLRPC 3.0 (APACHE XML-RPC, 2008). Tanto o cliente RMS quanto o servidor foram modificados para dar suporte ao protocolo SSL. Dessa forma, as credenciais do nó ou do usuário da Grade são passadas a ele durante sua instanciação e usadas junto aos *sockets* seguros. Durante a negociação de uma conexão HTTP(S), tanto os clientes quanto os servidores são autenticados, onde os servidores são ainda verificados quanto ao *hostname* contido no seu certificado de Grade.

Embora pudesse ser modificado para ser carregado por um servidor de aplicação (ex.: Tomcat, JBoss), o servidor XMLRPC fornecido por essa biblioteca é na verdade um *daemon* Unix convencional. As diferenças deste para a versão que roda sobre um servidor de aplicação ou um servidor *web* é que ele não implementa na totalidade o protocolo HTTP1.1 (FIELDING et al., 1999). Na implementação utilizada, por exemplo, não há suporte ao modo de comunição conhecido como *http chunk mode*. Assim, cada requisição HTTP contém o cabeçalho completo ao invés de ela ser quebrada em diversos pedaços (*chunks*), enviados sequencialmente no protocolo original. Essa limitação não chega a ocasionar problemas de desempenho pois no JavaRMS as chamadas/respostas a serviços RMS cabem em uma única requição HTTP. Entretanto, o servidor dá suporte ao *http keep-alive*, permitindo que uma mesma conexão TCP seja usada para transmitir diversas requisições HTTP, evitando-se assim o custo de estabelecimento de novas conexões

quando requisições sucessivas são feitas para um mesmo servidor. O *daemon* em si é todo *multi-thread* e utiliza um *pool* de 30 *threads* para a atender a requisições de clientes XMLRPC (os clientes dos serviços RMS).

Já no lado cliente, testou-se 3 implementações: *SunXmlRpcHttpClient*, *SimpleXmlrpcClient* e *JavaHttpConnectClient*, optando-se pela primeira (*SunXmlRpcHttpClient*) por ser o única a dar suporte a *http keep-alive*. Precisou-se ainda modificá-lo para dar suporte ao SSL em cima das credenciais da Grade. Para poder carregar essas credenciais, que são fornecidas junto a certificados X509 e criptografia de chave pública da Grade, utilizou-se funcionalidades de segurança do Java (pacote `java.security`) juntamente com a biblioteca *Not-Yet-Commons-SSL* (DAVIES, 2008) para poder carregar as chaves privadas armazenadas em disco no formato PKCS8 (não suportado nativamente pelo Java).

5.4 Serviços de Monitoramento

Embora a arquitetura dos Serviços de Monitoramento seja composta por 5 componentes, ná prática pouco dela foi implementada. Nos Serviços de Monitoramento de Estados Remotos, implementou-se apenas a geração de eventos baseada nas transições de estado observadas. Essas transições foram capturadas através dos Serviços P2P, mais precisamente através do consumo de itens do *buffer* de eventos de entrada/saída de nodos. Não há implementação para os demais serviços de monitoramento. A idéia é futuramente utilizar as ferramentas de monitoramento já existentes da Grade, tais como o MonaLISA (LEGRAND et al., 2004) e o Globus MDS (ZHANG; SCHOPF, 2004).

Os serviços de manutenção que utilizam o monitoramento de estados simplesmente executam seus protocolos toda vez que um nodo entra na rede, independentemente do tipo de mudança de estado provocada. Como ainda não se implementou o Monitoramento da Disponibilidade, o sistema não toma providências de manutenção para assegurar a disponibilidade dos dados gerenciados.

5.5 Gerência de Transferências

Os Serviços de Negociação de Protocolo foram simplificados para apenas viabilizar a movimentação de dados utilizando o protocolo GridFTP. Implementou-se um *wrapper* em Java para chamar externamente o cliente GridFTP que acompanha o *toolkit* do Globus. Ao instanciar o *wrapper*, são a ele passados a URL para o arquivo a ser transferido, o *offset* inicial e o número de *bytes* a transferir. A classe se encarrega de coletar a saída fornecida pelo cliente GridFTP e formatá-la para que seja utilizada pelo JavaRMS. Também foram implementados métodos para aguardar o término da transferência (síncrono, bloqueante), registrar objetos para receber notificação ao término da transferência (assíncrono), e também parar uma transferência.

Já a implementação dos Serviços de Transferência se encontra na classe *ReplicaManager* do protótipo e contém o algoritmo proposto. Todas as fontes disponíveis são sempre utilizadas na operação `transfer` e, após o término da transferência de cada um dos trechos, calcula-se *hash* SHA1 do conteúdo do arquivo em disco. Caso o resultado da verificação seja falso, gera-se uma exceção que deverá ser tratada pelo serviço que requisitou a transferência.

5.6 Gerência de Recursos

Uma vez que *OverlayWeaver* não dá suporte a nodos virtuais, nos serviços de Gerência de Nodos Virtuais intanciou-se diversos DHT's, cada um utilizando uma porta local diferente através da qual a instância se comunica com as demais, sejam elas pertencentes ao mesmo nó ou a nodos remotos. Dessa forma, consegue-se ter uma estimativa do uso de *sockets* para comunicação quando o número total de nodos (virtuais) aumenta na rede lógica P2P.

Cada DHT criada corresponde a um nó virtual especificado por um pedido de cota armazenado em disco, colocado em um diretório pré-configurado pelo administrador do recurso. Ainda que não exista uma implementação de um serviço RMS específico para gerenciar os nodos virtuais ativos, o controle do uso dos recursos de disco é possível pela retirada ou reposição manual desses documentos digitais por parte do administrador, porém se faz sempre necessário parar e reiniciar os serviços RMS após essas alterações. Para limitação do uso dos recursos de rede, o administrador também deve configurar manualmente uma *interface* de rede com essa restrição e especificar que ela deverá ser usada pelo JavaRMS nos seus arquivos de configuração. A opção adotada para fazer esse ajuste manual é através da ferramenta *shapecfg* (LAMETER; WATSON, 2008).

O Serviço de Localização utiliza sempre a primeira DHT instanciada para fazer os *lookups* ao invés de procurar aquela que tem um identificador virtual com valor mais próximo à chave procurada. Também implementou-se junto a esse serviço os procedimentos para geração das chaves baseadas em informações dos objetos de dados.

5.7 Gerência de Usuários e Cotas

Para viabilizar o controle de usuários e a contabilização do espaço por eles utilizado, primeiramente disponibilizou-se ferramentas para emissão, carga, transmissão e verificação dos documentos digitais usados no sistema. Esses documentos são:

- O pedido de cota para nodos;
- O pedido de cota para usuários;
- Os pedidos de arquivos.

Quando um desses documentos precisa ser enviado a outro nó, ele antes é formatado e codificado, gerando um conjunto de caracteres na base 64, assim como ocorre nos certificados X509. Esses documentos são sempre verificados quando carregados pelas entidades que os recebem, garantindo-se assim sua autenticidade e integridade. Para assiná-los, utilizam-se as chaves privadas correspondentes às chaves públicas presentes nos certificados X509 da Grade. Como a Gerência de Usuários e Cotas também distribui os certificados X509 e provê serviços para sua localização e aquisição, qualquer dos certificados X509 pode ser encontrado e usado para verificar o documento digital emitido pelo usuário ou pela autoridade certificadora. Adotou-se uma única CA para todo sistema, onde seu certificado X509 deve ser colocado manualmente pelo administrador em um diretório de configuração específico. Ao usar uma única CA simplificou-se os procedimentos de verificação das assinaturas.

Existe sempre um número fixo de nodos envolvidos no gerenciamento das informações de cada usuário registrado. Esse número foi configurado para ser igual a 5, sendo escolhido arbitrariamente. Entretanto, pode-se alterá-lo para que assuma qualquer valor

entre 1 (não recomendado por motivos de segurança) e k (o tamenho do *k-bucket*), onde este último representa o tamanho da lista de nodos retornados pelas operações de *lookup*. Nodos que não estão envolvidos no gerenciamento de um usuário requisitado geram exceções passadas remotamente pelo protocolo XMLRPC ao nó que solicitou as informações do usuário.

Programou-se ainda as operações para contagem do espaço utilizado por cada usuário, para verificar se um determinado arquivo já fora contabilizado para um usuário, para registrar usuários e para solicitar a contagem de um arquivo em nome de um usuário registrado. Já os procedimentos de manutenção necessários quando os nodos entram e saem da rede foram implementados em uma classe em separado, que monitora diretamente esses eventos da rede P2P. As operações de manutenção foram programadas para serem executadas na ocorrência de qualquer desses eventos de entrada ou saída, independentemente do estado atribuído a esses nodos vizinhos.

5.8 Gerência de Arquivos

Na prototipação dos serviços de Gerência de Arquivos, cada nó gerencia dois tipos de objetos de dados: os pedidos de arquivo e os fragmentos. Para ambos, o número de nodos envolvidos no gerenciamento é igual ao fator de replicação (R) solicitado ao arquivo. Cada nó oferece um conjunto de operações necessárias para o auto-gerenciamento dessas duas entidades. Ou seja, existe uma *interface* de operações através da qual os nodos informam seus vizinhos sobre a existência desses documentos ou os requisitam. São elas:

- Solicitar ao nó o registro de um pedido de arquivo:
- Solicitar ao nó a exclusão de um pedido de arquivo:
- Solicitar um pedido de arquivo armazenado pelo nó;
- Solicitar ao nó o armazenamento de um fragmento;
- Solicitar ao nó a exclusão de um fragmento;
- Solicitar o endereço para *download* de um fragmento armazenado pelo nó.

Por motivos de segurança, toda vez que uma requisição por essas operações é solicitada a um nó ele antes verifica se é um dos R responsáveis através do valor da chave calculado para esses objetos. A chave usada para os pedidos de arquivo junto à rede P2P é o *hash* SHA1 do nome lógico do arquivo (LFN) concatenado ao nome do usuário da Grade que emitiu o documento. Para os fragmentos, essa mesma função *hash* é aplicada ao seu conteúdo. O resultado do SHA1 é sempre um vetor de 20 bytes único para cada objeto de dados usado como entrada. Assim, consegue-se localizar deterministicamente qual é o grupo de nodos que deve gerenciar cada objeto, seja ele um pedido de arquivo ou um fragmento.

Ao realizar uma operação de inserção, solicita-se o registro do pedido de arquivo ao grupo de R nodos. Como não existe uma implementação de protocolo para fazer o *multicast* seguro do pedido de arquivo para esses nodos, cada nó é contactado individualmente. O mesmo acontece quando solicita-se o armazenamento de cada fragmento. Essas solicitações de armazenamento são enviadas em paralelo para todos os nodos do grupo. Assim, os R nodos transferem o fragmento simultaneamente. Na operação de remoção também

não existe *multicast* do pedido de arquivo. Cada nó do grupo responsável pelo seu gerenciamento é também contactado individualmente. Na operação de *retrieve*, os diferentes fragmentos que compõem o arquivo são transferidos em paralelo. Todas as R cópias são usadas simultaneamente para transferir cada um desses fragmentos. Devido a limitações do cliente do GridFTP, tornou-se necessário executar ainda uma etapa adicional para unir os fragmentos transferidos separadamente. Essa limitação advém da impossibilidade de usar um *offset* local para o arquivo transferido diferente do *offset* remoto.

Não foi fornecida uma operação específica para o procedimento de modificação de arquivo. Os arquivos modificados são inseridos no sistema como se estivessem sendo registrados pela primeira vez, deixando-se para os serviços de manutenção a tarefa de remover os fragmentos antigos que foram alterados e remover as cópias excedentes para os casos onde o novo fator de replicação é menor que o anterior. Ou seja, o usuário que modificar um arquivo não observará os resultados instantaneamente.

Quando se faz necessário consultar a cota disponível para um determinado usuário, requisita-se essa informação para todos os 5 nodos do grupo que é responsável por gerenciar esse usuário. A resposta considerada correta é aquela que tiver sido respondida um maior número de vezes e, em caso de empate, utiliza-se aquela respondida pelo nó com identificador mais próximo à chave que referencia o usuário. Esse procedimento se faz necessário por motivos de segurança, pois não foi implementado uma operação de coleta seguro. Esse mesmo mecanismo é usado para avaliar as respostas de quaisquer operações solicitadas a um grupo de nodos. Por exemplo, um pedido de arquivo só é considerado recusado se a maioria dos R nodos por ele responsável se recusar a armazená-lo.

Como os serviços para monitoramento da disponibilidade dos nodos não foram prototipados, o Serviço de Criação de Arquivos não foi implementado, pois não se pode obter uma estimativa do parâmetro γ, a indisponibilidade dos nodos. Portanto, deixa-se para o usuário a escolha do número de fragmentos e do fator de replicação a utilizar.

Assim como os procedimentos de manutenção dos Serviços de Gerência de Usuários, as operações de manutenção para os arquivos devido à entrada e saída de nodos foram implementadas em uma classe em separado. Essa classe é quem monitora esses eventos de entrada e saída da rede P2P. As operações de manutenção são sempre executadas na ocorrência de qualquer desses eventos. O serviço de controle da popularidade dos dados não foi protipado.

5.9 Serviços Orientados ao Usuário

Para permitir o uso dos serviços fornecidos pela arquitetura, desenvolveu-se um conjunto de ferramentas básicas usadas pela linha de comando. Essas ferramentas utilizam as duas bibliotecas cliente (`UserClient` e `CAClient`), prototipadas com o objetivo de facilitar o uso dos serviços mais comuns da arquitetura. Essas bibliotecas podem ser usadas tanto por ferramentas que interagem diretamente com o usuário quanto por suas aplicações que necessitam dos serviços de gerência de dados.

Desenvolveu-se ferramentas para realizar as seguintes funções:

- Criar um pedido de arquivo;
- Remover um arquivo;
- Inserir um arquivo;
- Localizar um arquivo;

- Recuperar (obter) um arquivo;
- Criar um pedido de cota para um nodo virtual (a ser usado pela CA);
- Criar um pedido de cota para um usuário (a ser usado pela CA);
- Registrar um usuário no sistema (a ser usado pela CA);
- Requisitar estatísticas de uso do servidor (a ser usado pelo administrador do recurso).

6 EXPERIMENTOS E RESULTADOS

Com o objetivo de validar o modelo proposto, realizou-se uma série de experimentos junto à Grade francesa Grid5000. Cerca de 1500 linhas de código foram programadas em *scripts* usados para automatizar a execução dos experimentos e a coleta dos resultados.

6.1 Ambiente de Execução de Testes

O Grid5000 (GRID'5000, 2008) é uma plataforma de Grade experimental criada a partir da união de 9 *sites* geograficamente distribuídos pela França. O objetivo principal da plataforma é servir como um ambiente de testes para a pesquisa em Computação em Grade. Os *sites* estão interconectados através da rede de pesquisa e educação RENATER (RENATER, 2008), cuja arquitetura básica é composta por linhas de $2,5Gbit/s$ (na prática, eles estão interconectados através de redes virtuais de $1Gbit/s$ ou, quando possível, $10Gbit/s$). Cerca de 1650 máquinas estão atualmente disponíveis, totalizando 4422 núcleos de CPU (de um total de 3326 CPUs).

As máquinas possuem processadores de alto desempenho das famílias AMD Opteron (com freqüências de operação que variam de $2GHz$ a $2,6GHz$ para os modelos com um único núcleo, e de $2,2GHz$ a $2,6GHz$ para os modelos com dois núcleos), Intel Itanium 2, Intel Xeon EM64T (freqüência de $3GHz$ para um único núcleo ou freqüencias de $1,6GHz$ a $2,4GHz$ para dois núcleos) e Intel Xeon IA32 (freqüência de $2,4GHz$). A grande maioria dos processadores dá suporte à arquitetura *x86* de 64 *bits*. De fato, apenas os processadores com suporte a essa arquitetura foram utilizados nos experimentos. Os nodos no Grid5000 ainda apresentam boa estabilidade. De fato, todos os resultados aqui apresentados resultaram de experimentos ausentes de falhas.

As redes usadas para interconectar os nodos dentro de um mesmo *site* são das famílias InfiniBand 10G, Myri-10G e Myrinet-2000 e Gigabit Ethernet. As máquinas pertencem às famílias Dell PowerEdge, HP Integrity, HP ProLiant, IBM System, IBM eServer e Sun Fire, com configurações de memória que variam de 1 *GByte* até 16 *GBytes*. Os recursos de armazenamento fornecidos por cada máquina variam de 70 *GBytes* até 600 *GBytes*.

A infra-estrutura da Grade permite que qualquer $software$ seja colocado nos nodos, incluindo até mesmo outro sistema operacional. Assim, preparou-se uma imagem da distribuição Debian GNU/Linux 4.0 (apelidada *etch*), com *software* compilado para a arquitetura x86 de 64 *bits* e usando o *kernel* versão 2.6.18-3 (amd64). Um conjunto mínimo de *softwares* de terceiros foi instalado para permitir o uso do protótipo. Dentre eles está o *toolkit* do Globus versão $4.0.5$, onde instalou-se apenas a infra-estrutura de segurança (Globus GSI) e o GridFTP (servidor e cliente). Para compilar e instalar o protótipo, utilizou-se a ferramenta Apache Ant $1.7.0$. A autoridade certificadora é uma entidade própria criada através de ferramentas disponibilizadas pelo Globus. Quanto à

rede, foi necessário desabilitar o protocolo IP versão 6 (*ipv6*) devido a problemas com o *middleware* P2P utilizado.

Os motivos para a escolha do Grid5000 como ambiente para a execução dos experimentos foram:

- Possibilidade de escalar para centenas de máquinas **com facilidade**;
- Possibilidade de instalar qualquer infra-estrutura de *software*, tendo-se inclusive acesso de super-usuário;
- O *hardware* disponível atende às exigências.

As demais infra-estruturas de Grade que se teve acesso e que também atendiam às exigências foram o PlanetLab (PETERSON et al., 2006) e a Grade de pesquisa da Universidade de Caxias do Sul (UCS). No entanto, o PlanetLab apresentou dificuldades para aquisição de um grande número de máquinas (na ordem de centenas), além de ser mais difícil isolar os efeitos da sua rede (pois os nodos estão interligados através da Internet). Já a Grade UCS não foi utilizada devido à ocupação constante das máquinas e à dificuldade imposta em acessá-la remotamente por causa da infra-estrutura de *software* de gerenciamento utilizado.

6.2 Avaliação da Operação *Lookup*

O objetivo deste teste é verificar o impacto que os parâmetros do sistema têm no desempenho da operação de *lookup*, que é a principal funcionalidade oferecida pelos serviços P2P. Na lista a seguir, compilou-se os principais parâmetros com influência sobre o desempenho dessa operação:

- *Rede*: devido à necessidade de a operação utilizar esse recurso para trocar mensagens com outros nodos;
- *Número de nodos (reais)*: um aumento no número de nodos na rede P2P implica uma quantidade maior de mensagens trafegando para concluir a operação com êxito;
- *Número de nodos virtuais por nó real*: devido à técnica de nodos virtuais implicar um aumento no número de nodos da rede P2P que devem ser consultados para que a operação de *lookup* seja concluída.

Devido à diversidade de *hardware* e à grande variedade de topologias que pode assumir, modelar a rede de forma a capturar seu impacto sobre a operação de *lookup* é uma tarefa bastante complicada. Nota-se também que esse recurso não é de uso exclusivo da aplicação no ambiente de Grade, podendo existir uma carga de trabalho que certamente influenciaria no tempo de execução da operação mensurada. Para se abstrair do impacto da rede, ainda que refletindo o desempenho da operação, adotou-se como métricas o **número de nodos (HOPs) contactados** e o **número total de mensagens enviadas**. A primeira foi usada para capturar o desempenho observado pelo nó que requisitou a operação, enquanto que a segunda foi usada para dar uma idéia do custo total distribuído entre as diversas máquinas da Grade. Outro motivo para o uso dessas métricas é que alguns fatores relacionados à linguagem Java podem confundir significativamente os resultados obtidos caso o tempo de execução fosse utilizado como métrica. A exemplo, tem-se a

presença do coletor de lixo e a possibilidade de o interpretador Java fazer otimizações em tempo de execução. Para tempos de execução pequenos, o fato de o coletor de lixo ter sido executado invalida completamente a aferição. Quando o tempo de execução do teste é prolongado, o interpretador tem tempo de realizar otimizações mais sofisticadas que culminam em uma diminuição no tempo de execução de operações sucessivas.

O experimento descrito a seguir tem o objetivo de responder às perguntas:

1. Como a operação de *lookup* escala com relação ao número de nodos reais que formam a rede P2P?

2. O impacto do uso de nodos virtuais no desempenho da operação de *lookup* é significativo? Se sim, como se comporta com o aumento do número de nodos virtuais em cada nó real?

3. O impacto do uso de nodos virtuais é independente do número de nodos reais? Ou existe uma correlação entre ambos?

Os testes consistem na execução de 10000 operações de *lookup*, provindas de um único usuário em um único nó do Grid5000. Variou-se o número de nodos reais ($NR = \{1, 4, 16, 64\}$) e o número de nodos virtuais por nó real ($NV = \{2, 8, 32, 128, 512\}$).

A Fig. 6.1 mostra como o número de nodos contactados cresce com relação ao aumento do número de nodos reais (NR), para diferentes quantidades de nodos virtuais em cada nó real (NV). Como o crescimento se dá por uma reta num gráfico em escala logarítmica, é possível concluir que o número de mensagens cresce de forma também logarítmica com o aumento de nodos reais presentes no sistema, seja qual for o número de nodos virtuais gerenciados por cada nó real. A exceção para $NR = 1$ e $NV = 2$ se justifica pelo fato de existirem apenas 2 nodos na rede P2P, sendo então impossível contactar mais de 1 nó para cada operação de *lookup*.

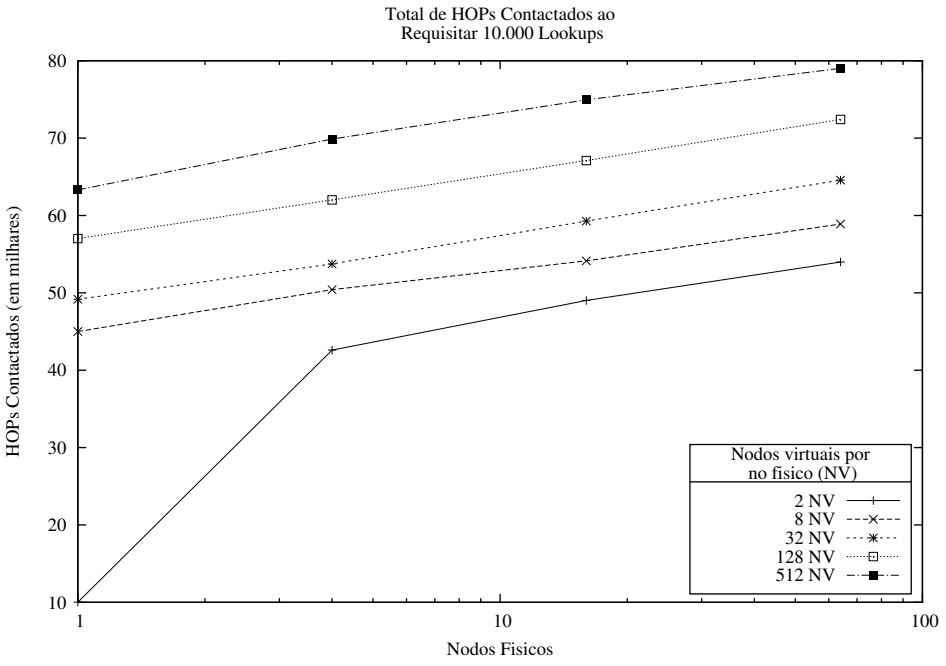

Figura 6.1: Crescimento do número de nodos contactados para concluir 10000 operações de *lookup*, de acordo com o número de nodos virtuais por nó real utilizado.

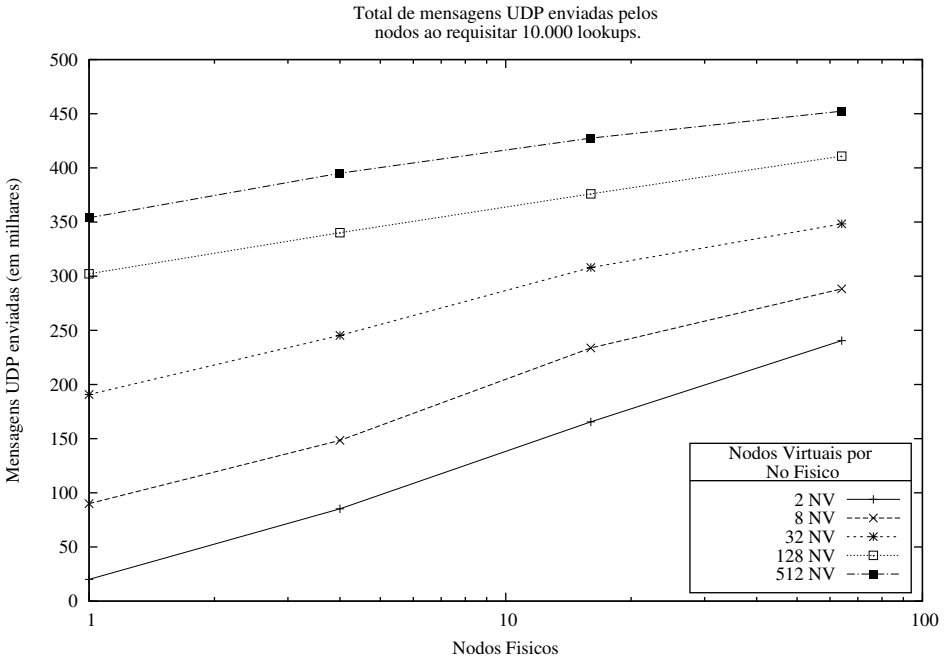

Figura 6.2: Crescimento do total de mensagens trocadas entre todos os nodos para a conclusão de 10000 operações de *lookup*, de acordo com o número de nodos virtuais por nó real utilizado.

Para capturar o impacto geral da operação sobre os nodos, a Fig. 6.2 mostra o número total de mensagens enviadas para esse mesmo experimento. Ao analisar apenas as curvas para $NV = 128$ e $NV = 512$, percebe-se um comportamento semelhante ao da métrica anterior. Ou seja, o número total de mensagens enviadas cresce de forma logarítmica com o aumento do número de nodos reais presentes. Nas demais curvas, porém, o crescimento se dá ligeiramente maior para uma quantidade pequena de máquinas. Como as tabelas de roteamento não estão totalmente preenchidas devido ao número pequeno de nodos presentes nesses casos, não há a necessidade de se trocar algumas mensagens para verificar quais das entradas dessa tabela devem ser substituídas. Por exemplo, quando $NR = 1$ e $NV = 2$, cerca de 20 mil mensagens são enviadas. Ou seja, para cada HOP contactado, são enviadas 2 mensagens (uma com a requisição e a outra com a resposta). Caso existissem nodos o suficiente para preencher as tabelas de roteamento, além das 2 mensagens por cada nó contactado, outras 2 seriam necessárias sempre que houvesse a necessidade de testar a validade da referência a um nó nessa tabela. Esse teste de validade ocorre sempre que um nó ainda não presente na tabela é contactado, servindo para decidir se o nó testado será substituído na tabela pelo nó recentemente contactado. Em nenhum caso foi observada a perda de mensagens.

Para responder à segunda questão levantada, plotou-se nas Figuras 6.3 e 6.4 esses mesmos resultados, porém com o número de nodos virtuais por nó real (NV) no eixo horizontal. Percebe-se que o número de nodos contactados também cresce de forma logarítmica com o aumento de NV, independentemente do número de nodos reais presentes, exceto para os casos onde não existe uma quantidade de nodos suficiente para ser requisitada.

Com o objetivo de responder à terceira e última pergunta, plotou-se na Fig. 6.5 o número de nodos contactados em função de diferentes configurações de NR e NV, de forma que o número **total** de nodos virtuais seja o mesmo ($NT = NR \times NV = 1024$).

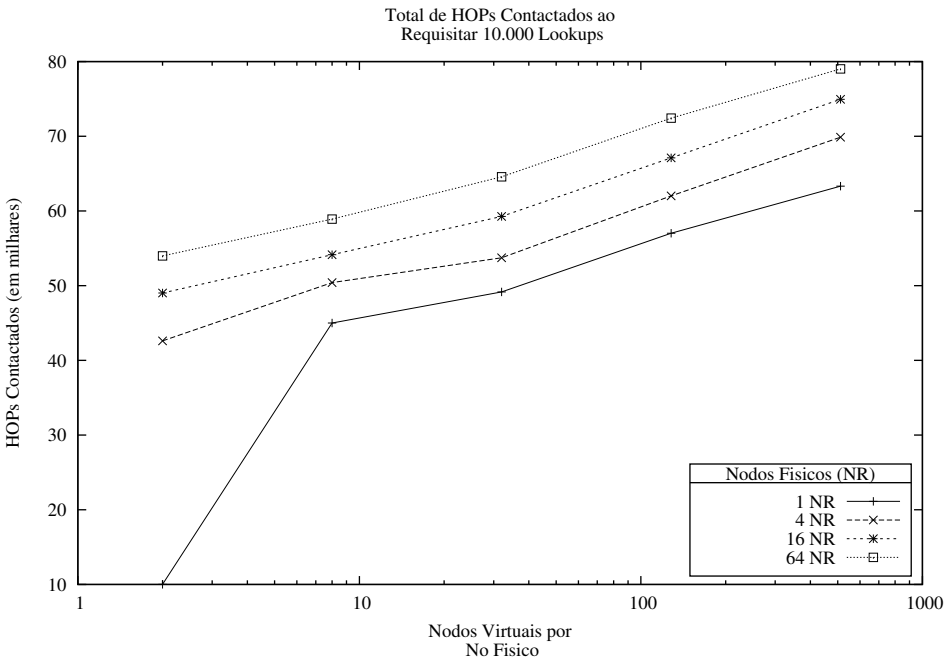

Figura 6.3: Crescimento do número de nodos contactados para concluir 10000 operações de *lookup*, de acordo com o número de nodos reais presentes.

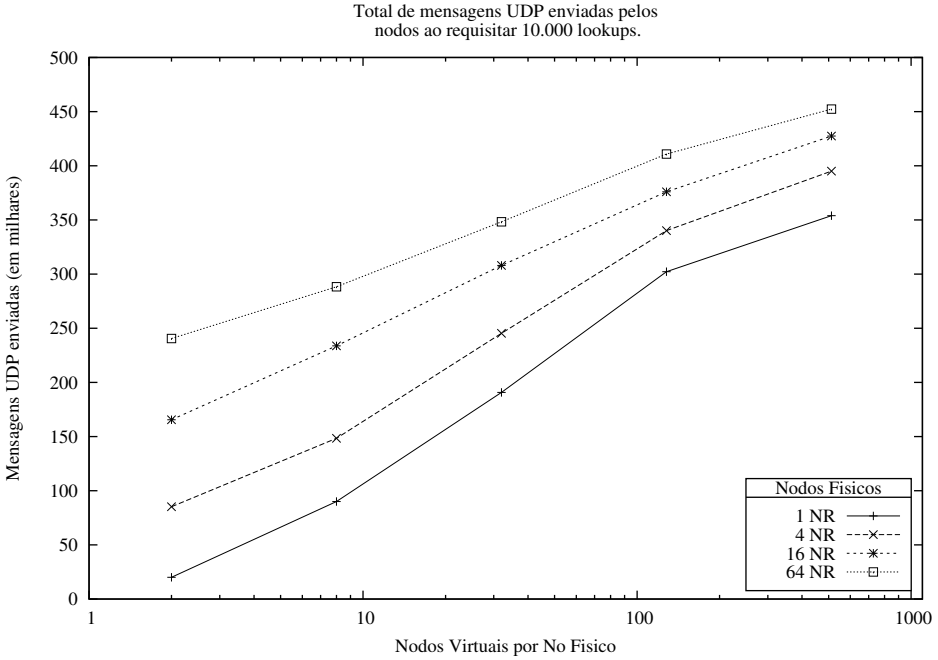

Figura 6.4: Crescimento do total de mensagens trocadas entre todos os nodos para a conclusão de 10000 operações de *lookup*, de acordo com o número de nodos reais utilizado.

O resultado comprova o esperado: não existe correlação entre os parâmetros NR e NV, ou seja, a influência de um não depende do outro na operação de *lookup*. Esse resultado já era esperado pois as DHTs são instanciadas independentemente umas das outras, comunicando-se como se estivessem em nodos distintos.

Embora o número de mensagens e o número de nodos contactados dependa apenas do total de nodos, espera-se que o tempo para concluir a operação de *lookup* seja menor em

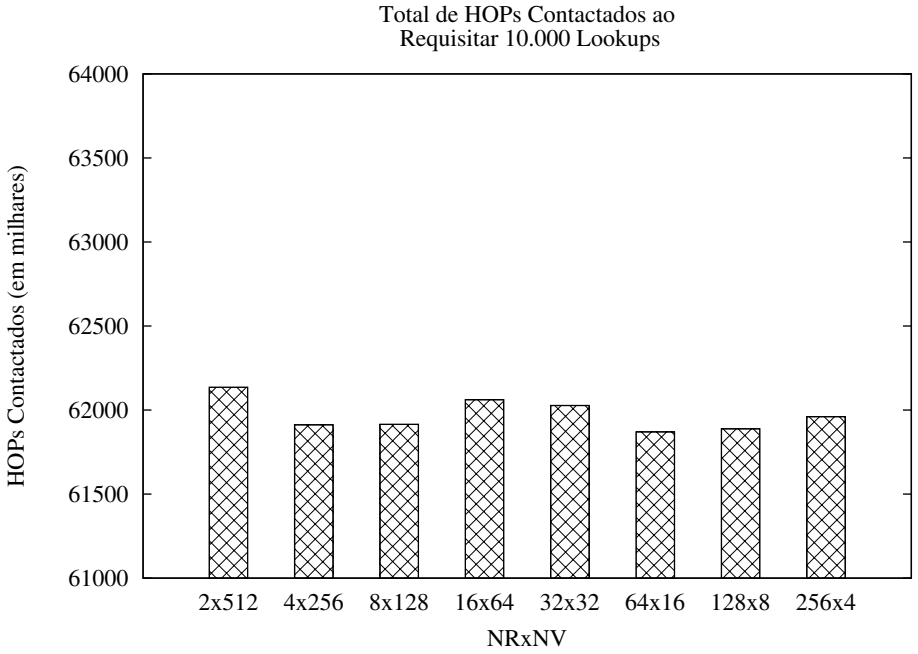

Figura 6.5: Total de nodos contactados ao requisitar 10000 *lookups*, para diversas combinações de NR e NV que resultam em 1024 nodos.

configurações onde exista um grande número de nodos virtuais por cada nó real. Principalmente quando a rede é lenta, o fato de uma grande quantidade de nodos estar agrupado em um único nó real permite que as mensagens sejam enviadas sem precisar usar o *hardware* de rede em si, viabilizando assim a troca de dados de maneira muito mais rápida. No entanto, caso o número de nodos reais seja muito maior que o de nodos virtuais por nó real, poucas mensagens serão enviadas dentro de uma mesma máquina e a rede precisará ser utilizada.

Pelos resultados apresentados nesta seção, é possível concluir que:

- O aumento do número de nodos virtuais tem um impacto semelhante ao do aumento no número de nodos reais, com relação ao número de mensagens enviadas e ao número de nodos contactados;

- O número de mensagens enviadas e o número de nodos contactados cresce logaritmicamente em relação ao número total de nodos;

6.3 Avaliação do Uso de Recursos de Armazenamento

6.3.1 Distribuição dos Dados

JavaRMS recai sobre os serviços P2P (operação *lookup*) para escolher quais nodos devem armazenar os fragmentos de um arquivo. Embora esse esquema seja determinístico, ele não resulta em uma distribuição uniforme dos dados entre os nodos que compõem o sistema (STOICA et al., 2001; BALAKRISHNAN et al., 2003). Segundo (STOICA et al., 2001), as diferenças no número de chaves gerenciadas pelos nodos podem chegar a $\mathcal{O}(logN)$, onde N é o número de nodos que compõem o sistema. Conseqüentemente, nodos que estejam fornecendo uma mesma quantidade de espaço de armazenamento podem apresentar diferenças consideráveis no espaço em disco utilizado.

O fato de usar uma estratégia de replicação dos fragmentos nos vizinhos dos nodos por eles responsáveis ajuda a balancear a distribuição dos dados. Entretanto, a replicação não foi utilizada no JavaRMS com esse objetivo devido ao alto custo de armazenamento associado à criação de um número considerável de réplicas, além de esta técnica apresentar eficácia limitada (usar mais do que 8 réplicas não melhora significativamente a distribuição dos dados – ver Anexo). Para lidar com essas diferenças, utilizou-se a técnica de nodos virtuais. Este experimento tem, portanto, o objetivo de avaliar o impacto que o uso desta técnica tem na distribuição dos dados entre os nodos que compõem o sistema.

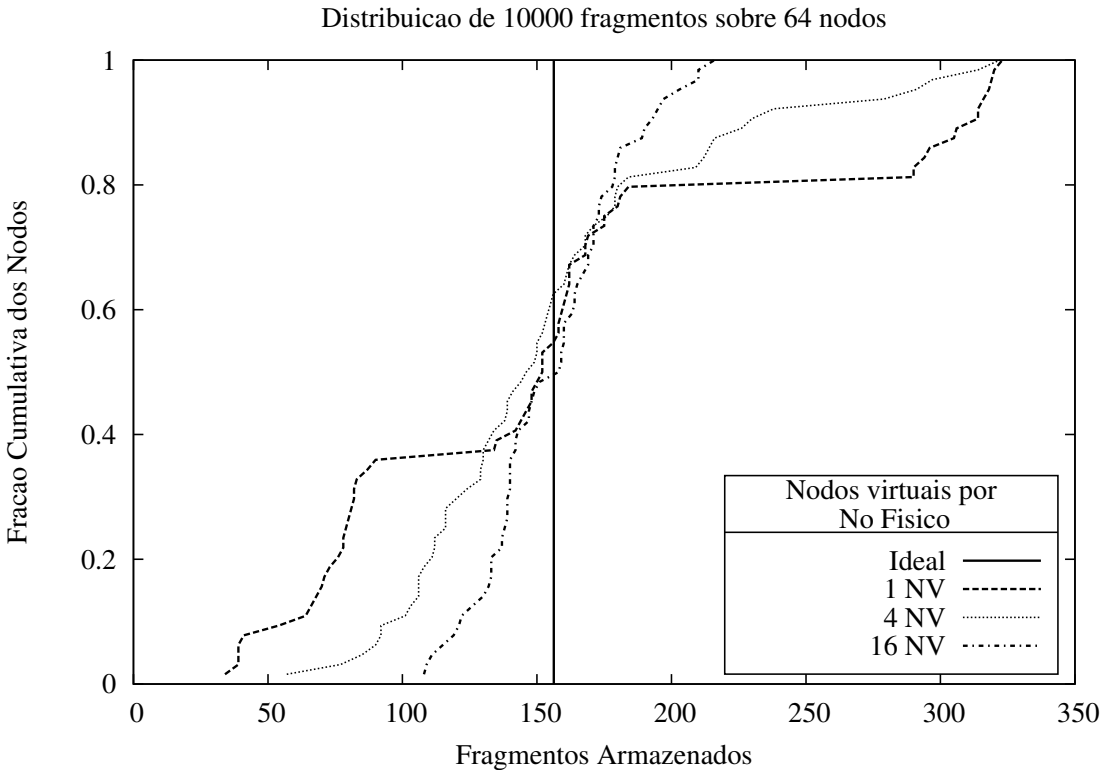

Figura 6.6: Efeito do uso de nodos virtuais na distribuição dos dados.

A Fig. 6.6 mostra a distribuição dos dados após a inserção de 10000 fragmentos num ambiente composto de 64 nodos reais. Em cada curva, todos os 64 nodos utilizam um mesmo número de nodos virtuais. Em uma distribuição ideal, cada nó deve armazenar $\frac{1}{64} \times 10000 = 156$ fragmentos (indicado no gráfico pela linha vertical).

Quando cada nó utiliza um único nó virtual (ou seja, o caso em que nodos virtuais não são utilizados), a distribuição é muito ruim. Nesse caso, quase 40% dos nodos armazenam menos de 100 fragmentos cada um, enquanto que outros 20% armazenam cerca de 300 ou mais fragmentos. Apenas 40% dos nodos armazenam por volta do ideal de 156 fragmentos. Entretanto, quando se usam 4 nodos virtuais por nó real, a distribuição se torna mais uniforme, embora ainda existam nodos armazenando pouco mais de 50 fragmentos simultaneamente a outros armazenando mais de 300. Ao utilizar um número de nodos virtuais ainda maior ($NV = 16$), essas discrepâncias desaparecem. Nesse caso, quase a totalidade dos nodos armazenam entre 100 e 200 fragmentos, valores próximos à curva ideal.

Embora esses resultados demonstrem que $NV = 16$ já seja suficiente para um bom balanceamento, não é possível concluir que esse mesmo valor resultaria em uma boa distribuição para configurações com um maior número de máquinas. Para fazer uma

boa escolha para NV, seria necessário ter uma boa estimativa do número de nodos reais presentes no sistema. Não obstante a essa indefinição, é notável que o uso de nodos virtuais resulta em uma melhora na distribuição dos dados, mesmo para valores pequenos de NV.

6.3.2 Controle de Espaço de Armazenamento

JavaRMS permite que o administrador de um nó controle o uso de seus recursos de armazenamento alterando o número de nodos virtuais (NV) por ele gerenciados. Para avaliar o quão efetivo é esse mecanismo de controle, este experimento configura um ambiente onde os nodos utilizam diferentes números de nodos virtuais. Existe um total de 11 nodos, que utilizam, respectivamente, as seguintes quantidades de nodos virtuais: 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, totalizando 2047 nodos virtuais.

A Fig. 6.7 mostra a quantidade de fragmentos de arquivo que cada um dos 11 nodos armazenou após a inserção de 10000 fragmentos (sem replicação). Percebe-se que o número de fragmentos armazenados é proporcional à quantidade de nodos virtuais gerenciados pelo nó. Por exemplo, para o nó com 128 nodos virtuais, o número de fragmentos gerenciados é 602, valor próximo do ideal $\frac{128}{2047} \times 10000 = 625$.

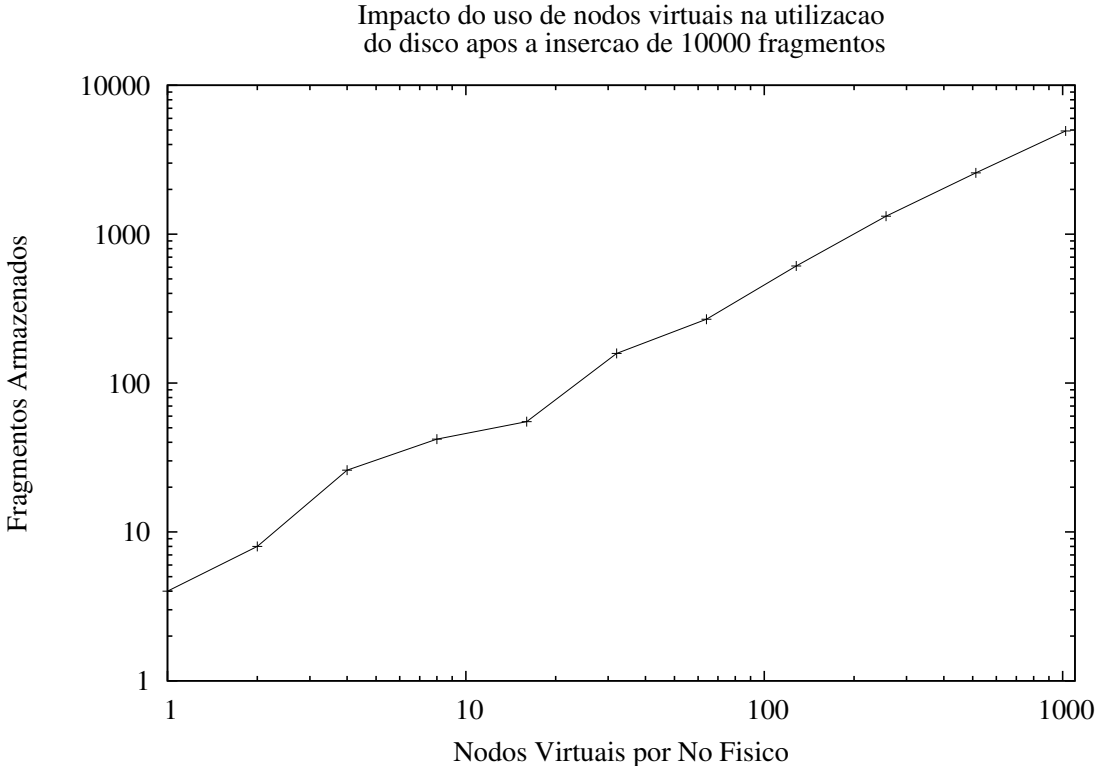

Figura 6.7: Efeito do uso de nodos virtuais no controle dos recursos de armazenamento.

O teste considera um tamanho de fragmento único de 100 *Bytes*. Embora existam diferentes possibilidades para o tamanho dos fragmentos, espera-se que sua distribuição seja uniforme devido aos diversos critérios usados pelos usuários no momento da inserção dos arquivos no sistema. Assim, após uma grande quantidade de arquivos ter sido inserida, o uso do espaço em disco de um nó será proporcional à quantidade de fragmentos gerenciados que, por sua vez, será proporcional ao número de nodos virtuais por ele utilizados. O administrador do recurso poderia então ajustar com facilidade o consumo

do uso de disco simplesmente alterando o número de nodos virtuais ativos.

6.4 Avaliação da Operação *Retrieve*

Dentre as funcionalidades fornecidas pelos serviços RMS, a operação *retrieve* é considerada a mais importante. Ela é uma operação dos Serviços de Arquivamento que consiste em adquirir um arquivo gerenciado pelo sistema, disponibilizando-o localmente ao usuário. Justifica-se essa importância devido ao tamanho potencialmente grande que os arquivos em Grades de Dados como a do HEP podem assumir, exigindo que uma grande quantidade de dados tenha que ser transferida. Embora a operação de *inserção* também envolva a movimentação de arquivos, a operação de *retrieve* tende a ser requisitada com uma freqüência muito maior devido à pouca mutabilidade dos arquivos.

Este teste tem o objetivo de avaliar o desempenho desse serviço, verificando-se o impacto dos parâmetros do sistema mais significativos. As variáveis mais importantes são:

- *Rede*: como os dados estão distribuídos entre os nodos que compõem o sistema, as diversas possibilidades de topologias de rede e a grande heterogeneidade do *hardware* têm grande influência sobre o desempenho da operação. Por ser um recurso compartilhado, a rede poderia ainda estar sendo utilizada por outras aplicações ou mesmo outras operações do JavaRMS, fato que também influenciaria no desempenho do *retrieve*;

- *Tamanho do Arquivo*: como arquivos maiores envolvem a movimentação de uma maior quantidade de dados, o tamanho do arquivo influencia no desempenho da operação;

- *Número de Fontes (F)*: uma vez que o número de fontes representa a quantidade de nodos envolvidos no gerenciamento de um arquivo, o fato de existirem mais fontes permite atender a uma maior demanda de operações de *retrieve*, diminuindo a possibilidade de se formarem gargalos na rede. Nos casos onde apenas uma única operação está em andamento, este parâmetro determina o grau de paralelismo possível para a transferência do arquivo. O número de fontes para um arquivo depende do número de fragmentos e do número de réplicas de cada fragmento ($F = R * n$).

Uma vez que é impraticável testar todas as configurações de rede ao qual o desempenho da operação é sensível, analisou-se apenas dois casos. No primeiro deles, apresenta-se uma configuração desfavorável, onde todas as máquinas fazem parte de um mesmo *cluster*, o *hardware* de rede é homogêneo, não há diferença entre as velocidades de *download* e *upload*, e não há presença de qualquer carga provocada por outras aplicações ou mesmo outras operações de *retrieve* simultâneas. Já no segundo, tornou-se o mesmo ambiente favorável pelo simples fato de limitar a taxa de *upload* das máquinas em $\frac{1}{125}$ vezes a taxa de *download* (de $1Gbps$ para $8Mbps$). As demais configurações continuaram as mesmas.

Com relação ao tamanho dos arquivos, adotou-se três tamanhos: $TA = \{5, 50, 500\}$ *MBytes*. Os dois primeiros tamanhos são considerados pequenos para Grades de dados como o do HEP. Já 500 *MBytes* é considerado um tamanho razoável. Embora fosse desejável testar com arquivos na ordem de *GBytes*, isso não foi possível devido à dificuldade de conseguir as máquinas por um longo tempo necessário para a conclusão de cada caso de teste.

Para o número de fontes, todos os testes utilizam uma única réplica por fragmento, ou seja, $F = n$. O motivo para essa escolha é que, para um mesmo número de fontes F, o custo de gerenciamento é ligeiramente maior quando $R = 1$. Isso se dá devido à necessidade de realizar um *lookup* por fragmento, independentemente do número de réplicas existentes. Testa-se, assim, o pior caso. Escolheu-se os seguintes valores para o número de fragmentos (fontes): $F = n = \{1, 10, 100, 1000\}$.

A Fig. 6.8 mostra o desempenho da operação para o caso onde a rede é desfavorável. Cada ponto corresponde ao tempo médio para fazer o *retrieve* de 30 arquivos, cujo conteúdo foi gerado aleatoriamente. As barras mostram os valores mínimo e máximo mensurados para cada conjunto. Observa-se também que o eixo horizontal do gráfico está em escala logarítmica. Foram utilizados 64 nodos, onde existe apenas 1 nó virtual em cada máquina, ou seja, não utilizou-se nodos virtuais.

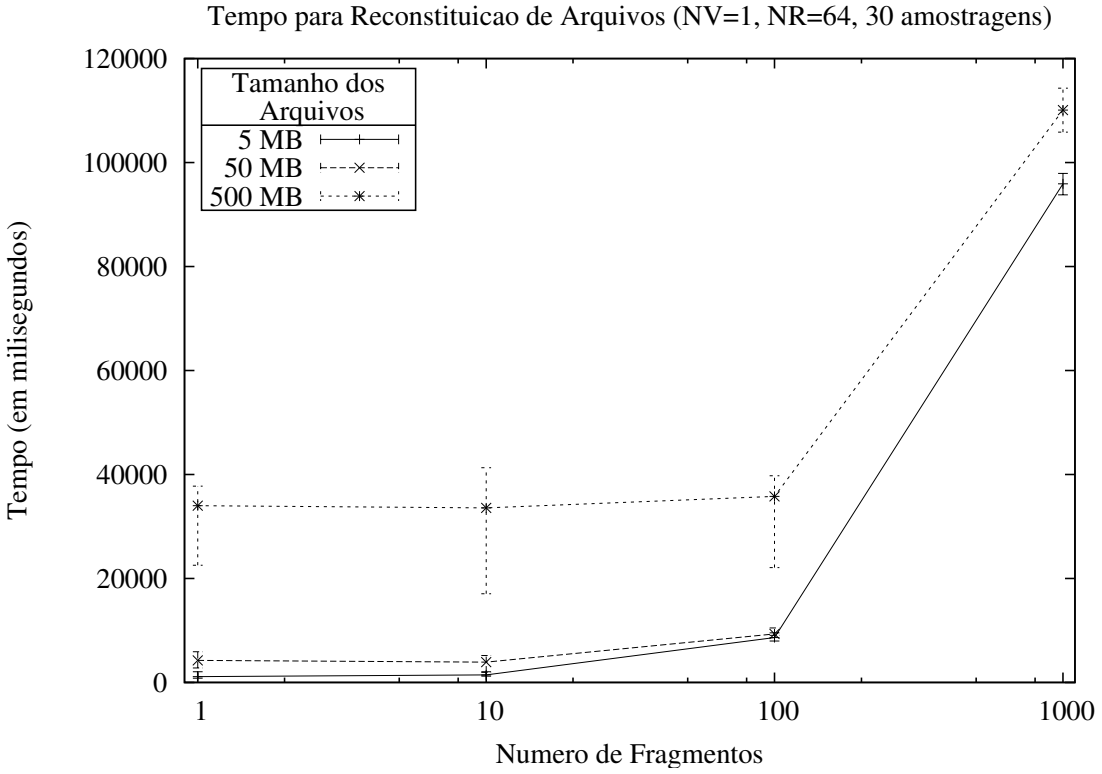

Figura 6.8: Desempenho da operação *retrieve* em uma configuração de rede desfavorável

O gráfico mostra que não há diferença quanto ao desempenho da operação ao aumentar de 1 para 10 fragmentos, ficando em torno de 35 segundos para os arquivos de 500 MB. Quando $F = n = 100$, no entanto, já é possível notar um ligeiro aumento no tempo de execução para os arquivos de 500 MB. Esse aumento se torna evidente para os arquivos menores, sendo que tanto os de 5 quanto os de 50 MB ficaram em aproximadamente 10 segundos. O fato de ambos apresentarem um mesmo tempo dá indícios de que existe um custo fixo para o gerenciamento dessa quantidade de fragmentos. Quando o número de fontes é ainda maior ($F = 1000$), o tempo de execução aumenta consideravelmente para todos os tamanhos de arquivo, ficando em quase 100 segundos para os arquivos pequenos.

Para verificar qual o custo de gerenciamento (G) envolvido em função do número de fontes, a Fig. 6.9 mostra o tempo de execução da operação para arquivos de tamanho desprezível (10 KB). Nesse gráfico, o eixo horizontal foi deixado na escala original para

facilitar a identificação do crescimento do custo, e foram mensurados também outros valores para o número de fontes ($F = n = 1, 10, 50, 100, 250, 500, 750, 1000$). Percebe-se claramente que existe um custo que cresce linearmente em relação ao número de fragmentos, independentemente do tamanho de arquivo utilizado. Verificou-se que esse custo se deve basicamente à grande quantidade de processos externos e conexões do GridFTP gerados para as transferências, exigindo quase 100% de CPU. Para 1000 fragmentos, por exemplo, existiam mais de 100 processos, 4500 conexões TCP e dos cerca de 90 segundos da operação, apenas cerca de 10 segundos foram gastos na etapa de *lookup*. Como são iniciadas uma grande quantidade de conexões do GridFTP, justifica-se o uso intenso de CPU devido à autenticação de cada canal de transferência (procedimento que envolve criptografia de chave pública).

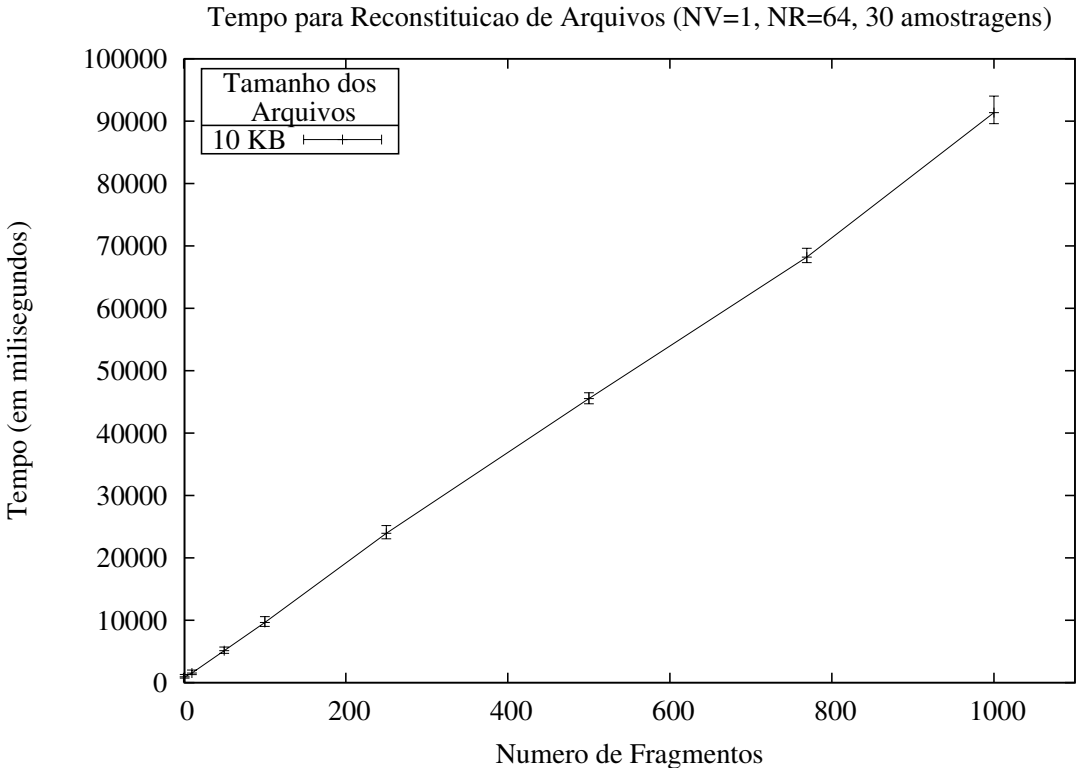

Figura 6.9: Desempenho da operação *retrieve* para arquivos de tamanho desprezível em uma rede com configuração desfavorável.

O resultado explica o crescimento do tempo de execução do gráfico anterior (Fig. 6.8). Esse crescimento só não é significativo para os arquivos de 500 MB quando 100 fragmentos são utilizados pois existe um paralelismo entre as tarefas de gerenciamento (G) e de transferência dos arquivos ($W_{download}$). Quando o tempo para transferir os dados é maior que o custo de gerenciamento ($W_{download} > G$), este último fica mascarado e apresenta impacto desprezível. No entanto, quando o tempo de transferência se torna menor, o desempenho da operação se dá basicamente devido ao custo de gerenciamento.

Conseqüentemente, pode-se concluir pela Fig. 6.8 que, quando o número de fontes é bem dimensionado (o custo de gerenciamento é inferior ao custo de transferência), o sistema obtém os arquivos com desempenho semelhante ao caso sequencial (1 único fragmento), mesmo em uma configuração de rede que não permite melhorar a vazão agregada.

Para comprovar os benefícios do modelo proposto, expõe-se na Fig. 6.10 o mesmo experimento para um caso onde a configuração da rede é favorável. Embora o espaço

em disco ocupado tenha sido o mesmo, foi possível diminuir o tempo de transferência de arquivos de 500 MB de cerca de 560 para até cerca de 60 segundos usando-se apenas 10 fragmentos. O fator de redução do tempo de execução (*speedup* relativo), nesse caso, é de cerca de $9,33$ vezes. Analogamente, o tempo para transferir arquivos de 50 MB caiu de cerca de 56 para até cerca de 7 segundos utilizando-se o mesmo número de fontes ($speedup = 8$). Para alguns arquivos de 5 MB, obteve-se *speedups* de até cerca de 4 vezes com os mesmos 10 fragmentos.

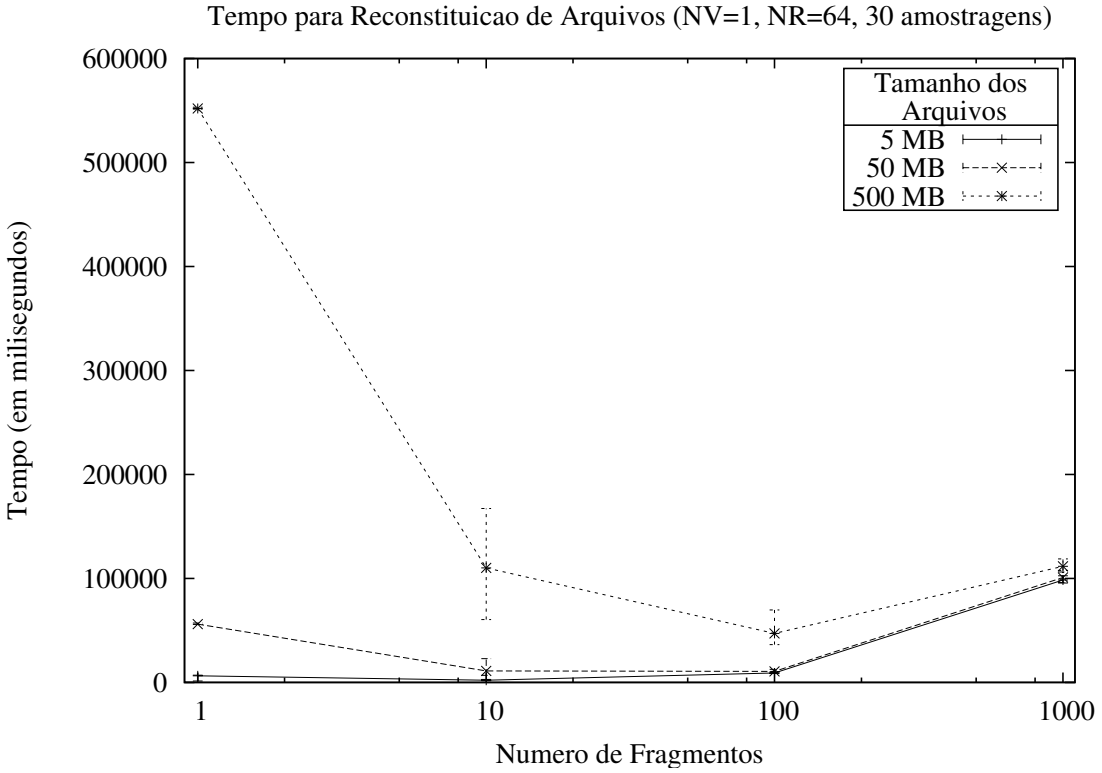

Figura 6.10: Desempenho da operação *retrieve* em uma configuração de rede favorável.

Nessa configuração de rede, o ganho máximo de desempenho se dá em algum valor entre 10 e 100 fragmentos. O caso ideal seria aquele onde cada fragmento é gerenciado por um nó distinto, pois permitiria uma maior vazão aos dados. Entretanto, como o mapeamento dos fragmentos aos nodos pode ser considerado de forma aleatória (obtido como resposta de um *lookup*), a probabilidade de isso acontecer é muito pequena. Para $n = 10$, por exemplo, isso aconteceu em apenas 5 dos 30 arquivos de 500 MB inseridos. Por outro lado, desses mesmos 30, em 3 casos aconteceu de 3 fragmentos serem mapeados para um mesmo nó. Isso explica a grande variabilidade mostrada pelas barras nesse ponto do gráfico. Os demais 22 arquivos ficaram todos concentrados em volta da média. Caso o número de nodos fosse na ordem de milhares ou milhões, poderia-se assumir que todos os fragmentos seriam mapeados para nodos distintos, melhorando assim ainda mais o desempenho da operação.

A exemplo do que aconteceu no experimento onde a rede era desvaforável (Fig. 6.8), quando o sobrecusto de gerenciamento se torna maior que o tempo necessário para transferir os arquivos, o desempenho da operação tende a diminuir proporcionalmente ao número de fontes utilizado. Especificamente no caso onde $F = n = 1000$, observou-se ainda perda significativa de pacotes devido à limitação da taxa de *upload* em cada nó

onde foi solicitado o *retrieve* de um arquivo. Para evitar a retransmissão de pacotes, seria necessário uma versão mais sofisticada do Gerente de Transferências que limitasse o número de fragmentos que poderiam estar sendo transferidos simultaneamente. Outro ajuste importante diz respeito ao tempo de *timeout* para as mensagens. Embora seja possível elevar esse valor no *OverlayWeaver* e no *GridFTP*, a versão do servidor *Web* que recebe as requisições XMLRPC não permite esse ajuste. Em um teste preliminar, a simples estratégia de dividir a operação de *retrieve* em duas fases, onde a primeira apenas localiza seqüencialmente cada fragmento junto à rede P2P e a segunda efetua as transferências em paralelo, diminuiu o número de pacotes perdidos e o tempo para o *retrieve* de arquivos de 5 MB caiu de cerca de 100 para cerca de 90 segundos.

Os resultados comprovam que, independentemente da configuração de rede, o tempo para a conclusão da transferência é dado por:

$$C \;=\; max\left\{G,\; W_{download}\right\}$$

Logo, a estratégia do JavaRMS de usar $n = \frac{s}{K \times v_{max}}$ e de permitir que o número de fontes usado esteja entre n e nR é vantajosa para a transferência de arquivos. Quando a rede é desfavorável, a exigência de usar $F = n = \frac{s}{K \times v_{max}}$ fragmentos não implica em um aumento significativo no tempo para concluir a transferência. Por outro lado, pode-se aumentar significativamente o desempenho quando a rede é, de alguma forma, favorável. O ganho no desempenho quando a rede é favorável também não fica limitado pelo baixo número de fragmentos escolhido, pois os fragmentos podem estar replicados, aumentando-se consideravelmente o número de fontes disponíveis.

7 CONSIDERAÇÕES FINAIS

7.1 Conclusões

Grades de Dados como a do HEP surgem da necessidade de armazenar e processar grandes volumes de dados coletados por experimentos ou gerados por grandes simulações (BUNN; NEWMAN, 2003). Para gerenciar esses dados, tipicamente se formam grandes centros de computação onde os recursos, por serem de alta confiabilidade, desempenho e estabilidade, apresentam um alto custo associado (ADYA et al., 2002). Parte da idéia de formar esses grandes centros surge da necessidade de dar vazão aos dados diretamente coletados junto à fonte dos dados, porém outra parte surge devido à imaturidade e à incapacidade dos serviços de Grade de gerenciar recursos menos capacitados. Portanto, deixa-se de lado a grande quantidade de recursos ociosos localizados em pequenos centros e laboratórios de pesquisa, onde se encontram os usuários que de fato precisam processar e analisar esses dados.

Por outro lado, soluções típicas de gerência de dados para ambientes onde os recursos são menos capacitados (ambientes P2P), na forma como são concebidas, não funcionam ou apresentam um baixo desempenho quando existe a necessidade de movimentar grandes quantidades de dados como parte de suas operações básicas. Ainda que gerenciassem bem o grande volume de dados, existe a necessidade de agregar os recursos computacionais existentes nos grandes centros sem subutilizá-los. Para atingir esse objetivo, se faz necessário lidar com a grande heterogeneidade de recursos resultante desse ambiente misto Grade/P2P. Outras exigências feitas por esse ambiente dizem respeito à segurança dos dados e à necessidade de interoperar com outros serviços de Grade.

JavaRMS foi proposto neste trabalho com o objetivo de atender a essas exigências. A arquitetura foi projetada para minimizar o custo das operações que movimentam grandes quantidades de dados. Ao utilizar uma rede estruturada P2P aliada às técnicas de replicação, nodos virtuais e fragmentação, agregou-se à Grade de Dados a grande quantidade de recursos menos capacitados, aumentando a abundância de recursos de armazenamento disponíveis e diminuindo custos. Assim, possibilitou-se utilizar maiores níveis de replicação de dados com o objetivo de melhorar o desempenho das operações sobre grandes volumes de dados e a disponibilidade desses dados. Além do gerenciamento dos arquivos e de seus atributos básicos, a arquitetura também fornece serviços que distribuem informações relativas aos usuários e ao controle de cotas, ainda que disponibilizando a infra-estrutura necessária para serviços futuros de gerência de metadados.

Para validar o modelo, prototipou-se em Java os componentes principais da arquitetura e adaptou-se serviços de comunicação e de formação da camada lógica P2P para atender, respectivamente, a necessidades de segurança e de funcionalidade. Através de testes realizados junto à Grade francesa Grid5000, avaliou-se as principais operações do

sistema no que dizem respeito ao sobrecusto de operação e escalabilidade. Posteriormente, avaliou-se a distribuição dos dados e o mecanismo de controle de uso de recursos de armazenamento do sistema.

A partir dos resultados obtidos, é possível concluir que a arquitetura permite fornecer os serviços atendendo à escala de nodos desejada, uma vez que o comportamento da operação *lookup* cresce logaritmicamente em relação à quantidade de nodos presentes no sistema. Esse comportamento se dá de forma similar com o uso da técnica de nodos virtuais. Os testes ainda permitem concluir que essa técnica pode ser usada com eficácia para lidar com a grande heterogeneidade dos nodos, pois comprovou-se que a quantidade de dados destinados ao gerenciamento por parte de cada nó é proporcional ao número de nodos virtuais utilizados. Já os experimentos com a principal operação envolvendo a transferência de dados comprovaram que os mecanismos planejados junto à concepção do modelo podem trazer um ganho significativo no desempenho. Também conclui-se, por esses resultados, que é extremamente importante a escolha dos parâmetros de fragmentação e replicação para otimizar o desempenho dessa operação, ainda que atendendo às exigências de disponibilidade e de custo de armazenamento.

Embora o ambiente de execução de testes não seja exatamente o mesmo ambiente alvo (devido a dificuldades de acesso a tal infra-estrutura), e dadas as dificuldades inerentes à explosão combinatória de valores para os parâmetros do sistema, a análise através do número de mensagens trocadas nas operações de localização, aliada a análise de pior caso para as operações de movimentação de dados, nos permitem concluir que o sistema é viável e é capaz de atender aos seus principais objetivos. No entanto, existe ainda muito trabalho a ser realizado, tanto em termos de pesquisa quanto de implementação, para que a arquitetura proposta atinja plenamente o seu potencial para o gerenciamento de dados.

7.2 Resumo de Contribuições

A principal contribuição resultante desta pesquisa é um modelo de gerenciamento de dados para um ambiente misto Grade/P2P. Sua originalidade se dá devido às seguintes características:

- *Considera-se uma grande heterogeneidade de recursos (disco e rede)*: o modelo permite agregar tanto recursos pouco capacitados (*Desktops* em redes com baixa conectividade) quanto recursos de grandes centros computacionais (grandes máquinas paralelas ligadas a sistemas de armazenamento em massa, com alta conectividade), ainda que sem subutilizá-los;

- *Otimiza-se as operações para arquivos grandes*: o modelo contempla estratégias para minimizar o tempo necessário para concluir operações que envolvem transferências de arquivos, cujo desempenho é crítico quando grandes volumes de dados estão envolvidos;

- *Interopera-se com demais serviços da Grade*: o modelo é orientado a serviços e utiliza-se de protocolos abertos e padronizados que também são usados por outros serviços de Grade, permitindo o compartilhamento de componentes que realizam funções em comum e facilitando o seu acesso por parte das aplicações de Grade. O uso de *web services* também permite agregar um maior número de recursos pois está menos sujeito a problemas com *firewalls*;

- *Segurança*: além de utilizar comunicação segura, com todas as entidades (recursos e usuários) sendo identificadas por certificados de Grade e por outros documentos digitais, o modelo fornece mecanismos para proteção contra fraudes. As operações são realizadas de forma confiável mesmo sem a necessidade de atribuir tarefas especias a recursos diferenciados (recursos mais capacitados e/ou confiáveis);

- *Gerencia-se os dados de acordo com critérios escolhidos pelo usuário*: o modelo fornece serviços para a escolha de parâmetros que permitem aos usuários obter um bom compromisso entre disponibilidade, desempenho e custo de armazenamento de acordo com a necessidade do usuário para seus dados.

A lista a seguir apresenta as contribuições individuais alcançadas por esta pesquisa:

- Modelagem de uma arquitetura para a gerência de arquivos, metadados, usuários e recursos baseada em serviços e interoperável com outros componentes de Grade;

- Modelagem de um sistema de controle de cotas distribuído baseado em uma camada de sobreposição P2P (totalmente descentralizado) para um ambiente misto Grade/P2P;

- Modelagem de um sistema de cadastro e gerenciamento de usuários também baseado em uma camada de sobreposição P2P, a ser concebido em um ambiente misto Grade/P2P;

- Modelagem de um sistema de arquivamento também baseado em uma camada de sobreposição P2P, a ser concebido em um ambiente misto Grade/P2P;

- Desenvolvimento de um sistema para a gerência de recursos baseado na capacidade de disco fornecida e que admite o controle e o estabelecimento de políticas de uso por parte de seus administradores;

- Desenvolvimento de estratégias para a transferência de arquivos que permitam obter uma vazão maior para os dados, melhorando o desempenho de operações críticas sobre grandes volumes de dados;

- Desenvolvimento de uma estratégia simples de replicação para distribuição de dados baseado em DHTs;

- Prototipação da arquitetura e dos subsistemas principais, atestando sua viabilidade e disponibilizando-a como uma solução para a gerência dos dados em projetos de Grade como o do HEP;

- Construção de um modelo de escolha de parâmetros de armazenamento de arquivos que permitam ao usuários fazer escolhas de acordo com os recursos a ele disponíveis, ainda que atendendo um bom compromisso entre disponibilidade, desempenho e custo de armazenamento.

7.3 Trabalhos Futuros

Embora este trabalho tenha atingido os principais objetivos estabelecidos inicialmente, ainda é preciso desenvolver diversos pontos relacionados ao modelo proposto para que se

consiga obter uma solução completa para a gerência de dados. Existe também a necessidade de prototipar funcionalidades já previstas pelo modelo ou substituir as versões simplificadas desses componentes que foram utilizadas para atender a algumas exigências. Por fim, identificou-se ainda a necessidade de investigar e avaliar outros parâmetros do sistema relevantes a alguns serviços oferecidos para fins de validação de aspectos do modelo atacados por esses serviços. A lista a seguir descreve os principais pontos passíveis de pesquisa futura.

- *Utilizar IDA para melhorar a disponibilidade dos dados*: em pesquisa publicada recentemente por Williams e colegas (WILLIAMS et al., 2007), comprovou-se que utilizar algoritmos de dispersão da informação (IDA) aliados a técnicas de replicação é vantajoso para o gerenciamento dos dados em ambientes P2P, especialmente no caso de arquivos grandes. A aplicação dessa técnica possibilitaria utilizar maiores níveis de fragmentação ainda que mantendo a mesma disponibilidade para o arquivo a um custo baixo de armazenamento;

- *Testar outros algoritmos P2P para a construção da rede lógica*: embora testes por simulação tenham indicado que o Kademlia resulte numa melhor distribuição para os dados, a orientação do sistema a serviços permite que outros algoritmos para a estruturação da rede P2P possam ser utilizados com facilidade. Essa característica permite inclusive que algoritmos não baseados em DHTs sejam utilizados. Particularmente, existe interesse em verificar como se dá a distribuição dos dados e o desempenho dos mecanismos de localização quando a rede P2P é estruturada por *SkipLists* (HARVEY et al., 2003), uma estrutura de dados que distribui a informação de acordo com a proximidade de rede (diferentemente de DHTs, onde apenas o roteamento das mensagens pode ser otimizado de acordo com a proximidade de rede);

- *Corrigir bugs e implementar funcionalidades no middleware P2P*: existe a necessidade de corrigir diversos *bugs* no *middleware* P2P utilizado para viabilizar a prototipação de outros serviços da arquitetura. No que diz respeito aos mecanismos para o tratamento de erros, o *middleware* ainda deixa muito a desejar. Cogita-se também programar uma nova implementação para fins de otimização, já que a sua característica de *framework* lhe impõe um sobrecusto de gerenciamento de estruturas de dados significativo. Essa mesma característica também dificulta a detecção de *bugs* e a implementação de novos serviços;

- *Fornecer uma solução para comunicação na presença de firewalls e redes privadas*: tanto para a construção da rede estruturada P2P quanto para os serviços de movimentação de dados se faz necessário prover mecanismos para possibilitar a comunicação quando as máquinas se encontram em redes privadas ou atrás de *firewalls*. Com isso, conseguiria-se agregar um maior número de recursos ao sistema. Estuda-se a possibilidade de utilizar técnicas como *Hole Punching* (FORD; SRISURESH; KEGEL, 2005) para lidar com esse problema;

- *Desenvolver os serviços de monitoramento e implementar o modelo de estados*: existe a necessidade de investigar as ferramentas de monitoramento de Grade, tais como o MonaLISA e o Globus MDS, para disponibilizar as funcionalidades de monitoramento necessárias aos demais componentes da arquitetura. Particularmente,

precisa-se implementar junto a esses serviços de monitoramento o modelo de estados necessário para viabilizar os protocolos de manutenção dos dados em ambientes bastante dinâmicos como o P2P;

- *Aprofundar o desenvolvimento dos serviços de gerenciamento de transferências*: por envolver a movimentação de grandes quantidades de dados, os serviços de transferência precisam ser evoluídos para lidar com a reutilização dinâmica de fontes, para distribuir melhor a banda de rede disponível quando diversas requisições de transferência são executadas simultaneamente e para fazer a contenção de uso de recursos de CPU ajustando o número de fragmentos de acordo com a vazão máxima que a rede lhe permite utilizar (mecanismo possivelmente adaptativo). Este tópico já está sendo pesquisado pelo doutorando Marko Petek;

- *Desenvolver os serviços de controle de popularidade dos dados*: o problema gerado devido às diferenças de popularidade dos dados e suas variações exige mecanismos possivelmente adaptativos para que o sistema não tenha seu desempenho degradado. Neste trabalho, sugeriu-se o uso de *caching* e/ou replicação automática, mas não se tratou dos diversos aspectos envolvidos com o uso dessas técnicas;

- *Desenvolver os serviços de gerência de metadados*: apesar de a infra-estrutura de distribuição dos dados já estar consolidada, existe a necessidade de desenvolver os serviços de metadados para atacar problemas que surgem de necessidades para este serviço. Um exemplo é a busca de metadados por intervalos de valores sobre redes P2P estruturadas, problema que tem gerado muita pesquisa em trabalhos relacionados a redes P2P, bancos de dados distribuídos e Grades. Este tópico já está sendo pesquisado pelo doutorando Marko Petek;

- *Desenvolver os serviços orientados ao usuário*: a arquitetura carece ainda de uma modelagem detalhada para os serviços do Sistema de Arquivos Virtual e do Sistema de Visões (tópicos também pesquisados no trabalho do doutorando Marko Petek);

- *Prototipar ferramentes para o controle de recursos locais*: existe a necessidade de implementar ferramentas que possibilitem aos administradores dos recursos controlar o espaço de armazenamento disponibilizado (através do controle de nodos virtuias) e limitar o uso dos recursos de rede por parte do sistema. Ainda se faz necessesário poder disponibilizar ferramentas que permitam a elaboração de políticas mais sofisticadas para o uso desses recursos;

- *Implementar os serviços de criação de arquivos*: embora se tenha feito uma análise detalhada da relação entre disponibilidade, desempenho e custo de armazenamento, chegando-se inclusive em um equacionamento para a escolha do número de fontes, ainda não se prototipou serviços que auxiliem na criação dos arquivos;

- *Prototipar operações de manutenção da disponibilidade dos dados*: o protótipo do JavaRMS ainda carece da implementação de serviços que monitorem a disponibilidade dos arquivos e gerem novas cópias de fragmentos automaticamente a fim de atender à disponibilidade exigida. No momento, apenas a durabilidade dos dados é garantida pelas operações de manutenção;

- *Fazer testes junto a um ambiente típico P2P para verificar o impacto de outros parâmetros*: com o objetivo de verificar melhor a influência de alguns parâmetros

como o número de fragmentos e o fator de replicação para um arquivo, faz-se necessário realizar os testes em um ambiente típico P2P. Ou seja, ambientes onde a latência para a comunicação entre os nodos é maior e os *links* que os interconectam apresentam baixas taxas de transferência. Um exemplo de ambiente como esse é o fornecido pelo PlanetLab;

- *Avaliar a capacidade de armazenamento global do sistema*: conforme o espaço de armazenamento global vai se esgotando, a probabilidade de armazenar novos dados com sucesso diminui. De fato, pode ser vantajoso adotar técnicas como a do PAST (ROWSTRON; DRUSCHEL, 2001a) (uma espécie de ponteiros para os arquivos) para possibilitar o uso com sucesso do pouco espaço ainda disponível. Outro aspecto envolvido com o esgotamento do armazenamento global diz respeito ao comportamento que o sistema deve apresentar caso a disponibilidade desses recursos caia drasticamente (devido ao grande número de falhas ou particionamentos da rede), de forma a impossibilitar a manutenção dos dados já inseridos;

- *Disponibilizar um cliente para uso dos serviços através de uma interface gráfica*: apesar de disponibilizar bibliotecas cliente para as aplicações dos usuários e de disponibilizar ferramentas por linha de comando, alguns dos serviços necessitam interagir melhor com os usuários (ex.: o serviço de criação de arquivos). De fato, ferramentas gráficas seriam de grande valia para que os usuários pudessem controlar melhor seus dados e recursos, ainda que possibilitando um melhor ajuste de parâmetros do sistema;

- *Implementar um protocolo simples para movimentação de dados*: outra necessidade é a de um protocolo simples para a movimentação de dados para os casos onde os nodos envolvidos não possuem um protocolo de movimentação em comum para efetivar a transferência dos dados.

REFERÊNCIAS

ADYA, A. et al. FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment. In: SYMPOSIUM ON OPERATING SYSTEMS DESIGN AND IMPLEMENTATION, 5., 2002, Boston, Massachusetts, USA. **Proceedings...** New York: ACM Press, 2002. p.1–14.

ALLCOCK, W. et al. The Globus Striped GridFTP Framework and Server. In: CONFERENCE ON SUPERCOMPUTING, 2005. **Proceedings...** Washington: IEEE Computer Society, 2005. p.54.

ALLCOCK, W.; FOSTER, I.; MADDURI, R. Reliable Data Transport: A Critical Service for the Grid. In: GLOBAL GRID FORUM, BUILDING SERVICE BASED GRIDS WORKSHOP, 11., 2004, Honolulu, Hawaii, USA. **Proceedings...** [S.l.]: Open Grid Forum, 2004.

ANDERSON, D. P. et al. SETI@home: An Experiment in Public-Resource Computing. **Communications of the ACM**, New York, NY, USA, v.45, n.11, p.56–61, Nov. 2002.

ANDROUTSELLIS-THEOTOKIS, S.; SPINELLIS, D. A Survey of Peer-to-Peer Content Distribution Technologies. **ACM Computing Surveys**, New York, NY, USA, v.36, n.4, p.335–371, Dec. 2004.

APACHE XML-RPC. Disponível em: <http://ws.apache.org/xmlrpc/>. Acesso em: fevereiro 2008.

AZZEDIN, F.; MAHESWARAN, M. Towards Trust-Aware Resource Management in Grid Computing Systems. In: INTERNATIONAL SYMPOSIUM ON CLUSTER COMPUTING AND THE GRID, CCGRID, 2., 2002, Berlin, Germany. **Proceedings...** [S.l.]: IEEE Computer Society, 2002. p.452–452.

BAKER, R.; YU, D.; WLODEK, T. A Model for Grid User Management. In: INTERNATIONAL CONFERENCE ON COMPUTING IN HIGH ENERGY AND NUCLEAR PHYSICS, CHEP, 2003, La Jolla, California, USA. **Proceedings...** [S.l.]: World Scientific Publishing, 2003.

BALAKRISHNAN, H. et al. Looking Up Data in P2P Systems. **Communications of the ACM**, New York, NY, USA, v.46, n.2, p.43–48, Feb. 2003.

BUNN, J. J.; NEWMAN, H. B. Data Intensive Grids for High Energy Physics. In: GRID COMPUTING: MAKING THE GLOBAL INFRASTRUCTURE A REALITY, 2003. **Proceedings...** [S.l.]: John Wiley & Sons, 2003. (Wiley Series in Communications Networking & Distributed Systems).

BUYYA, R.; VAZHKUDAI, S. Compute Power Market: Towards a Market-Oriented Grid. In: INTERNATIONAL SYMPOSIUM ON CLUSTER COMPUTING AND THE GRID, CCGRID, 1., 2001. **Proceedings...** [S.l.]: IEEE Computer Society, 2001. p.574.

BYERS, J.; CONSIDINE, J.; MITZENMACHER, M. Simple Load Balancing for Distributed Hash Tables. In: INTERNATIONAL WORKSHOP ON PEER-TO-PEER SYSTEMS, IPTPS, 2., 2003, Berkeley, CA, USA. **Proceedings...** Berlin/Heidelberg: Springer-Verlag, 2003. p.80–87. (Lecture Notes in Computer Science, v.2735).

CAI, M.; CHERVENAK, A.; FRANK, M. A Peer-to-Peer Replica Location Service Based on Distributed Hash Table. In: CONFERENCE ON SUPERCOMPUTING, 2004. **Proceedings...** [S.l.]: IEEE Computer Society, 2004. p.56.

CAMARGO, R. Y. de; KON, F. Distributed Data Storage for Opportunistic Grids. In: INTERNATIONAL MIDDLEWARE DOCTORAL SYMPOSIUM, MDS, 3., 2006, Melbourne, Australia. **Proceedings...** New York: ACM Press, 2006.

CAMARGO, R. Y. de; KON, F. Design and Implementation of a Middleware for Data Storage in Opportunistic Grids. In: INTERNATIONAL SYMPOSIUM ON CLUSTER COMPUTING AND THE GRID, CCGRID, 7., 2007, Rio de Janeiro, Brazil. **Proceedings...** [S.l.]: IEEE Computer Society, 2007. p.23–30.

CAMERON, D. et al. Replica Management in the European DataGrid Project. **Journal of Grid Computing**, Netherlands, v.2, n.4, p.341–351, Dec. 2004.

CHEN, S. et al. Evaluation and Modeling of Web Services Performance. In: INTERNATIONAL CONFERENCE ON WEB SERVICES, ICWS, 2006, Chicago, IL, USA. **Proceedings...** [S.l.]: IEEE Computer Society, 2006. p.437–444.

CHERVENAK, A. et al. The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets. **Journal of Network and Computer Applications**, Oxford, United Kingdom, v.23, n.3, p.187–200, July 2000.

CHERVENAK, A. et al. Giggle: A Framework for Constructing Scalable Replica Location Services. In: CONFERENCE ON SUPERCOMPUTING, 2002, Baltimore, Maryland, EUA. **Proceedings...** [S.l.]: IEEE Computer Society, 2002. p.1–17.

CHERVENAK, A. L. et al. Performance and Scalability of a Replica Location Service. In: INTERNATIONAL SYMPOSIUM ON HIGH PERFORMANCE AND DISTRIBUTED COMPUTING, HPDC, 13., 2004, Honolulu, Hawaii, USA. **Proceedings...** Washington: IEEE Computer Society, 2004. p.182–191.

CHOON-HOONG, D.; NUTANONG, S.; BUYYA, R. **Peer-to-Peer Content Distribution Networks**. Hershey, PA, USA: Idea Group Publishers, 2005. p.28–65. (Peer to Peer Computing: The Evolution of a Disruptive Technology).

COHEN, B. Incentives Build Robustness in BitTorrent. In: WORKSHOP ON ECONOMICS OF PEER-TO-PEER SYSTEMS, 1., 2003, Berkeley, CA, USA. **Proceedings...** [S.l.: s.n.], 2003. Disponível em: <http://www2.sims.berkeley.edu/research/conferences/p2pecon/papers/s4-cohen.pdf>. Acesso em: fevereiro 2008.

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. **Distributed Systems - Concepts and Design**. 4th ed. Reading, USA: Addison-Wesley, 2005.

CROWCROFT, J. et al. **Peer-to-Peer Technologies**. San Francisco, CA, USA: Morgan Kaufmann, 2004. p.593–622. (The Grid 2: Blueprint for a New Computing Infrastructure).

DABEK, F. et al. Wide-area cooperative storage with CFS. In: SYMPOSIUM ON OPERATING SYSTEMS PRINCIPLES, SOSP, 18., 2001, Banff, Alberta, Canada. **Proceedings...** New York: ACM Press, 2001. p.202–215.

DAVIES, J. **Not-Yet-Commons-SSL**. Disponível em: <http://juliusdavies.ca/commons-ssl>. Acesso em: fevereiro 2008.

DIERKS, T.; RESCORLA, E. **The Transport Layer Security (TLS) Protocol Version 1.1**: RFC 4346. [S.l.]: Internet Engineering Task Force, Network Working Group, 2006. Disponível em: <http://www.ietf.org/rfc/rfc4346.txt>. Acesso em: fevereiro 2008.

DRUSCHEL, P.; ROWSTRON, A. PAST: A large-scale, persistent peer-to-peer storage utility. In: WORKSHOP ON HOT TOPICS IN OPERATING SYSTEMS, HOTOS, 8., 2001, Elmau/Oberbayern, Germany. **Proceedings...** Washington: IEEE Computer Society, 2001. p.75–80.

EBERSPäCHER, J.; SCHOLLMEIER, R. Past and Future. In: STEINMETZ, R.; WEHRLE, K. (Ed.). **Peer-to-Peer Systems and Applications**. Berlin, Germany: Springer-Verlag, 2005. p.17–23. (Lecture Notes in Computer Science, v.3485).

FIELDING, R. et al. **Hypertext Transfer Protocol – HTTP/1.1**: RFC 2616. [S.l.]: Internet Engineering Task Force, Network Working Group, 1999. Disponível em: <http://www.faqs.org/rfcs/rfc2616.html>. Acesso em: fevereiro 2008.

FORD, B.; SRISURESH, P.; KEGEL, D. **Peer-to-Peer Communication Across Network Address Translators**. Disponível em: <http://www.bford.info/pub/net/p2pnat/>. Acesso em: fevereiro 2008.

FOSTER, I. What Is the Grid? A Three Point Checklist. **Grid Today**, San Diego, CA, USA, v.1, n.6, p.22–25, July 2002.

FOSTER, I. et al. A Security Architecture for Computational Grids. In: CONFERENCE ON COMPUTER AND COMMUNICATIONS SECURITY, 5., 1998, San Francisco, CA, USA. **Proceedings...** New York: ACM Press, 1998. p.83–92.

FOSTER, I.; IAMNITCHI, A. On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing. In: INTERNATIONAL WORKSHOP ON PEER-TO-PEER SYSTEMS, IPTPS, 2., 2003, Berkeley, CA, USA. **Proceedings...** Berlin/Heidelberg: Springer-Verlag, 2003. p.118–128. (Lecture Notes in Computer Science, v.2735).

FOSTER, I.; KESSELMAN, C. Globus: A Metacomputing Infrastructure Toolkit. **International Journal of Supercomputer Applications and High Performance Computing**, Cambridge, MA, USA, v.11, n.2, p.115–128, Summer 1997.

FOSTER, I.; KESSELMAN, C. Computational Grids. In: FOSTER, I.; KESSELMAN, C. (Ed.). **The Grid**: blueprint for a new computing infrastructure. San Francisco, CA, USA: Morgan Kaufmann, 1999. p.15–51.

FOSTER, I.; KESSELMAN, C.; TUECKE, S. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. **International Journal of High Performance Computing Applications**, Thausand Oaks, CA, USA, v.15, n.3, p.200–222, 2001.

FOSTER, I.; TUECKE, S. Describing the Elephant: The Different Faces of IT as Service. **ACM Queue**, New York, NY, USA, v.3, n.6, p.26–29, July/Aug. 2005.

GEELS, D.; KUBIATOWICZ, J. Replica Management Should Be A Game. In: SIGOPS EUROPEAN WORKSHOP, 10., 2002, Saint-Emilion, France. **Proceedings...** New York: ACM Press, 2002. p.235–238.

GERMAIN, C. et al. XtremWeb: Building an Experimental Platform for Global Computing. In: INTERNATIONAL WORKSHOP ON GRID COMPUTING, GRID, 1., 2000, Bangalore, India. **Proceedings...** London: Springer-Verlag, 2000. p.91 – 101. (Lecture Notes in Computer Science, v.1971).

GLOBUS. **Globus Toolkit 4.0**: Security. Disponível em: <http://globus.org/toolkit/docs/4.0/security/>. Acesso em: fevereiro 2008.

GNUTELLA. Disponível em: <http://www.gnutella.com>. Acesso em: fevereiro 2008.

GOMES, D. S.; PETEK, M.; GEYER, C. F. R. **Data Distribution in Structured Peer-to-Peer Algorithms Using Replication**. Artigo submetido ao CCGrid 2007.

GOPALAKRISHNAN, V. et al. Adaptive Replication in Peer-to-Peer Systems. In: INTERNATIONAL CONFERENCE ON DISTRIBUTED COMPUTING SYSTEMS, ICDCS, 24., 2004, Tokyo, Japan. **Proceedings...** [S.l.]: IEEE Computer Society, 2004. p.360–369.

GRID'5000. Disponível em: <http://www.grid5000.fr>. Acesso em: fevereiro 2008.

HANUSHEVSKY, A.; TRUNOV, A.; COTTRELL, L. Peer-to-Peer Computing for Secure High Performance Data Copying. In: INTERNATIONAL CONFERENCE ON COMPUTING IN HIGH ENERGY AND NUCLEAR PHYSICS, CHEP, 2001, Beijing, China. **Proceedings...** [S.l.]: World Scientific Publishing, 2001.

HARVEY, N. J. et al. SkipNet: A Scalable Overlay Network with Practical Locality Properties. In: USENIX SYMPOSIUM ON INTERNET TECHNOLOGIES AND SYSTEMS, USITS, 4., 2003, Seattle, WA, USA. **Proceedings...** [S.l.]: USENIX Association, 2003. v.4.

HASAN, R. et al. A Survey of Peer-to-Peer Storage Techniques for Distributed File Systems. In: INTERNATIONAL CONFERENCE ON INFORMATION TECHNOLOGY: CODING AND COMPUTING, ITCC, 2005, Las Vegas, Nevada, USA. **Proceedings...** Washington: IEEE Computer Society, 2005. v.2, p.205–213.

HOSCHEK, W. et al. Data Management in an International Data Grid Project. In: INTERNATIONAL WORKSHOP ON GRID COMPUTING, GRID, 1., 2000, Bangalore, India. **Proceedings...** London: Springer-Verlag, 2000. p.77–90. (Lecture Notes in Computer Science, v.1971).

HUEBSCH, R. et al. Querying the Internet with PIER. In: VERY LARGE DATA BASES CONFERENCE, VLDB, 29., 2003, Berlin, Germany. **Proceedings...** [S.l.]: Morgan Kaufmann, 2003. p.321–332.

ITU. **ITU-T Recommendation X.509**: information technology - open systems interconnection - the directory: public-key and attribute certificate frameworks. Disponível em: <http://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-X.509-200003-I!!PDF-E&type=items>. Acesso em: fevereiro 2008.

JABBER Protocols. Disponível em: <http://www.jabber.org/protocol/>. Acesso em: fevereiro 2008.

KANASKAR, N. V.; TOPALOGLU, U.; BAYRAK, C. Globus Security Model for Grid Environment. **ACM SIGSOFT Software Engineering Notes**, New York, NY, USA, v.30, n.6, p.1–9, Nov. 2005.

KRAWCZYK, H.; BELLARE, M.; CANETTI, R. **HMAC**: Keyed-Hashing for Message Authentication: RFC 2104. [S.l.]: Internet Engineering Task Force, Network Working Group, 1997. Disponível em: <http://www.faqs.org/rfcs/rfc2104.html>. Acesso em: fevereiro 2008.

LAMETER, C.; WATSON, C. **Linux shapecfg Man Page**. Disponível em: <http://www.penguin-soft.com/penguin/man/8/shapecfg.html>. Acesso em: fevereiro 2008.

LEGRAND, I. C. et al. MonALISA: An Agent based, Dynamic Service System to Monitor, Control and Optimize Grid based Applications. In: INTERNATIONAL CONFERENCE ON COMPUTING IN HIGH ENERGY AND NUCLEAR PHYSICS, CHEP, 2004, Interlaken, Switzerland. **Proceedings...** [S.l.]: World Scientific Publishing, 2004.

LHC: The Large Hadron Collider. Disponível em: <http://lhc.web.cern.ch/lhc/>. Acesso em: fevereiro 2008.

LITZKOW, M.; LIVNY, M.; MUTKA, M. Condor - A Hunter of Idle Workstations. In: INTERNATIONAL CONFERENCE OF DISTRIBUTED COMPUTING SYSTEMS, 8., 1988. **Proceedings...** [S.l.: s.n.], 1988. p.104–111.

MAYMOUNKOV, P.; MAZIèRES, D. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In: INTERNATIONAL WORKSHOP ON PEER-TO-PEER SYSTEMS, 1., 2002. **Proceedings...** London: Springer-Verlag, 2002. p.53–65. (Lecture Notes In Computer Science, v.2429).

MILOJICIC, D. S. et al. **Peer-to-Peer Computing**. Palo Alto, CA, USA: Hewlett-Packard Laboratories, 2002. (HPL-2002-57).

MINAR, N.; HEDLUND, M. A Network of Peers: peer-to-peer models through the history of the internet. In: ORAM, A. (Ed.). **Peer-to-Peer**: harnessing the power of disruptive technologies. Sebastopol, CA, USA: O'Reilly & Associates, 2001. p.3–20.

MOORE, R. W.; RAJASEKAR, A.; WAN, M. Data Grids, Digital Libraries, and Persistent Archives: An Integrated Approach to Sharing, Publishing, and Archiving Data. **Proceedings of the IEEE**, Washington, DC, USA, v.93, n.3, p.578–588, Mar. 2005.

NAPSTER Free. Disponível em: <http://www.napster.com>. Acesso em: fevereiro 2008.

NEWBOLD, D. **The Compact Muon Solenoid – Technical Proposal**. Switzerland: CERN, European Laboratory for Particle Physics, 1994. (CERN/LHCC 94-38 and CERN/LHCC-P1).

PETEK, M.; GOMES, D. S.; STEENBERG, C.; GEYER, C. F. R.; DIVERIO, T. A.; SANTORO, A. A Model to Implement a Files and Replicas System in Clarens. In: INTERNATIONAL WORKSHOP ON HIGH-PERFORMANCE DATA MANAGEMENT IN GRID ENVIRONMENTS, HPDGRID, 2006, Rio de Janeiro, RJ, Brazil. **Proceedings...** [Rio de Janeiro: UFRJ], 2006.

PETERSON, L. et al. Experiences Implementing PlanetLab. In: SYMPOSIUM ON OPERATING SYSTEMS DESIGN AND IMPLEMENTATION, OSDI, 7., 2006, Seattle, WA, USA. **Proceedings...** Berkeley: USENIX Association, 2006. p.351–366.

PLANK, J. S. A Tutorial on Reed-Solomon Coding for Fault-Tolerance in RAID-like Systems. **Software – Practice & Experience**, New York, NY, USA, v.27, n.9, p.995–1012, September 1997.

RABIN, M. O. Efficient Dispersal of Information for Security, Load Balancing, and Fault-Tolerance. **Journal of the ACM (JACM)**, New York, NY, USA, v.36, n.2, p.335–348, April 1989.

RENATER. **Site web du GIP RENATER**. Disponível em: <http://www.renater.fr/>. Acesso em: fevereiro 2008.

RENESSE, R. van et al. Heterogeneity-Aware Peer-to-Peer Multicast. In: INTERNATIONAL SYMPOSIUM ON DISTRIBUTED COMPUTING, DISC, 17., 2003, Sorrento, Italy. **Proceedings...** Berlin: Springer-Verlag, 2003.

RHEA, S. et al. Maintenance-Free Global Data Storage. **IEEE Internet Computing**, Piscataway, NJ, USA, v.5, n.5, p.40–49, Sept. 2001.

RHEA, S. et al. Pond: the Oceanstore Prototype. In: USENIX CONFERENCE ON FILE AND STORAGE TECHNOLOGIES, FAST, 2., 2003, San Francisco, CA, USA. **Proceedings...** [S.l.]: USENIX Association, 2003. p.1–14.

RIPEANU, M.; FOSTER, I. A Decentralized, Adaptive Replica Location Mechanism. In: INTERNATIONAL SYMPOSIUM ON HIGH PERFORMANCE DISTRIBUTED COMPUTING, HPDC, 11., 2002. **Proceedings...** Washington: IEEE Computer Society, 2002. p.24.

RODRIGUES, R.; LISKOV, B. High Availability in DHTs: Erasure Coding vs. Replication. In: INTERNATIONAL WORKSHOP ON PEER-TO-PEER SYSTEMS, IPTPS, 4., 2005. **Proceedings...** Berlin:Springer-Verlag, 2005. p.226–239. (Lecture Notes in Computer Science, v.3640).

ROWSTRON, A.; DRUSCHEL, P. Storage Management and Caching in PAST, a Large-Scale, Persistent Peer-to-Peer Storage Utility. **Operating Systems Review**, New York, NY, USA, v.35, n.5, p.188–201, Dec. 2001. Trabalho apresentado no ACM Symposium on Operating Systems Principles, 18., 2001, Alberta, Canada.

ROWSTRON, A.; DRUSCHEL, P. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In: IFIP/ACM INTERNATIONAL CONFERENCE ON DISTRIBUTED SYSTEMS PLATFORMS, MIDDLEWARE, 18., 2001, Heidelberg, Germany. **Proceedings...** Berlin: Springer-Verlag, 2001. p.329–350. (Lecture Notes in Computer Science, v.2218).

SATYANARAYANAN, M. **A Survey of Distributed File Systems**. Pittsburgh, Pennsylvania, USA: Carnegie Mellon University, 1989. (CMU-CS-89-116).

SHIRKY, C. Listening to Napster. In: ORAM, A. (Ed.). **Peer-to-Peer**: harnessing the power of disruptive technologies. Sebastopol, CA, USA: O'Reilly & Associates, 2001. p.21–37.

SHUDO, K. **Overlay Weaver Web Site**. Disponível em: <http://overlayweaver.sourceforge.net>. Acesso em: fevereiro 2008.

SHUDO, K.; TANAKA, Y.; SEKIGUCHI, S. Overlay Weaver: An Overlay Construction Toolkit. **Computer Communications**, Oxford, United Kingdom, v.31, n.2, p.402–412, Feb. 2008.

SIDELL, J. et al. Data Replication in Mariposa. In: INTERNATIONAL CONFERENCE ON DATA ENGINEERING, 12., 1996. **Proceedings...** Los Alamitos: IEEE Computer Society, 1996. p.485–494.

SLAZINSKI, E. D. Views - The 'Other' Database Object. In: ANNUAL INFORMATION SYSTEMS EDUCATION CONFERENCE, ISECON, 18., 2001, Cincinnati, Ohio, USA. **Proceedings...** [S.l.]: AITP Foundation for Information Technology Education, 2001.

STEENBERG, C. et al. The Clarens Grid-enabled Web Services Framework: Services and Implementation. In: INTERNATIONAL CONFERENCE ON COMPUTING IN HIGH ENERGY AND NUCLEAR PHYSICS, CHEP, 2004, Interlaken, Switzerland. **Proceedings...** [S.l.]: World Scientific Publishing, 2004.

STEINMETZ, R.; WEHRLE, K. What Is This "Peer-to-Peer" About? In: STEINMETZ, R.; WEHRLE, K. (Ed.). **Peer-to-Peer Systems and Applications**. Berlin, Germany: Springer-Verlag, 2005. p.9–16. (Lecture Notes in Computer Science, v.3485).

STOICA, I. et al. Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications. In: SPECIAL INTEREST GROUP ON DATA COMMUNICATIONS, SIGCOMM, 2001, San Diego, CA, USA. **Proceedings...** [S.l.: s.n.], 2001. p.149–160.

TANENBAUM, A. S. **Modern Operating Systems**. 2nd ed. Upper Saddle River, New Jersey, USA: Prentice Hall PTR, 2001.

VAZHKUDAI, S. et al. FreeLoader: Scavenging Desktop Storage Resources for Scientific Data. In: CONFERENCE ON SUPERCOMPUTING, SC, 2005, Seattle, Washington, USA. **Proceedings...** New York: ACM Press, 2005. p.56–56.

VENUGOPAL, S.; BUYYA, R.; RAMAMOHANARAO, K. A Taxonomy of Data Grids for Distributed Data Sharing, Management, and Processing. **ACM Computing Surveys**, New York, NY, USA, v.38, n.1, p.1–53, March 2006.

WEATHERSPOON, H.; KUBIATOWICZ, J. Erasure Coding vs. Replication: A Quantitative Comparison. In: INTERNATIONAL WORKSHOP ON PEER-TO-PEER SYSTEMS, IPTPS, 1., 2002. **Proceedings...** London: Springer-Verlag, 2002. p.328–338. (Lecture Notes in Computer Science, v.2429).

WILLIAMS, C. et al. Redundancy Management for P2P Storage. In: INTERNATIONAL SYMPOSIUM ON CLUSTER COMPUTING AND THE GRID, CCGRID, 7., 2007, Rio de Janeiro, RJ, Brasil. **Proceedings...** [S.l.]: IEEE Computer Society, 2007. p.15–22.

WINER, D. **XML-RPC Specification**. [S.l.]: UserLand Software, 1999. Disponível em: <http://www.xmlrpc.com/spec>. Acesso em: fevereiro 2008.

ZHANG, X.; SCHOPF, J. Performance Analysis of the Globus Toolkit Monitoring and Discovery Service, MDS2. In: IPCCC INTERNATIONAL WORKSHOP ON MIDDLEWARE PERFORMANCE, IWMP, 2004. **Proceedings...** Washington: IEEE Computer Society, 2004.

APÊNDICE A KADEMLIA

O algoritmo Kademlia adota como métrica de distância a função XOR. Suas propriedades são a simetria e a unidirecionalidade. A simetria permite que um nó atualize as suas informações de roteamento baseado nas mensagens recebidas de outros nodos, pois as mensagens sempre vem de possíveis candidatos a preencher as entradas da tabela de roteamento. A unidirecionalidade assegura que as consultas por uma chave convergem sempre para o mesmo caminho, independentemente do nó que as originou. Dessa forma, pode-se aplicar esquemas de *caching* com boas taxas de acerto. No entanto, XOR não é uma métrica Euclidiana.

Cada nó no Kademlia recebe um identificador de m bits (onde tipicamente $m = 160$) e arnazena uma tabela de roteamento com informações de outros nodos. Essa tabela é composta de m listas com k entradas cada uma. Essas listas recebem o nome de *k-buckets*. Os nodos que se encontram no i-ésimo *k-bucket* ($0 \leq i < m$) estão a uma distância entre 2^i e 2^{i+1} do nó em questão.

Para localizar um item, basta casar 1 bit da chave a cada nó consultado. Diz-se que o conjunto de nodos consultados para encontrar a chave constituem um **caminho** do nó que procura a chave até o nó por ela responsável. O resultado da operação de localização é o conjunto dos k nodos com identificadores mais próximos à chave que foram encontrados, ou seja, o conjunto dos k nodos por ela responsáveis. O algoritmo ainda pode ser alterado para armazenar mais informações de roteamento de forma a casar b bits da chave a cada nó consultado. Como podem existir até $N = 2^m$ nodos, esse mecanismo de roteamento se dá em $logN$ passos. Ou seja, utilizam-se $\mathcal{O}(logN)$ mensagens.

Uma vez que existem $2^{i+1} - 2^i$ possíveis candidatos para ocupar as k entradas do i-ésimo *k-bucket*, o algoritmo precisa adotar uma estratégia para determinar quais nodos vão ser colocados no *k-bucket*. Tal estratégia consiste em consultar o nó menos recentemente observado que está no *k-bucket* e, caso ele não responda, o nó candidato é inserido no seu lugar. Esse procedimento só é realizado quando o *k-bucket* já está cheio. Caso contrário, o nó candidato é diretamente incluído.

O fato de manter até k entradas para cada bit do identificador permite que, durante um *lookup*, escolham-se o nodos que estão mais próximos na rede física (usando-se, por exemplo, o valor da latência). Também é possível consultar mais de um desses nodos em paralelo de forma a minimizar o tempo de resposta.