Sergio Raymundo Loest

Um Sistema de Backup Cooperativo Tolerante a Intrusões

Dissertação submetida ao Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica do Paraná como requisito parcial para a obtenção do título de Mestre em Informática.

Curitiba PR
Junho de 2009

Sergio Raymundo Loest

Um Sistema de Backup Cooperativo Tolerante a Intrusões

Dissertação submetida ao Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica do Paraná como requisito parcial para a obtenção do título de Mestre em Informática.

Área de concentração: *Ciência da Computação*

Orientador: Carlos Alberto Maziero
Co-orientador: Lau Cheuk Lung

Curitiba PR
Junho de 2009

Loest, Sergio Raymundo.

Um Sistema de Backup Cooperativo Tolerante a Intrusões. Curitiba, 2009. 86p.

Dissertação - Pontifícia Universidade Católica do Paraná. Programa de Pós-Graduação em Informática.

1. sistemas de backup 2. quóruns bizantinos 3. tolerância a intrusões 4. peer-to-peer. I. Pontifícia Universidade Católica do Paraná. Centro de Ciências Exatas e de Tecnologia. Programa de Pós-Graduação em Informática II-t.

Esta folha deve ser substituída pela ata de defesa devidamente assinada, que será fornecida pela secretaria do programa após a defesa.

A Minha esposa Valéria e meu filho Rafael que tanto me apoiaram e incentivaram neste projeto.

Agradecimentos

Agradeço aos professores e a Pontifícia Universidade Católica do Paraná que contribuíram de alguma forma para o bom andamento deste trabalho.

Agradeço, especialmente, aos meus orientadores Professor Doutor Carlos Alberto Maziero e ao Professor Doutor Lau Cheuk Lung que tanto contribuiram e me apoiaram durtante todo este trabalho com seus questionamentos, motivação e correções.

Por último, mas não menos importante, agradeço também a Siemens que me proporcionou esta oportunidade de crescimento.

Resumo

O armazenamento confiável de grandes volumes de dados sempre foi um problema para as corporações. Eficiência, disponibilidade, integridade e confidencialidade dos dados são algumas das características que um sistema de *backup* deve oferecer. Ao mesmo tempo, essas corporações possuem muitos computadores com espaço livre nos discos locais e boa conectividade de rede. Neste trabalho é proposto um sistema de *backup* cooperativo tolerante a intrusões, que se utiliza dos recursos ociosos dos computadores para prover um serviço eficiente e seguro de armazenamento. O sistema de *backup* proposto usa de forma eficiente os recursos da rede e de armazenamento através de processos de verificação, técnicas de compressão e criptografia. Além disso, o uso de protocolos de quóruns Bizantinos permite ao sistema também tolerar comportamentos inesperados. Os experimentos realizados para avaliar a proposta demonstram sua viabilidade.

Palavras-chave: sistemas de backup, quóruns bizantinos, tolerância a intrusões.

Abstract

Reliable storage of large amounts of data was always a problem for the enterprise world. Availability, efficiency, data integrity, and confidentiality are some of the desirable features a data backup system should provide. At the same time, corporate computers offer spare disk space and unused networking resources. In this document, we propose an intrusion tolerant cooperative backup system that provides a reliable collaborative backup resource by leveraging these independent, distributed resources. This system makes efficient use of network and storage resources through use of compression techniques, encryption, and efficient verification processes. It also implements a protocol to tolerate Byzantine behaviors when nodes arbitrarily deviate from their specifications. Experiments performed to evaluate the proposal showed its viability.

Keywords: backup systems, byzantine quorum, intrusion tolerance.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Símbolos

K	chave criptográfica utilizada no esquema de compartilhamento de segredos de Shamir (SHA)
κ	(kappa) um certo número de partes de K
λ	(lambda) representa o número de partes nas quais a chave K é subdividida
k	chave de criptografia simétrica
f	limite máximo de nós faltosos
p_i	nó i do sistema de *backup* cooperativo
x	Um arquivo a ser armazenado no sistema de *backup* cooperativo
$\mathbb{C}$	conjunto de nós com resposta correta
$\mathbb{E}$	conjunto de nós com erro
$\mathbb{L}_i$	*conjunto-folha* do nó p_i fornecido pelo substrato *Pastry*
$\mathbb{P}$	conjunto com N nós $p_1 \dots p_N$
$\mathbb{Q}^*$	Um quórum qualquer do sistema de quóruns
$\mathbb{R}$	conjunto de nós que responderam
$\mathbb{S}$	sistema de quóruns bizantinos
$\mathbb{T}$	conjunto de nós com *time-out*

Lista de Abreviações

API	Interface de Programação de Aplicações
BQS	Byzantine Quorum System (sistemas de quóruns bizantinos)
CPU	Central Processing Unit
DHT	Distributed Hash Table
ID ou NodeID	Identificador do nó
IP	Internet Protocol
IT	Information Technology
LAN	Local Area Network
p2p	peer-to-peer
WAN	Wide Area Network

Capítulo 1

Introdução

1.1 Motivação

A capacidade de armazenamento de dados digitais segue a lei de Moore, crescendo a uma taxa anual de 100% [Growchowski, 1988]. Ao mesmo tempo, essa mesma capacidade de armazenamento crescente é consumida pela produção de novos dados. Logo, a necessidade de espaço para armazenar dados cresce na mesma proporção da capacidade de armazenamento de dados [Killijian and Courtes, 2006]. Como conseqüência direta desta relação, a necessidade de espaço para armazenamento de cópias de segurança (*backup*) aumenta na mesma proporção do aumento da capacidade de armazenamento de dados digitais.

As corporações também seguem a lei Moore, ou seja, cada vez mais estas necessitam de mais espaço para armazenamento de *backup*. As corporações possuem suas próprias redes internas de computadores (*intranets*) para dar suporte às suas operações. A gerência dos recursos computacionais dessas redes internas deve garantir a integridade, disponibilidade e confidencialidade de suas informações. Uma das ferramentas utilizadas para garantir estes requisitos é a cópia de segurança (*backup*). O *backup* é a cópia dos dados de um dispositivo para outro, visando sua recuperação futura em caso de perdas. Ele é útil em duas situações: para restabelecer um computador a um estado operacional anterior a um desastre e para recuperar alguns arquivos após a sua remoção acidental ou corrupção. Dependendo da quantidade de dados críticos que devem ser preservados, o espaço necessário para *backup* pode chegar facilmente a centenas de terabytes. Uma corporação pode ter uma grande capacidade de armazenamento ociosa, pois em média 50% dos discos rígidos de cada computador de sua rede interna estão livres [Douceur and Bolosky, 1999, Cipar et al., 2007]. Além disso, existe também uma grande capacidade de processamento ociosa e disponível nessa mesma rede, pois, a atividade mais comum de um computador pessoal é esperar por entradas do usuário.

1.2 Objetivo

O objetivo deste trabalho é construir um sistema de *backup* que faça uso das capacidades de armazenamento e processamento ociosas presentes nas corporações. Assim, a capacidade de armazenamento da rede interna da corporação poderá ser utilizada com mais eficiência, aproveitando a sua capacidade ociosa de armazenamento para a guarda de cópias de segurança

(*backup*). Da mesma forma, a capacidade ociosa de processamento pode ser utilizada para a implementação de um serviço de *backup* cooperativo.

O objetivo dos sistemas de *backup* é tolerar faltas em dispositivos de armazenamento, independentemente de sua localização (próximo ou distante) ou de sua estrutura (centralizada ou compartilhada) [Killijian and Courtes, 2006]. Eles devem garantir a confidencialidade, integridade e consistência dos dados armazenados. Sistemas de *backup* cooperativos [Lillibridge et al., 2003] usam a tecnologia *peer-to-peer* para construir um ambiente de *backup* distribuído não-hierárquico, no qual cada nó do ambiente usa o espaço ocioso em seu disco local para, armazenar dados de outros nós de forma cooperativa. Além disso, para um sistema de *backup* ser útil, ele deve estar disponível, isto é, ser resiliente a faltas, especialmente no que concerne ao comportamento malicioso (bizantino) dos nós faltosos do sistema. Em especial, um sistema *peer-to-peer* está muito mais sujeito ao comportamento malicioso dos nós do sistema do que um sistema centralizado devido a sua natureza distribuída. Por este motivo, o uso de técnicas de tolerância a faltas, tais como o uso de sistemas de quóruns bizantinos, é fundamental.

Os sistemas de quóruns são ferramentas muito úteis na construção de serviços de dados replicados com alta disponibilidade e eficiência. Um sistema de quóruns bizantinos (BQS – *Byzantine Quorum System*) [Malkhi and Reiter, 1997] é definido como um sistema distribuído de armazenamento de dados replicados com garantias de consistência e disponibilidade, mesmo na ocorrência de faltas bizantinas em algumas de suas réplicas [Dantas et al., 2007]. Um sistema de quóruns é um conjunto de conjuntos de nós (chamados quóruns) e cada par de quóruns se interseccionam. A intersecção entre os quóruns é projetada de forma a apresentar uma propriedade específica que garante que cada leitura acessa sempre os valores mais recentemente escritos em uma dada variável compartilhada [Malkhiy et al., 2000]. O princípio por trás do uso dos quóruns em serviços de dados distribuídos é que, se uma variável compartilhada é armazenada em um conjunto de servidores, as operações de escrita e leitura nessa variável precisam ser executadas somente em um quórum desses servidores e não em todo o sistema de quóruns, assim aumentando o desempenho do sistema. Os protocolos usados em sistemas de quóruns têm término garantido, por isso eles dispensam a implementação de protocolos de acordo para garantir a consistência entre réplicas.

Por fim, o sistema BackupIT, proposto nesta dissertação, foi concebido tendo em vista o seu uso em *Intranets* de corporações, pois estas fornecem um ambiente para o qual não é necessária a gerência de credibilidade dos nós participantes do sistema pois, todos os nós são conhecidos, possuem, em princípio, objetivos comuns e estão sob a jurisdição de um único domínio administrativo. Além disso, a questão da entrada e saída dos nós no sistema (*churn*) fica reduzida, pois nestes ambientes os computadores ou ficam ligados 24 horas por dia ou são ligados no início do expediente e desligados ao seu final dependendo das políticas de gerência de energia e disponibilidade do sistema implementadas pela corporação. Neste caso os computadores serão acessíveis pelo menos no período do expediente de funcionamento da corporação. Para que o sistema proposto tolere intrusões foram implementados protocolos de sistemas de quóruns bizantinos (BQS – *Byzantine Quorum System*) que são utilizados por todos os nós participantes do sistema. Desta forma, as vantagens dos sistemas de quóruns são agregadas às dos sistemas de *backup* cooperativos para fornecer tolerância a intrusão e assim apresentar o primeiro sistema de *backup* cooperativo tolerante a intrusões que faz uso de sistemas de quóruns bizantinos.

1.3 Organização do documento

Esta dissertação de mestrado é organizada como segue: no próximo capítulo (capítulo 2) é apresentada uma visão geral sobre as redes *peer-to-peer* e serviços cooperativos tolerantes a faltas. O capítulo 3 mostra uma visão geral sobre sistemas de *backup* cooperativos e apresenta os principais trabalhos realizados nessa área. No capítulo 4 são apresentados os conceitos de sistemas de quóruns bizantinos e a arquitetura do sistema proposto. No capítulo 5 a implementação do protótipo e os resultados das experimentações são apresentados. No último capítulo (capítulo 6) são apresentas as conclusões e discutidas as possibilidades de trabalhos futuros.

Capítulo 2

Sistemas Peer-to-Peer

2.1 Introdução

A tecnologia *peer-to-peer (p2p)* está dentre as tecnologias em computação que crescem mais rapidamente e isto pode ser percebido pela a crescente utilização de aplicações *p2p* na Internet hoje em dia. Estudos recentes apontaram a dominante participação do tráfego *p2p* na Internet chegando a ocupar entre 40 e 60% da capacidade dos *backbones*, como apresentado em [TorrentFreak, 2008] e [Sandvine, 2008].

Sistemas *p2p* são sistemas distribuídos consistindo de nós interconectados, capazes de se auto-organizar em topologias de rede com o propósito de compartilhar grandes quantidades de recursos tais como conteúdo (arquivos), ciclos de CPU, espaço de armazenamento e capacidade de transmissão. Inicialmente esta tecnologia foi desenvolvida para permitir o compartilhamento de dados em redes não estruturadas, no entanto as propostas mais recentes contemplam o suporte ao compartilhamento de dados em redes estruturadas [Sung et al., 2006]. Nos sistemas *p2p*, os pares (nós) se comunicam através de topologias de redes auto-organizadas e tolerantes a faltas, que operam como uma rede sobreposta sobre a rede física. Estes sistemas são amplamente distribuídos, altamente voláteis porém podem estar sujeitos a intrusões por parte de nós maliciosos.

A distribuição de conteúdo, uma proeminente área de aplicação de sistemas *p2p*, é baseada em sistemas e infra-estruturas projetadas para compartilhar mídia digital e outros tipos de dados entre usuários. Os sistemas *p2p* para distribuição de conteúdo variam desde aplicações relativamente simples para compartilhamento de arquivos até sistemas mais sofisticados que criam uma infra-estrutura para armazenamento distribuído para publicação, organização, indexação, busca, atualização e recuperação segura e eficiente.

As redes de computadores *p2p* como o Gnutella [Kirk, 2003], BitTorrent [Pouwelse et al., 2004], KaZaA [Leibowitz et al., 2003] e outras são sistemas distribuídos ad-hoc. Estas redes baseiam-se principalmente na capacidade de processamento e na largura de banda dos participantes da rede ao invés de se concentrar em um pequeno número de servidores. Estas redes são utilizadas para compartilhamento direto de recursos computacionais tais como capacidade de processamento (CPU), memória e dados sem intermediação de órgãos centralizados. Assim, quanto maior a quantidade de nós na rede, maior será a capacidade do sistema.

Em um sistema *p2p* puro não existe a noção de cliente ou servidor, todos os pares fazem o papel de servidor e cliente com outros nós da rede. A natureza distribuída de uma rede

p2p garante a sua robustez no caso de falhas devido à replicação dos dados através de múltiplos pares e, em sistemas *p2p* puros, pela capacidade dos pares de localizar dados sem necessitar de um servidor de indexação centralizado, evitando assim a existência de pontos de falha únicos no sistema.

As principais características de um sistema *p2p* são: a escalabilidade, a resistência à censura e ao controle centralizado, a auto-organização em presença de uma população de nós altamente variável, a tolerância a falhas de rede e de nós, a inexistência de um servidor centralizado e conseqüentemente do custo da sua administração.

Será apresentado no restante deste capítulo a definição adotada para este trabalho para redes *p2p*, sua classificação e suas características principais. Serão apresentados também alguns conceitos sobre tabelas de *hash* distribuídas (DTH), que também são utilizadas em infra-estruturas para sistemas *p2p* e por último os sistemas de quóruns bizantinos, que fornecem protocolos para tolerância a intrusões além de desempenho, escalabilidade e disponibilidade ao sistema *p2p*.

2.2 Definição de rede *p2p*

Existe uma grande quantidade de definições para redes *p2p*, dependendo da abrangência que é dada ao termo. A definição mais rigorosa de sistemas *p2p* puros refere-se a sistemas totalmente distribuídos nos quais todos os nós têm as mesmas funcionalidades e tarefas. Porém existem muitas arquiteturas diferentes para estes sistemas, algumas utilizam supernós outras utilizam servidores centralizados para executar funções auxiliares. Muitos sistemas são chamados de *p2p* pela forma como são percebidos pelos usuários e não pela sua arquitetura. Resumindo, não se chegou a um consenso sobre a definição de sistemas *p2p*. Desta forma, para este trabalho foi utilizada a definição apresentada em [Androutsellis-Theotokis and Spinellis, 2004]:

> "Sistemas *peer-to-peer* são sistemas distribuídos constituídos de nós interconectados capazes de se organizar em topologias de redes com a intenção de compartilhar recursos como conteúdo, ciclos de CPU, armazenamento e largura de banda, capazes de se adaptar a faltas e acomodar uma população de nós transiente enquanto mantém uma conectividade e desempenho aceitável, sem requerer a intermediação ou o suporte de uma autoridade ou servidor centralizado e global."

Uma rede pode ser dita *peer-to-peer*, mesmo que algumas das funções de controle da rede estejam localizadas em um servidor central (ponto de falha). Esta definição tem por objetivo ser bastante abrangente, e assim, englobar as várias classificações de sistemas *p2p* que serão discutidas mais adiante.

2.3 Redes sobrepostas

Uma rede sobreposta (*overlay network*) é uma rede de computadores construída sobre outra rede, ou seja, é uma rede virtual criada sobre uma rede existente. Por exemplo, muitas redes *p2p* são redes sobrepostas porque elas operam sobre a Internet, com infra-estrutura IP [Kamienski et al., 2005]. Esta rede cria uma arquitetura com nível mais alto de abstração, de modo a poder solucionar vários problemas que, em geral, são difíceis de ser tratados no nível

dos roteadores da rede subjacente. Os nós da rede sobreposta podem ser vistos como estando conectados através de conexões virtuais ou lógicas (figura 2.1). Cada uma destas conexões corresponde a um caminho através de um número de conexões físicas na rede subjacente. As redes sobrepostas podem oferecer várias funcionalidades aos seus componentes, tais como, arquitetura de roteamento WAN robusta, busca eficiente de dados, seleção de pares próximos, armazenamento redundante, durabilidade dos dados, nomenclatura hierárquica, autenticação, anonimato, escalabilidade e tolerância a faltas [Lua et al., 2005].

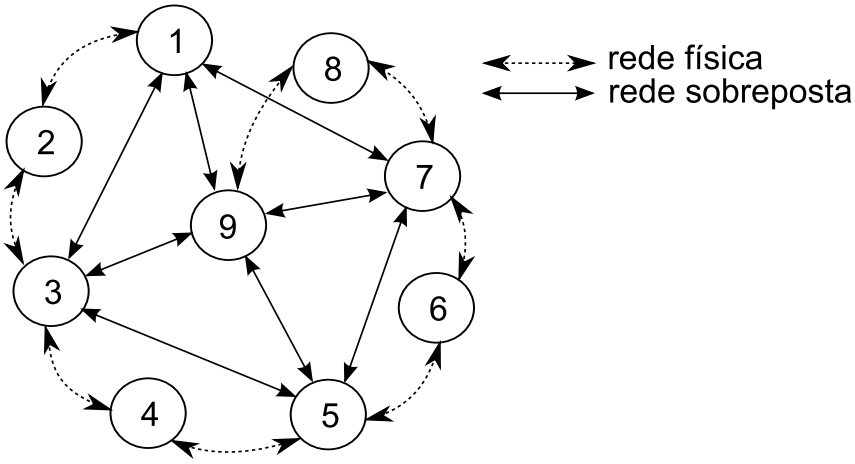

Figura 2.1: Rede *p2p* sobreposta (*Overlay*). Adaptado de [Kamienski et al., 2005]

A figura 2.2 mostra um resumo da arquitetura de uma rede *p2p* sobreposta ilustrando os seus componentes estruturais. A camada de Comunicação de Rede descreve as características da rede de computadores conectada através da Internet ou outras redes *ad-hoc*[1]. A camada de Gerência de Nós Sobrepostos gerencia os pares, o que inclui detecção de pares e algoritmos de roteamento para otimização do sistema. A camada de Gerência de Funções lida com segurança, confiabilidade, resiliência a faltas e aspectos de disponibilidade de recursos agregados para a manutenção da robustez dos sistemas *p2p*. A camada de Serviços Específicos dá suporte à infra-estrutura *p2p* subjacente e aos componentes específicos das aplicações através de agendamento paralelo e intensivo de tarefas, gerência de conteúdo e arquivos. A camada de Aplicação está envolvida com ferramentas, aplicações e serviços que são implementados com funcionalidades específicas sobre a infra-estrutura *p2p* sobreposta.

A estrutura, topologia e o grau de centralização da rede *p2p* sobreposta, os mecanismos de localização e roteamento que ela emprega são importantíssimos para a operação do sistema na medida em que afetam a sua tolerância a faltas, capacidade de se automanter, adaptabilidade a falhas, desempenho, escalabilidade e segurança.

2.4 Classificação de redes *peer-to-peer*

As redes *peer-to-peer* podem ser classificadas quanto ao seu grau de centralização ou quanto a sua estrutura como veremos a seguir.

[1]Uma rede *ad-hoc* é uma rede que foi formada espontaneamente conforme os equipamentos se conectam a ela. Em Latin, *ad-hoc* significa literalmente "com este objetivo". Geralmente significa uma solução designada para um problema ou tarefa específicos, que não pode ser aplicada em outros casos. Normalmente indica uma situação provisória ou improvisada.

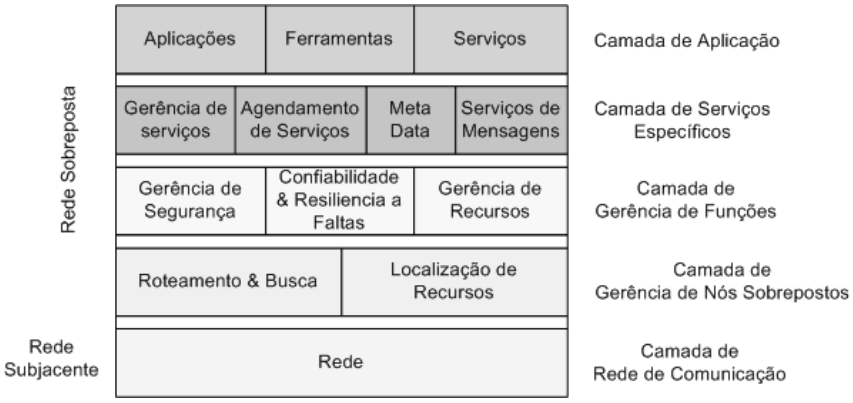

Figura 2.2: Arquitetura de uma rede *p2p* sobreposta. Adaptado de [Lua et al., 2005]

2.4.1 Graus de centralização das redes *peer-to-peer*

Na prática são encontrados sistemas com vários graus de centralização, especialmente a arquitetura descentralizada pura, arquitetura parcialmente descentralizada e a arquitetura descentralizada híbrida, descritas a seguir:

Arquitetura descentralizada pura: Na arquitetura descentralizada pura todos os nós da rede executam exatamente as mesmas tarefas, agindo como servidores e clientes, e sem uma coordenação centralizada para as suas atividades. Os nós são também chamados "*servents*" (SERVidores + cliENTES).

O Gnutella é um exemplo deste tipo de arquitetura. Ele monta uma rede virtual sobreposta com seus próprios mecanismos de busca, permitindo aos seus usuários compartilhar arquivos entre si. Ele usa o IP como seu serviço de rede subjacente e a comunicação entre seus *servents* é especificada como uma forma de protocolo da camada de aplicação.

A arquitetura original do Gnutella utilizava o mecanismo de *broadcast* para distribuir suas mensagens. Cada nó desta rede repassa as mensagens recebidas para todos os seus vizinhos. A mensagem de resposta é roteada de volta pelo caminho oposto através do qual a mensagem original chegou. Para limitar a inundação destas mensagens pela rede cada cabeçalho de mensagem contém um campo *time-to-live* (TTL) que a cada *hop* é decrementado, quando chega a zero a mensagem é descartada.

Arquitetura parcialmente descentralizada: A arquitetura parcialmente descentralizada é semelhante aos sistemas descentralizados puros. No entanto, alguns de seus nós têm um papel mais importante, servindo de indexadores, *cache* de arquivos e *proxy* para as requisições de busca para os pares locais. Todas as solicitações de busca são direcionadas inicialmente a esses nós, por isso denominados "supernós". A forma como estes supernós têm seu papel definido pela rede varia de sistema para sistema. É importante salientar que estes supernós são definidos dinamicamente e, se eles vierem a falhar, a rede irá substituí-los automaticamente. Kazaa e Edutella são exemplos de arquiteturas parcialmente descentralizadas.

As maiores vantagens dos sistemas parcialmente centralizados são:

- O tempo de localização é reduzido em comparação com os sistemas puramente descentralizados.

- Não há ponto de falha único, já que nenhum serviço é afetado pela falha de alguns de seus nós ou supernós, que são substituídos assim que falham.
- A heterogeneidade inerente das redes *p2p* é explorada. Em uma rede descentralizada pura, todos os nós são igualmente carregados em relação à sua capacidade de fornecer ciclos de CPU, banda de comunicação ou capacidade de armazenamento. Em sistemas parcialmente centralizados, no entanto, os supernós irão assumir uma grande porção da carga da rede, enquanto que os outros nós receberão uma carga bastante leve em comparação.

Arquitetura descentralizada híbrida: Nestes sistemas há um servidor central facilitando a interação entre os pares através da manutenção de diretórios de metadados, com a descrição dos arquivos compartilhados armazenados pelos pares. Estes servidores centrais facilitam a interação através da execução de buscas e identificação dos nós que armazenam os arquivos procurados.

A figura 2.3 a seguir ilustra a arquitetura típica de um sistema *p2p* híbrido descentralizado. Cada computador cliente armazena conteúdo (arquivos) que é compartilhado com o resto da rede. Todos os clientes se conectam com um servidor central de diretórios que mantém uma tabela de informações de conexão dos usuários registrados (endereço IP, banda da conexão, etc.) e uma tabela listando os arquivos que cada usuário guarda e compartilha com a rede, juntamente com as descrições dos arquivos (metadados).

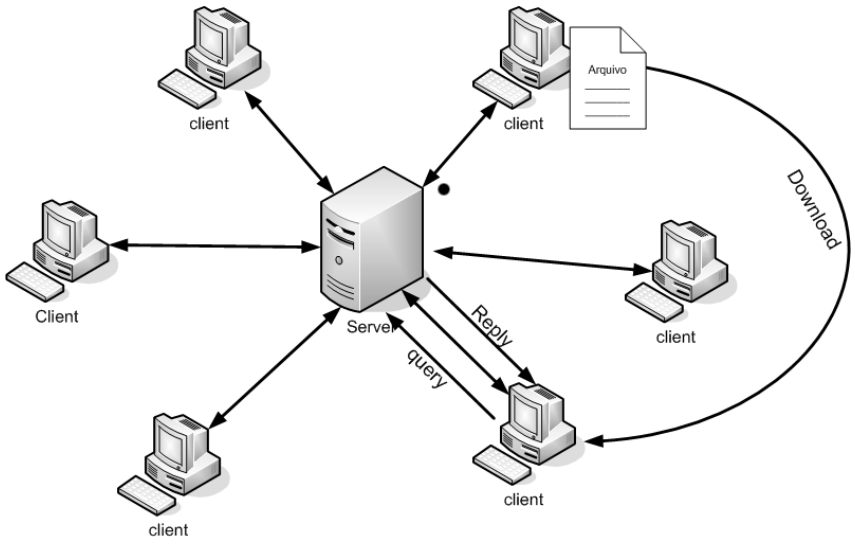

Figura 2.3: Sistema descentralizado híbrido. Adaptado de [Androutsellis-Theotokis and Spinellis, 2004]

A vantagem dos sistemas descentralizados híbridos é que eles são simples de se implementar e localizam arquivos de forma rápida e eficiente. A sua principal desvantagem é que eles são vulneráveis à censura, ações legais, monitoramento, ataques maliciosos e falhas técnicas, porque o conteúdo compartilhado ou pelo menos os seus descritores e a habilidade de acessá-los são controlados por quem mantém o servidor central (instituições, companhias ou usuários). Além do mais, estes sistemas são considerados inerentemente não escaláveis. O Napster [Saroiu et al., 2003] e Publius [Waldman et al., 2000] são exemplos de sistemas *p2p* descentralizados híbridos.

2.4.2 Estrutura das redes *p2p*

Como dito anteriormente, as redes *p2p* podem ser caracterizadas em termos de seu grau de centralização e sua estrutura. A forma como a rede *p2p* é criada, baseada em regras específicas ou de forma não determinista (*ad-hoc*), fornece a classificação da estrutura, que pode ser estruturada ou desestruturada.

Redes desestruturadas: Nas redes desestruturadas, a disposição dos arquivos não tem relação com a topologia da rede sobreposta. Em uma rede desestruturada, o conteúdo deve ser localizado. Mecanismos de busca variam desde a força bruta, como a propagação de buscas em largura ou profundidade (*breadth-first* ou *depth-first*) até que o conteúdo seja localizado, até estratégias mais sofisticadas e econômicas que incluem o uso de caminhos aleatórios e rotas indexadas. Os mecanismos de busca empregados em redes desestruturadas têm implicações em sua disponibilidade, escalabilidade e persistência.

Sistemas desestruturados são mais apropriados para acomodar populações de nós altamente transientes. Alguns exemplos destes sistemas são Napster, Publius, Gnutella, Edutella [Edutella, 2004] e o FreeHaven [Dingledine et al., 2000].

Redes estruturadas: O significado técnico para o termo "estruturado" é que a topologia da rede *p2p* sobreposta é firmemente controlada e o conteúdo é colocado em localizações específicas e não de forma aleatória, e fará com que as buscas sejam mais eficientes. Estes sistemas usam tabelas de *hash* distribuídas (DHT) [Mondejar et al., 2005] com base nas quais a informação da localização dos objetos é estabelecida deterministicamente nos pares, com identificadores correspondendo às chaves dos objetos. Os sistemas baseados em DHT serão discutidos mais detalhadamente na próxima seção (2.5).

2.5 Tabelas Hash Distribuídas

As tabelas de hash distribuídas (*Distributed Hash Tables* – DHT) são um dos mais recentes modelos para implementação de sistemas *p2p* estruturados para localização de informação. Uma DHT é uma tabela hash que é mantida por um conjunto de pares formando uma infra-estrutura *p2p*. Esta tabela hash é dividida em partes não sobrepostas (partição) e cada par deste conjunto passa a ser responável pela manutenção de cada uma das partições. Uma DHT pode servir como um repositório de objetos distribuídos, onde a localização de um objeto é determinada por uma chave, que pode ser obtida por uma função hash do nome do objeto. Esta chave (ID) é usada como identificador único para este objeto. Um ID aleatório e único é associado a cada par da rede que conhece um determinado número de pares.

No caso de sistemas *p2p* para publicação e armazenamento de conteúdo, estes objetos são documentos. Quando um documento é publicado (compartilhado) em tal sistema, um ID é associado ao documento baseado em uma função hash do seu conteúdo e no seu nome. Cada nó então encaminha o documento ao nó cujo ID é mais próximo do ID do documento. Esse processo é repetido até que o ID do nó atual seja o mais próximo do ID do documento. Cada operação de roteamento também garante que uma cópia local do documento seja mantida. Quando um par solicita o documento de um sistema *p2p*, a requisição irá até o nó com ID mais semelhante ao ID do documento. Esse processo continua até que uma cópia do documento seja encontrada. Então o documento é transferido ao nó que originou a requisição,

enquanto cada par que participou do roteamento permanecerá com uma cópia local do documento [Lua et al., 2005].

Apesar de o modelo DHT ser eficiente para comunidades grandes e globais, ele apresenta um problema relacionado ao ID do documento. Este precisa ser conhecido antes que uma requisição do documento seja realizada. Assim, é mais difícil implementar uma pesquisa nesse modelo que no modelo de inundação. Além disso, pode ocorrer a formação de "ilhas", onde a comunidade se divide em subcomunidades que não possuem nenhuma ligação entre si.

Os sistemas baseados em DHT têm como propriedade a designação uniformemente aleatória e consistente de IDs a um conjunto de pares chave/valor em um grande espaço de identificadores. Os identificadores designados aos objetos são únicos – chamados de chaves – e são escolhidos do mesmo espaço de identificadores. As chaves são mapeadas pelo protocolo da rede sobreposta a um único par ativo da rede.

Cada par mantém uma pequena tabela de roteamento consistindo do *nodeId* dos seus vizinhos e o seu endereço IP. Pedidos de busca ou roteamento de mensagens são encaminhados através dos caminhos sobrepostos até os pares, de uma forma progressiva ao *nodeId* que tem a chave mais próxima dentro do espaço de identificadores. Sistemas baseados em DHT possuem organizações diferentes para os objetos, suas chaves e suas estratégias de roteamento. Teoricamente, os sistemas baseados em DHT podem garantir que qualquer objeto pode ser localizado em um pequeno número de hops (em média $O(\log N)$, onde N é o número de pares no sistema). O caminho entre dois pares na rede subjacente pode ser significativamente diferente do caminho na rede DHT sobreposta. Assim, a latência das buscas em redes *p2p* baseadas em DHT pode ser bastante alta e pode afetar adversamente o desempenho das aplicações em execução sobre elas. Sistemas baseados em DHT são uma classe importante de infra-estruturas de roteamento *p2p*. Elas suportam o desenvolvimento rápido de uma vasta variedade de aplicações na escala da Internet, desde arquivos distribuídos e sistemas de nomes até camadas de aplicações multicast. Elas também viabilizam a recuperação de informação compartilhada de forma escalável dentro de WANs.

Atualmente as principais implementações do protocolo DHT são o Chord [Stoica et al., 2001], Bamboo/OpenDHT [Rhea et al., 2005], CAN [Ratnasamy et al., 2001], Pastry [Rowstron and Druschel, 2001a], Bunshin [Mondejar et al., 2005], DKS System [Alima et al., 2003], Kademlia [Maymounkov and Mazieres, 2002] e Tapestry [Apache, 2008].

2.6 Aplicações de redes *peer-to-peer*

As redes *p2p* têm sido usadas para uma vasta gama de aplicações, entre as quais podemos citar comunicação, colaboração, computação distribuída e distribuição de conteúdo.

2.6.1 Comunicação

A possibilidade de poder observar as pessoas entrando na rede e enviar uma mensagem em tempo real, tem tornado as aplicações de mensagem instantânea (IM – *Instant Messaging*) muito populares da Internet [Kamienski et al., 2005].

Diferentemente do correio eletrônico, em que uma mensagem é armazenada em uma caixa postal e posteriormente entregue ao usuário que verificou a caixa postal no seu servidor, os sistemas IM fornecem entrega imediata ao usuário. Se o usuário não está disponível, a mensagem pode ser armazenada até que o mesmo se torne "on-line", ou ela pode ser simplesmente

descartada. Para evitar esta incerteza na entrega, os sistemas IM fornecem uma lista de contatos com um mecanismo capaz de identificar um usuário e determinar o seu estado, por exemplo, ativo, inativo ou ocupado.

Devido à popularidade das mensagens instantâneas, não é surpresa que exista uma variedade de aplicações IM. Algumas dessas soluções são o AIM (AOL Instant Messenger - [AOL, 2009]), o MSN Messenger da Microsoft [MSN, 2009] e o Yahoo! Messenger [Yahoo, 2009] entre outras.

2.6.2 Colaboração

Os programas de trabalho colaborativo ou *groupware* são projetados para melhorar a produtividade de indivíduos com metas e interesses comuns. A expressão *groupware* é definida como um software que suporta colaboração, a comunicação e coordenação de vários usuários em uma rede. Isto inclui a integração de características como e-mail, calendário, espaços de trabalho, listas de discussão, sistemas de gerenciamento de documentos, vídeo conferência, entre outras [Kamienski et al., 2005].

Os sistemas *groupware* tradicionais como o Lotus Notes foram projetados para suportar o trabalho colaborativo em redes locais. A combinação das soluções *groupware* com a tecnologia *p2p* trouxe novas oportunidades. As atuais aplicações possibilitam a criação e extensão de grupos de trabalho espontaneamente, sem levar em consideração a localização. Algumas aplicações bem conhecidas são Windows Meeting Space [WindowsMeetingSpace, 2009] e Groove [Groove, 2007].

2.6.3 Computação distribuída

Usar os recursos computacionais ociosos de uma rede pode fornecer um serviço com alto desempenho computacional. A Grade computacional (*grid computing*) é um outro conceito de computação distribuída. A internet com sua flexibilidade e crescente largura de banda fornece uma infra-estrutura que preenche bem os requisitos de computação distribuída. Um dos primeiros eventos visíveis de computação distribuída ocorreu em janeiro de 1999, quando o projeto distributed.net [DCT, 2009], contando com a ajuda de várias dezenas de milhares de computadores na internet, quebrou o algoritmo RSA em menos de 24 horas [Kamienski et al., 2005].

Recentes projetos têm estimulado o interesse de muitos usuários dentro da comunidade Internet. O projeto *SETI@home* [UCA, 2009], por exemplo, possui um poder computacional de aproximadamente 25 TFlops/s (trilhões de operações de ponto flutuante por segundo), coletado de mais de três milhões de computadores conectados à Internet. Esse projeto visa utilizar essa grande capacidade de processamento para analisar os sinais obtidos a partir do rádio-telescópio do observatório de Arecibo a procura de algum sinal inteligente proveniente de fora da Terra.

2.6.4 Distribuição de conteúdo

A distribuição de conteúdo é uma das áreas de sistemas *peer-to-peer* que mais se sobressai atualmente. Em sua forma básica, os sistemas *p2p* para distribuição de conteúdo criam um espaço de armazenamento que permite a publicação, busca e recuperação de arquivos pelos membros da rede. Com os sistemas tornando-se mais sofisticados, funcionalidades podem ser

fornecidas, incluindo segurança, anonimato, aumento de escalabilidade e desempenho, assim como gerência de recursos e capacitação organizacional. As tecnologias *p2p* atuais podem ser agrupadas em aplicações *p2p* e em infra-estruturas *p2p*.

As aplicações *p2p* são sistemas para distribuição de conteúdo baseados na tecnologia *p2p*. Este grupo pode ser subdividido em dois outros grupos, baseado nos objetivos dessas aplicações e sua complexidade:

Sistemas de troca de arquivos *p2p*: Estes sistemas são orientados para trocas de arquivos simples entre os pares. São usados para configuração da rede de pares e fornecer serviços de busca e transferência de arquivos entre eles. São tipicamente aplicações leves que adotam a abordagem *best-effort* sem preocupações com segurança, disponibilidade e persistência.

Sistemas *p2p* para publicação e armazenamento de conteúdo: Estes sistemas são orientados para a criação de uma mídia de armazenamento distribuída em cujos usuários são capazes de publicar, armazenar e distribuir conteúdos de uma forma segura e persistente. O foco principal destes sistemas é a segurança e a persistência. Muitas vezes o alvo é a incorporação de meios para a responsabilização, anonimato e resistência à censura, assim como serviços de gerência de conteúdo persistente (atualização, remoção e controle de versão).

As infra-estruturas *p2p* incluem as infra-estruturas que não constituem aplicações, mas fornecem serviços baseados em *p2p* e estruturas para aplicações. Elas podem ser subdivididas em infra-estruturas para roteamento e localização, para fornecer anonimato e para gerência de reputação, como mostrado a seguir:

Roteamento e Localização: Qualquer sistema *p2p* de distribuição de conteúdo baseia-se em uma rede de pares na qual as requisições e mensagens devem ser roteadas com eficiência e com tolerância a faltas, através da qual os pares e o conteúdo podem ser eficientemente localizados.

Anonimato: Sistemas baseados em infra-estruturas *p2p* têm sido projetados visando o anonimato.

Gerência de Reputação: Em uma rede *p2p* não há uma organização central para manter informações sobre a reputação dos usuários e seu comportamento. A informação de reputação é, então, hospedada em vários nós da rede. Para que a informação de reputação possa ser mantida com segurança, atualizada e disponível através da rede, infra-estruturas complexas de gerência de reputação devem ser empregadas.

2.7 Migração, Replicação e *Cache*

Os Sistemas *p2p* para distribuição de conteúdo baseiam-se na replicação de conteúdo em mais de um nó para melhorar a disponibilidade, desempenho e resistência à censura. A replicação de conteúdo pode ser classificada nas seguintes categorias:

Replicação passiva: A replicação do conteúdo ocorre naturalmente no sistema conforme os nós solicitam e copiam os dados de uns para os outros.

Replicação baseada em *cache*: É uma forma de replicação empregada em muitos sistemas como o OceanStore [Kubiatowicz et al., 2000], Mojonation [McCoy, 2000] e o Freenet [Clark et al., 2000]. É o resultado do cache de cópias enquanto elas passam pelos nós através da rede.

Replicação ativa (ou proativa): A replicação do conteúdo de forma ativa e os métodos de migração são normalmente empregados para melhorar a capacidade do sistema em localizar dados, assim como a disponibilidade e desempenho.

Gerenciamento de réplicas introspectivo: São técnicas empregadas pelo sistema OceanStore e pelo sistema MonjoNation, na qual os nós monitoram os padrões de uso, a atividade da rede e a disponibilidade dos recursos para se adaptar a paradas locais ou ataques de negação de serviço. Assim os dados são migrados para as áreas de uso, mantendo níveis de redundância dos dados em um nível suficientemente alto para atender a demanda.

Gerenciamento de réplicas dinâmico: São algoritmos usados no sistema Scan [Chen et al., 2000] para alocar dinamicamente um número mínimo de réplicas suficiente para atender a restrições de qualidade de serviço e a capacidade dos servidores. Estes algoritmos são projetados para satisfazer tanto a latência dos clientes como a carga dos servidores da seguinte forma: primeiramente procurando por cópias que atendam as restrições de latência dos clientes sem se sobrecarregar e, se não obtiver sucesso, alocando novas réplicas.

Ao se replicar conteúdos surgem questões relacionadas à consistência de dados e sincronização, especialmente em sistemas que permitem a remoção e atualização de conteúdo. Algumas aplicações enfraquecem seus requisitos de consistência em favor de uma replicação de dados mais extensiva e maior disponibilidade.

2.8 Aspectos de Segurança

Arquiteturas *p2p* para distribuição de conteúdo representam um desafio para o fornecimento de níveis de disponibilidade, privacidade, confidencialidade, integridade e autenticidade normalmente requeridas devido a sua natureza aberta e autônoma. Os nós da rede devem ser considerados partes não confiáveis e não se pode fazer nenhuma suposição em relação ao seu comportamento.

As questões sobre controle de acesso, autenticação e gerência de identificação são normalmente ignoradas em sistemas *p2p* para distribuição de conteúdo. Quando um mecanismo de controle de acesso é utilizado ele segue o paradigma do controle de acesso discricionário, que é claramente uma abordagem insegura quando entram em cena clientes não confiáveis. As listas de controle de acesso também podem ser atribuídas a objetos por seus autores através do uso de certificados assinados como, por exemplo, no sistema OceanStore.

Muitos algoritmos e protocolos para criptografia são empregados para prover segurança ao conteúdo publicado e armazenado em redes *p2p*, entre os quais podem ser mencionados:

Dados auto-verificáveis: São dados cuja integridade pode ser verificada pelo nó que os está recuperando [Castro et al., 2002]. Um nó que insere um arquivo na rede calcula a assinatura *hash* do seu conteúdo baseado em uma função *hash* conhecida para produzir a chave

do arquivo. Quando um nó resgata o arquivo utilizando esta chave, ele calcula a mesma função *hash* para verificar a integridade do arquivo.

Dispersão da informação: O algoritmo da dispersão da informação é amplamente usado [Rabin, 1989]. Os arquivos são codificados em m blocos, de forma que n são suficientes para remontar o arquivo original. Isto provê resiliência proporcional a um fator de redundância igual a m/n.

Esquema de compartilhamento secreto de Shamir (SHA): Apresentado em [Shamir, 1979]. O nó que publica a informação cifra o arquivo com uma chave K, então divide K em λ partes, de tal forma que qualquer quantidade κ delas pode reproduzir K, mas $\kappa - 1$ não fornece nenhuma informação sobre K. Cada servidor então cifra uma das partes juntamente com o arquivo. Para que o arquivo fique inacessível, pelo menos $(\lambda - \kappa - 1)$ servidores contendo a chave precisam cair.

Retransmissores criptográficos anônimos: Um mecanismo baseado em pares com funções *publisher*, *forwarder*, *storer* e *client* comunicam-se através de camadas de conexão anônima. Um *publisher* seleciona vários *forwarders* e envia a eles, através de uma conexão anônima, partes cifradas de um arquivo. Os *forwarders*, por sua vez, selecionam outros nós para agir como *storers* e enviam estas partes aos *storers*, também via uma conexão anônima. Assim que todas as partes cifradas estiverem armazenadas, o *publisher* destrói as suas cópias locais e anuncia o nome do arquivo juntamente com a lista dos *forwarders* que foram usados.

Para recuperar a informação, um *client* contactará um *forwarder* que por sua vez contactará um servidor aleatório para decifrar os endereços dos *storers* das partes cifradas. Os *forwarders* solicitarão aos *storers* as partes da informação. Estes irão decifrar os dados e devolvê-los ao *client*. O processo irá se repetir até que existam partes suficientes para reconstruir o arquivo.

Sistema de arquivos distribuído Esteganográfico: Baseado em [Anderson et al., 1998]. A sua principal propriedade é que blocos cifrados são indistinguíveis de um substrato aleatório, de tal forma que a sua presença não pode ser detectada. O sistema é preparado para inicialmente escrever dados aleatórios em todos os blocos e então os arquivos são armazenados através da cifragrem dos seus blocos e de sua inserção de forma pseudo-aleatória. Para evitar colisões, uma quantidade considerável de replicas é necessária.

Código de apagamento: Apresentado em[Lamport et al., 1982]. Os dados são divididos em blocos e espalhados por vários servidores. Somente uma fração destes é necessária para reconstruir o arquivo (similar ao algoritmo de dispersão da informação).

2.8.1 Anonimato

O anonimato também é um aspecto de segurança, sendo o foco principal de muitas infra-estruturas baseadas e *p2p* e sistemas de distribuição de conteúdo objetivando a privacidade, confidencialidade e a resistência à censura.

O Freenet é um sistema de distribuição de conteúdo *p2p* que objetiva especificamente o fornecimento de anonimato para os seus usuários, por tornar impraticável a descoberta da verdadeira origem ou destino de um arquivo passando pela rede e dificultar a um determinado nó

a determinação ou responsabilidade pelo conteúdo atual que esteja armazenando. Ele emprega um mecanismo no qual quando uma busca acontece, o arquivo é transferido do nó que o possuía até o seu requisitante através de cada nó que encaminhou a requisição de busca. Para alcançar o anonimato, cada o nó pertencente a este caminho pode unilateralmente decidir declarar ele mesmo ou outro nó escolhido arbitrariamente como sendo a fonte do arquivo. Desta forma, não há conexão entre o requisitante e o atual fornecedor do arquivo.

Os sistemas de distribuição de conteúdo que fornecem anonimato normalmente empregam infra-estruturas com camadas de conexões anônimas. Estas camadas podem empregar diversas técnicas diferentes, tais como a partição do documento em partes codificadas e enviá-las através de uma camada de nós para anonimato e enviar estas partes a outros nós que as irão armazenar, destruindo depois disto o documento original.

O ambiente Chord [Stoica et al., 2001] fornece resistência à censura por se focar no anonimato do distribuidor, armazenador e solicitante, tornando difícil para um nó se tornar voluntariamente responsável pelo documento.

2.8.2 Repudiação

Repudiação em sistemas *p2p* para distribuição de conteúdo refere-se à habilidade de cada usuário repudiar (negar) o conhecimento do conteúdo armazenado em seu nó. Como conseqüência, os usuários não podem ser responsabilizados pelo conteúdo armazenado em seu nó ou por ações efetuadas por seus nós como parte de sua operação em uma rede *p2p*. A negação pode ser aplicada ao conteúdo armazenado como também ao conteúdo sendo transferido.

A negação do conteúdo armazenado é oferecida por sistemas que armazenam partes codificadas e não armazenam as suas chaves e, portanto não podem ter conhecimento do conteúdo dos arquivos cujas partes eles estão armazenando. Similarmente, quando se usam sistemas distribuídos de armazenamento esteganográfico como infra-estrutura, a negação também é fornecida pelo fato de os blocos do arquivo escrito no sistema de arquivos do nó não serem detectáveis.

A negação do conteúdo em trânsito pode ser fornecida através do uso de camadas de conexões anônimas incorporadas ao sistema, por tornar impraticável a descoberta da origem real de um arquivo que está circulando pela rede. Durante a recuperação de um arquivo, mais nós do que seriam necessários são contactados e mais arquivos do que o necessário são recuperados, de forma que o objeto real é despistado. Sistemas estruturados não podem oferecer negação, pois os identificadores dos arquivos armazenados nos nós são associados ao endereço do nó. Por outro lado, o dono do nó não necessariamente requisitou o arquivo e não tem controle sobre se ele será armazenado em seu nó, logo não poderá ser responsabilizado.

2.9 Mecanismos de Incentivo e Responsabilização

A operação, desempenho e disponibilidade de um sistema *p2p* descentralizado conta, em grande parte, com a participação voluntária dos seus usuários. Para isto, faz-se necessário empregar mecanismos que forneçam incentivos e estimulem a cooperação entre os usuários, assim como também é necessária alguma noção de responsabilização pelas ações feitas. O fornecimento de incentivos e responsabilização em redes *p2p* com populações transientes de usuários, onde a identificação dos pares e a obtenção de informações sobre o seu comportamento passado para prever o seu desempenho futuro pode ser uma tarefa particularmente desafiadora,

especialmente devido à ausência de um sistema ubíquo, efetivo, robusto e seguro para fazer e aceitar micro pagamentos.

Dois tipos de mecanismos de incentivo são possíveis nestes casos, mecanismos baseados em confiança ou em permuta. No primeiro caso a confiança é um incentivo justo para a cooperação na qual cada um se engaja na transação baseado na confiança na outra parte. Os mecanismos de reputação pertencem a esta categoria. Já nos mecanismos de incentivo baseados em permuta, cada parte que oferece algum serviço à outra é explicitamente remunerada direta ou indiretamente.

2.10 Gerência de Recursos

Os recursos que os sistemas *p2p* para distribuição de conteúdo lidam tipicamente são arquivos (conteúdo), armazenamento (espaço de disco) e capacidade de transmissão (largura de banda). Qualquer sistema deve executar, no mínimo, operações para inserir, localizar e recuperar conteúdos. O gerenciamento de recursos pode oferecer, adicionalmente, facilidades como remoção, atualização, manutenção de versões, gerência do espaço de armazenamento e ajuste dos limites da capacidade de transmissão.

A remoção e atualização de arquivos: Não são operações naturais em ambientes *p2p* se deve ser mantida a sincronização corretamente. Sistemas como MojoNation [McCoy, 2000] usam arquivos imutáveis que não podem ser atualizados. A única forma de atualizá-los é pela disponibilização de uma nova versão do documento com um nome diferente. O PAST [Rowstron and Druschel, 2001b] é similar neste aspecto, ele não oferece a funcionalidade de remoção, mas não garante que arquivo estará disponível em algum lugar da rede. O FreeHaven também proíbe a retirada ou a revogação de documentos principalmente por razões de segurança e robustez a ataques. Já o sistema Publius oferece ambos, atualização e remoção, pois ele é baseado em um conjunto estático de servidores para armazenamento de conteúdo.

Expiração da validade: A data de expiração da validade de documentos foi efetivamente introduzida pelo sistema FreeHaven, através do uso de contratos com durações diferentes.

Versionamento: Uma abordagem mais sofisticada é empregada pelo sistema OceanStore, que oferece um sistema de armazenamento baseado em versões arquivadas.

Estrutura de diretórios: Um sistema completo de estrutura de diretórios distribuído e *inodes* como os do Unix está disponibilizado pelo Mnemosyne [Hand and Roscoe, 2002].

Localização de conteúdo: Facilidades de busca podem também variar no seu grau de funcionalidade e desempenho. Sistemas não estruturados como Gnutella, Kazaa e FreeHaven oferecem mecanismos de busca por palavras chave que são convenientes, porém têm problemas de escalabilidade. Em sistemas estruturados, as buscas são muito eficientes, porém só podem ser baseadas em identificadores de arquivos.

Capacidade de armazenamento e de transmissão: O gerenciamento do espaço de armazenamento disponível para os pares também varia entre sistemas. Em sistemas como MojoNation os usuários contribuem com armazenamento em troca de compensações econômicas ou de outro tipo. O PAST usa um sistema de quotas seguro onde os usuários recebem

uma quota fixa de armazenamento para seu uso ou podem usar uma quantidade equivalente a qual eles contribuem em seus nós.

2.11 Conclusão

Os sistemas *peer-to-peer* são amplamente distribuídos e têm uma população de nós altamente voláteis. Estes sistemas têm como um de seus objetivos o compartilhamento de grandes quantidades de recursos. Os pares se comunicam através de topologias de redes auto-organizadas e tolerantes a faltas, que operam como uma rede sobreposta sobre a rede física. Estas características fazem dos sistemas *p2p* uma base para a criação de novos serviços muito flexíveis e escaláveis.

Neste capítulo foi apresentado o estado da arte relacionado as tecnologias *p2p*, onde foram tratados vários temas entre eles: definição, características, estrutura, funcionalidades e implementações. Também foi abordada a tecnologia de tabelas hash distribuídas. No próximo capítulo serão apresentados uma conceituação de sistemas de *backup* cooperativo e em seguida os principais trabalhos desenvolvidos na área de sistemas de *backup* distribuído.

Capítulo 3

Sistemas de *backup* cooperativos

3.1 Introdução

Um *backup* é a cópia de dados de um dispositivo para o outro com o objetivo de posteriormente os recuperar no caso de perda de dados. O *backup* é útil em princípio por duas razões: para recuperação de um computador para um estado operacional anterior a um desastre e para recuperar um pequeno número de arquivos após o seu apagamento acidental ou corrupção. Os dispositivos utilizados para armazenamento de *backups* podem ser unidades de fita magnética, sistemas de discos rígidos, unidades de disco óticas ou ainda unidades de armazenamento de estado sólido entre outros. Um sistema de *backup* tradicional é normalmente composto por um servidor remoto (muitas vezes isolado e protegido contra fogo, intempéries e outros riscos físicos), acessível através da rede e com uma grande capacidade local de armazenamento de dados. A organização e manutenção do processo de *backup* é uma tarefa complexa. Os *backups* são a última defesa contra a perda de dados e conseqüentemente são os sistemas menos granulares e convenientes de se usar.

Os *backups* e os sistemas de *backup* são frequentemente confundidos com cópias e sistemas de arquivos tolerantes a faltas. Os *backups* diferem das cópias no sentido de que os primeiros são uma cópia primária dos dados, normalmente guardados para uso futuro, enquanto os *backups* são uma cópia secundária, guardados para o caso da necessidade de recuperação do arquivo original. A função de um sistema de *backup* é tolerar faltas que afetam alguns equipamentos de armazenamento independente de sua localização (local ou distante) ou concepção (centralizado ou compartilhado) [Killijian and Courtes, 2006]. Os sistemas de *backup* diferem dos sistemas de arquivos tolerantes a faltas porque os sistemas de *backups* assumem que uma falta causará perda de dados e os sistemas tolerantes a faltas assumem que as faltas não causarão perdas. Um sistema de *backup* contém pelo menos uma cópia de todos os dados sensíveis e conseqüentemente os requisitos para armazenamento de dados podem ser consideráveis.

Será apresentado neste capítulo inicialmente uma conceituação de sistemas de *backup* cooperativo e, em seguida, uma revisão dos principais trabalhos nessa área.

3.2 Características dos Sistemas de *Backup*

Nesta seção são apresentadas as principais características dos sistemas de *backup*.

Modelos de repositórios de dados: Qualquer estratégia de *backup* inicia com um conceito de repositório de dados. Os *backups* precisam ser armazenados e organizados. Podem-se citar vários modelos de repositórios diferentes tais como: *não estruturados*, *completo+incremental*, *completo+diferencial*, *espelho+incremental reverso* e *proteção de dados contínua* [Chervenak et al., 1998].

O modelo de repositório *não estruturado* pode utilizar qualquer tipo de mídia com um mínimo de informação sobre o seu conteúdo. É o mais fácil de ser implementado, mas também é o que tem o menor grau de capacidade para recuperar dados.

O modelo de repositório *completo+incremental* tem como objetivo tornar possível o armazenamento de muitas cópias dos dados de origem. Inicialmente um *backup* completo é feito. Os *backups* subseqüentes são incrementais, isto é, somente os arquivos que foram modificados são armazenados. Este modelo oferece um alto nível de segurança, porém o problema é ter de lidar com longas séries de incrementos e com a alta necessidade de armazenamento.

O modelo *completo+diferencial* difere do modelo *incremental* por ter cada um dos seus *backups* parciais capturando todas as mudanças desde o *backup* completo. A sua vantagem é que a recuperação envolve somente a recuperação do *backup* completo mais o último *backup* diferencial.

O modelo de repositório *espelho+incremental reverso* é similar ao modelo *completo+incremental*. Ao invés de um *backup* completo seguido por uma série de incrementos este modelo oferece um espelho que reflete o estado do sistema como se fosse o último *backup* e um histórico de incrementos reversos. Um benefício deste modelo é que ele requer somente um *backup* inicial. Cada *backup* incremental é aplicado imediatamente ao espelho e os arquivos que ele substituiu são movidos para o incremento reverso. Este modelo não é adequado ao uso com mídias removíveis, porque cada *backup* deve ser feito em comparação com o espelho.

O modelo *proteção de dados contínua* vai um passo além e ao invés de fazer *backups* periódicos, o sistema faz um registro no histórico (log) imediatamente após cada mudança nos dados. Isto permite que sejam recuperadas versões anteriores dos arquivos através do uso do histórico.

Utilização do repositório: Além do modelo de repositório de dados deve ser levado em consideração a relação entre acessibilidade, segurança e custo na utilização do repositório [Chervenak et al., 1998].

O repositório pode fornecer *backup on-line*, que é o tipo mais acessível de armazenamento de dados, que pode ser recuperado em milisegundos. Este tipo é muito conveniente e rápido, porém é relativamente caro. O *backup on-line* é vulnerável ao apagamento ou sobrescrita acidental ou de forma maliciosa.

O armazenamento *near-line* é menos acessível e menos caro que o armazenamento *on-line*. Um equipamento mecânico (biblioteca de fitas magnéticas) normalmente está envolvido e o tempo de recuperação passa para a ordem de segundos.

O armazenamento *off-line* é parecido ao *near-line*, porém requer a interação humana para manipular a mídia. Os tempos de *backup* podem levar mais de uma hora.

Para a proteção contra desastres ou outros problemas específicos o *backup* pode ser armazenamento em outra localização física, chamado de armazenamento *off-site*.

Pode ser citado também o *backup site*, também chamado de *Disaster Recovery Center*, que em caso de um desastre para o qual somente a mídia de *backup* não é suficiente para a recuperação, mas todo o sistema de computação e rede propriamente configurados são necessários, o que pode significar um investimento muito elevado.

Utilização de recursos: Os sistemas de *backup* podem utilizar técnicas de compressão clássicas para economizar o espaço de armazenamento e também a largura de banda da rede, podendo ser aplicados tanto do lado do cliente como do lado do servidor. A técnica de armazenamento de instância única (*single-storage instance*) surgiu recentemente [Bolosky et al., 2000a] para reduzir a área de armazenamento necessária para o *backup* de vários sistemas de arquivos, através do armazenamento de uma única cópia de um bloco de dados que está presente no sistema de arquivos de vários usuários, ou se houver várias instâncias do mesmo bloco em um sistema de arquivos.

Desempenho: O desempenho de um sistema de *backup* é medido em termos de tempo de *backup* e tempo de recuperação. Em um sistema de *backup* cooperativo, outra métrica que pode ser utilizada como exemplo é o número de nós que o sistema pode comportar.

Integridade e consistência: Um serviço de *backup* deve garantir a integridade e consistência dos dados recuperados. Qualquer alteração (intencional ou não) nos dados do *backup* deve ser detectada durante a recuperação. A consistência torna-se um problema quando múltiplos itens devem garantir algum tipo de semântica em comum, como por exemplo, em um sistema de controle de produção, onde os sistemas de controle de estoque, entradas e saídas podem estar distribuídos entre vários processos em diferentes máquinas. Nestes casos, um cuidado especial deve ser tomado no gerenciamento de dependências.

Confidencialidade e Privacidade: O serviço de *backup* cooperativo deve garantir a confidencialidade dos dados. Além disto, o serviço deve proteger a privacidade dos seus usuários. Por exemplo, ele não deve enviar nenhuma informação referente à localização passada ou presente dos seus usuários.

Disponibilidade: O objetivo primário de um sistema de *backup* é a garantia da disponibilidade dos dados armazenados por longo prazo. Além disto, para ser útil, o sistema de *backup* deve estar disponível, isto é, ser resiliente a falhas, especialmente no que diz respeito a ataques maliciosos.

3.3 Características dos Sistemas de *backup* Cooperativos

Sistemas de *backup* cooperativos [Lillibridge et al., 2003] usam a tecnologia *peer-to-peer* para construir um ambiente de *backup* distribuído não-hierárquico, no qual cada nó do ambiente usa o espaço ocioso em seu disco local para, de forma cooperativa, armazenar dados de outros nós.

Os serviços de *backup* cooperativos devem oferecer confidencialidade e privacidade aos seus usuários, integridade, consistência e disponibilidade de dados, sinergia dos recursos e gerência de confiança aos pares [Killijian and Courtes, 2006]. Na seção anterior os itens

confidencialidade, privacidade, integridade, consistência e disponibilidade de dados já foram apresentados.

Sinergia: A sinergia em sistemas de *backup* cooperativos somente pode ser alcançada se os nós cooperarem entre si, ao invés de seguirem alguma estratégia individualista de curto prazo para obter vantagens. Uma forma de se garantir a sinergia é através do estímulo da propriedade das trocas justas (*fair exchange property*): é desejável que a quantidade de recursos oferecidas e obtidas do serviço sejam equivalentes [Killijian and Courtes, 2006].

Gerência de confiança: A implementação de um serviço de *backup* cooperativo entre nós sem o prévio estabelecimento dos relacionamentos de confiança não é trivial, pois novas ameaças devem ser consideradas:

- Nós com comportamento egoísta que se recusam a cooperar;
- Os repositórios de *backup* podem falhar ou atacar a confiabilidade ou a integridade dos dados;
- Dispositivos mal intencionados podem procurar negar serviços aos seus pares inundando-os com solicitações falsas de *backup*.

Por estes motivos fazem-se necessários mecanismos de gerência de confiança para dar suporte a serviços de *backup* cooperativo entre dispositivos mutuamente suspeitos.

3.4 Principais Trabalhos na Área de Sistemas de *Backup* Cooperativos

Nesta seção serão apresentados os principais trabalhos na área de sistemas de *backup* cooperativos, iniciando pelo sistema *CBS*, e em seguida são apresentados os sistemas *Pastiche*, *Venti-DHash*, *PeerStore*, *pStore*, *ABS*, *Phoenix*, *DIBS*, *friend-to-friend* e finalmente o *BAR*, nessa ordem.

3.4.1 Sistema de *Backup* Cooperativo *CBS*

O *CBS* e foi proposto em [Lillibridge et al., 2003]. Este sistema requer que os pares estejam conectados à Internet para permitir as operações de *backup* e *restore* (recuperação) a qualquer momento. Os pares formam uma rede sobreposta *p2p*. O sistema utiliza uma infraestrutura mínima de suporte. Ele não requer que os nós tenham identificadores únicos, chaves públicas certificadas ou algoritmos de roteamento de mensagens descentralizados. O *CBS* precisa somente de um serviço de localização de pares, o que pode ser implementado por um serviço *p2p* ou por outros meios (*out-of-band*), como por exemplo *Instant-Messaging, e-mail*, ou qualquer outro tipo de comunicação entre os pares.

Cada participante forma um conjunto de pares composto de nós de diferentes localizações geográficas. A diversidade geográfica é importante para garantir modos de falha independentes para os pares – isto é análogo a fazer fitas de *backup off-site* para os sistemas de *backup* tradicionais. O relacionamento entre os pares é recíproco. No entanto, este relacionamento pode ser transitivo, pois podem haver parceiros com comportamento arbitrário. Por exemplo, estes

parceiros podem estar sempre desligados ou desconectados. Neste caso, o acordo de *backup* será cancelado com este parceiro e outro será localizado em seu lugar.

Tomando-se como base as funções de um sistema de *backup* (localização de recursos, redundância de dados, recuperação de dados), este é o mais simples.

O uso de *backup* incremental, preservação de recursos e otimização de desempenho não são tratados nele. Um servidor centralizado é utilizado para localização de parceiros, e por isto, este sistema já apresenta um ponto de falha único, tornando-o vulnerável a ataques. Alguns mecanismos foram criados para defendê-lo contra ataques, como o *periodic random challenges*[1] para garantir que os pares continuam armazenando os dados de *backup*. Ele também emprega códigos de apagamento [Rodrigues and Liskov, 2005] em conjunto com técnicas criptográficas para obter tolerância a faltas. Todavia, a maior parte do esforço do projeto foi voltado para tentar impedir comportamentos egoístas.

3.4.2 O sistema *Pastiche*

O sistema *Pastiche* [Cox et al., 2002] utiliza-se de parte da capacidade de armazenamento de dados não utilizada dos pares em seu sistema para fornecer um serviço de *backup* eficiente e sem custos administrativos. Os nós do sistema *Pastiche* operam de forma cooperativa. Devido ao *churn*, cada nó do sistema deve replicar seus dados em mais de um par. A maioria das réplicas ficam próximas, em termos de localização física, para reduzir a sobrecarga da rede e minimizar o tempo de *restore*, porém pelo menos uma réplica deve ser guardada em outra localização para prevenção de catástrofes. O sistema *Pastiche* não consome recursos da parte do usuário e consome pouco espaço de disco adicional para fornecer um serviço de *backup* automático. Este sistema é voltado para os computadores do usuário final. O sistema *Pastiche* não guarda dados duplicados em seus pares. Ele identifica estes dados duplicados e os agrupa para fazer um uso mais eficiente do espaço de armazenamento do sistema. O sistema *Pastiche* prepara pequenos sumários do conteúdo do sistema de arquivos dos pares para que os potenciais parceiros possam inspecionar e verificar se existe sobreposição de dados e assim identificar duplicatas dos mesmos.

O sistema *Pastiche* foi construído sobre três tecnologias recentes, o Pastry, *Content–based indexing* [Manber, 1994], que fornece uma forma flexível de localização de dados redundantes em arquivos similares, e *Convergent encryption* [Bolosky et al., 2000b], que permite que nós utilizem a mesma representação criptográfica para dados comuns, sem a necessidade de compartilhar chaves.

O sistema *Pastiche* fornece um novo sistema de arquivos, o *Chunkstore*, que armazena todos os dados do nó, assim como o status do *backup*, sem comprometer o desempenho do sistema. O status dos arquivos é guardado por uma árvore de metadados. A raiz desta árvore pode ser recuperada a partir do substrato Pastry somente com o nome e uma *passphrase* da máquina a ser recuperada. Um sistema de arquivos completo pode ser recuperado tão facilmente quanto um arquivo simples.

Para atender ao problema de armazenar dados em nós não confiáveis, o sistema *Pastiche* utiliza mecanismos probabilísticos para detectar perdas de *backups*, através de solicitações periódicas de dados armazenados aos pares parceiros.

[1] *Periodic random challenges* são questionamentos periódicos feitos de um nó a outro para verificar se este ainda está em posse dos dados de *backup*. Nestes questionamentos são solicitadas informações aleatórias sobre os dados de *backup* que o nó somente será capaz de responder se ainda estiver em posse dos dados de *backup*.

3.4.3 O sistema *Venti-DHash*

O *Venti-DHash* [Sit et al., 2003] é um sistema de *backup* cooperativo do tipo *off-site*. É baseado em uma infra-estrutura DHT e foi projetado para suportar a recuperação de dados após um desastre através do armazenamento regular de *snapshots* [2] de sistemas de arquivos em pares distribuídos pela Internet. Pelo fato de construir este sistema sobre uma DHT, a aplicação de *backup* herdou as suas propriedades e, por conseguinte, pode ser utilizada como uma aplicação de larga escala. O *Venti-DHash* funciona como um sistema armazenador de arquivos que guarda *snapshots* completos dos sistemas de arquivos, dividido em forma de blocos. Cada bloco é único e só é armazenado uma vez, mesmo entre vários *snapshots*. O sistema *Venti-DHash* faz um balanceamento entre a carga relativa ao armazenamento e da rede, assim como fornece uma disponibilidade dos blocos na ordem de 5 noves por bloco (99,999%).

O sistema *Venti-DHash* utiliza um armazenamento completamente distribuído entre todos os participantes da mesma forma que em um sistema de arquivos compartilhados *peer-to-peer*.

3.4.4 O sistema *PeerStore*

O sistema *PeerStore* [Landers et al., 2004] se diferencia de outros sistemas por separar a gerência de metadados da gerência de armazenamento de *backup*, conforme mostrado na figura 3.1.

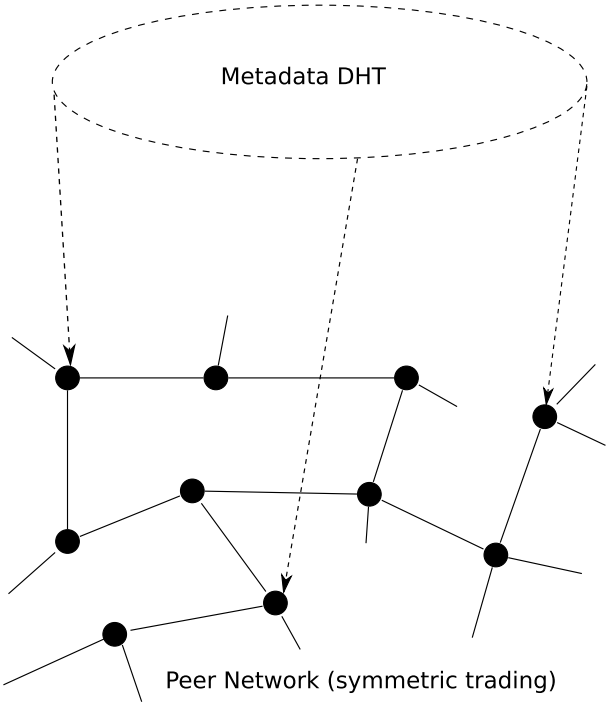

Figura 3.1: PeerStore: níveis de metadados e *backups* [Landers et al., 2004].

Desta forma ele oferece algumas vantagens em relação aos outros sistemas, tais como:

[2] Um *snapshot* (instantâneo) é um conjunto de arquivos e diretórios no estado em que eles se encontravam em algum ponto do passado. O termo foi utilizado como uma analogia ao mesmo em fotografia.

- Emprega estratégias diferentes para os seus dois níveis: o nível de metadados e o nível de armazenamento. Emprega um mecanismo de buscas eficiente para o nível de metadados e um mecanismo de armazenamento flexível para disposição dos dados de *backup*.
- O nível de armazenamento pode ser ajustado visando lealdade no uso do espaço de armazenamento.
- Os metadados podem ser utilizados para localizar rapidamente todas as réplicas de um determinado arquivo, mesmo se as réplicas estiverem armazenadas em locais distintos.
- Pelo fato de os metadados serem pequenos em relação aos arquivos, os metadados são mantidos de forma replicada para garantir a sua disponibilidade.

No sistema *PeerStore*, a gerência dos metadados é realizada por uma DHT. Através do armazenamento dos metadados desta forma, a detecção de duplicatas dos dados pode ser executada de forma eficiente. Ao mesmo tempo, nenhum dado real necessita ser migrado quando nós entram ou saem da rede, somente a informação contida nos registros dos metadados é que deve ser transferida e atualizada, poupando assim uma grande parte dos custos de manutenção. O armazenamento de dados, por outro lado, baseia-se em um esquema de trocas simétrico. Um par que deseja fazer *backup* de seus dados deve, por sua vez, armazenar em seu sistema de arquivos dados de *backup* de algum de seus parceiros. O sistema *PeerStore* é capaz de executar *backup* incremental somente para os dados novos ou modificados recentemente.

3.4.5 O sistema *pStore*

O sistema *pStore* [Batten et al., 2002] foi motivado na necessidade de um sistema de *backup* para os dados pessoais dos usuários. Ele foi projetado para que os usuários possam armazenar e recuperar *backups* de forma segura em uma rede distribuída formada por nós não confiáveis (*untrusted*). O sistema *pStore* mantém *snapshots* de cada arquivo, permitindo assim que o usuário recupere qualquer *snapshot* em uma data posterior. Ele possui três objetivos primários: confiabilidade (*reliability*), segurança e eficiência no uso de recursos. Este sistema fornece confiabilidade através de replicação, ou seja, cópias secundárias são distribuídas por vários nós da rede, para o caso de que alguns deles estejam indisponíveis ou sejam maliciosos.

Pelo fato de os dados dos clientes estarem replicados em nós que estão fora do seu controle, o sistema *pStore* fornece um nível de segurança razoável a estes dados. Através do uso de técnicas criptográficas, ele cifra os dados de forma que somente o seu dono possa lê-los. Além disso, somente o dono dos dados pode removê-los e qualquer alteração nos dados pode ser facilmente detectada. Finalmente, como frequentemente os *backups* são grandes, o sistema *pStore* procura reduzir o uso dos recursos do sistema através do compartilhamento dos dados armazenados, somente enviando dados pelo sistema quando necessário.

3.4.6 O sistema *ABS*

O sistema *ABS* [Cooley et al., 2004] utiliza um sistema de versionamento baseado no Rsync [3]. Também utiliza técnicas de codificação [Rodrigues and Liskov, 2005] para tratar

[3] O Rsync é um aplicativo de software para o sistema operacional Unix que sincroniza arquivos e diretórios de um local para outro enquanto minimiza a transferência de dados através do uso de codificação delta quando

problemas de *churn* e perda de dados. Para fornecer privacidade e segurança ele utiliza criptografia. O sistema *ABS*, através do uso de armazenamento de instância única *single instance store* [Bolosky et al., 2000a] fornece um uso eficiente da área de armazenamento. Ele utiliza o versionamento para ter um arquivamento eficiente dos dados e implementa uma DHT para prover um serviço de localização para o armazenamento dos *backups*, balanceamento de carga e gerência do espaço de chaves. Finalmente, o sistema *ABS* implementa um esquema para verificação dos dados armazenados inovador e muito eficiente, como mostrado na figura 3.2 a seguir.

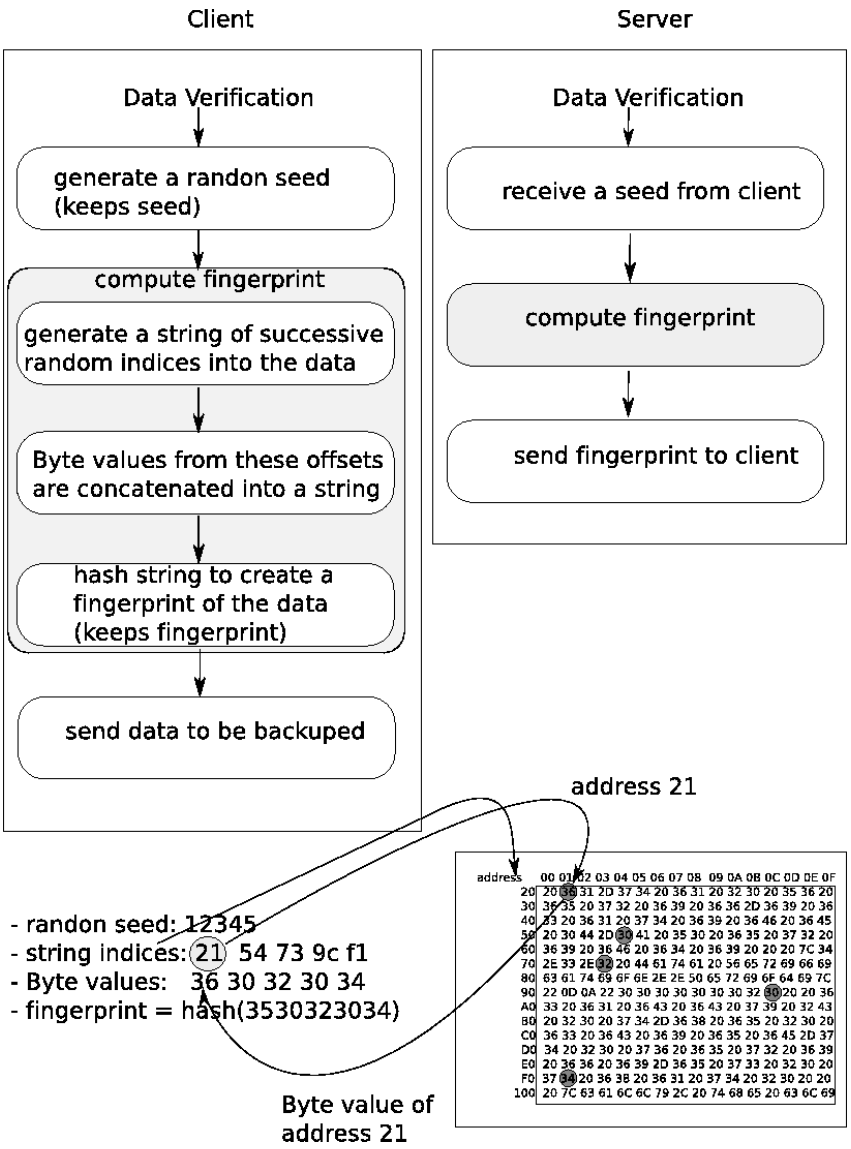

Figura 3.2: ABS: Processo de verificação dos dados armazenados.

Com este esquema pode-se detectar o comportamento egoísta de nós. Este esquema consiste em, a partir de uma semente numérica e um algoritmo para geração de números alea-

apropriado. Uma funcionalidade importante do Rsync que não é encontrada na maioria dos programas similares é que o espelhamento de dados acontece somente em uma vez em cada direção. O Rsync pode copiar ou mostrar o conteúdo de diretórios e copiar arquivos, opcionalmente pode utilizar compressão e recursão.

tórios (utilizados como endereços dentro do arquivo), coletar alguns bytes deste arquivo. Com estes bytes concatenados, e com o uso de um algoritmo de assinatura *hash*, é criada uma assinatura para este arquivo. Sempre que um nó quiser verificar se os seus parceiros ainda estão armazenando o seu *backup*, este nó envia a semente numérica ao seu parceiro para que este calcule a assinatura do *backup* em seu poder e a retorne. Caso a assinatura guardada com o nó não coincida com a que foi calculada pelo seu parceiro, significa que há um problema e este parceiro perdeu ou corrompeu o *backup*. Com este procedimento é possível ter uma garantia de que os pares irão guardar os *backups* até pelo menos a primeira verificação, quando eles poderão então calcular a assinatura e descartar o arquivo.

3.4.7 O sistema *Phoenix*

O sistema *Phoenix* [Junqueira et al., 2003] é um sistema de *backup* remoto cooperativo que protege os dados armazenados contra catástrofes provenientes da Internet. Ele introduziu uma nova abordagem para sobreviver a epidemias provenientes da Internet, infestações de vírus, *worms* e outros, nomeadas de catástrofes pelo grande dano e prejuízo causado. Esta nova abordagem foi chamada de *informed replication*. Ela baseia-se no fato de que as epidemias exploram vulnerabilidades compartilhadas, ou seja, o mesmo sistema operacional ou web server, ou cliente de e-mail, etc. Assim, através da replicação do sistema por *hosts* que não compartilham as mesmas vulnerabilidades, o sistema *Phoenix* provê um alto grau de disponibilidade ao serviço fornecido.

O sistema *Phoenix* utilizou o substrato *Pastry* para seu serviço de DHT. O modelo de uso do sistema *Phoenix* é direto, ou seja, o sistema protegerá uma quantidade de dados proporcional à quantia que o usuário se dispuser a fornecer ao sistema.

Foi desenvolvido um modelo de sistema chamado *core abstraction* para representar a correlação de falhas em sistemas distribuídos. Um *core* foi definido como um subconjunto mínimo de componentes tal que a probabilidade de falha em todos os *hosts* de um core seja desprezível. Os atributos representam características dos *hosts* que os torna passíveis de falha, tais como seu sistema operacional e seus serviços de rede. Desde que os *hosts* normalmente tem muitas características que os tornam vulneráveis a falhas, estes atributos foram agrupados em configurações para representar o conjunto de vulnerabilidades de um determinado *host*. Um sistema pode usar as configurações de todos os seus *hosts* para determinar quantas réplicas são necessárias e em quais *hosts* elas devem ser armazenadas para sobreviver a epidemias.

O exemplo de sistema a seguir ilustra este conceito. Neste sistema os *hosts* são caracterizados por seis atributos, classificados em três categorias: sistema operacional, web server e web browser.

```
Exemplo de sistema:

        Atributos do sistema:
                Sistema Operacional: Unix, Windows
                Web Server: Apache, IIS
                Web Browser: IE, Netscape
        Hosts:
                H1 = Unix, Apache, Netscape
                H2 = Windows, IIS, IE
                H3 = Windows, IIS, Netscape
                H4 = Windows, Apache, IE
```

```
Cores:  {{H1 , H2}, {H1 , H3 , H4 }}
```

Para a seleção dos *cores* foi implementada uma nova heurística que teve como resultado uma garantia de confiabilidade tal que os dados dos usuários tem uma probabilidade de sobreviver a ataques simples e duplos (duas pragas distintas) maior que 0,99. O sistema *Phoenix* impõe uma baixa sobrecarga e requer no máximo três cópias para sobreviver a um ataque simples e no máximo 5 cópias para sobreviver a um ataque duplo.

3.4.8 O sistema *DIBS*

O sistema *DIBS* [Hsu et al., 2004] foi concebido para ser uma solução de baixo custo para a recuperação de dados críticos. Ele não visa o *backup* dos aplicativos ou sistemas operacionais, pois estes podem ser recuperados de forma mais eficiente. Este sistema assume que seus pares são semi-confiáveis, e que a comunicação se dá em uma LAN onde os pares são conhecidos entre si. Além disso, o sistema *DIBS* tem como principal objetivo de segurança a garantia da privacidade, através da proteção dos conteúdos e nomes dos arquivos com uso de criptografia. Como objetivos secundários de segurança, o sistema *DIBS* fornece autenticação e integridade dos arquivos.

De uma forma geral o sistema *DIBS* replica os arquivos dos usuários entre pares semi-confiáveis. Através de requisições aos seus pares, um usuário pode construir dados perdidos ou danificados. Cada nó do sistema *DIBS* mantém uma lista (lista de servidores) de outros nós que estão acessíveis no momento. Os nós do sistema constroem esta lista escutando mensagens que foram enviadas por *broadcast* de outros nós. Todos os nós, de forma periódica, fazem um *broadcast* de seus identificadores únicos, seus usuários, seus endereços IP e da quantidade de espaço que eles escolheram para oferecer para *backup*. Os nós também armazenam o tempo desde a última mensagem recebida de outros nós para poder determinar se um nó deixou a rede.

Um nó possui uma lista com os arquivos que ele está armazenando para outros e dos que ele está armazenando em outros. Cada entrada nesta lista é composta do caminho local do arquivo, seu proprietário, o dono da máquina e um *hash* MD5 do conteúdo do arquivo. Esta lista funciona como um *cache* do estado do sistema.

3.4.9 O sistema de *backup friend-to-friend*

Recentemente foram propostos sistemas de *backup friend-to-friend* (*f2f*) [Li and Dabek, 2006], nos quais o sistema *p2p* é formado por uma rede social ao invés de pares anônimos. Nesse tipo de sistema, a presença de pares maliciosos é minimizada, devido ao estabelecimento prévio de relações de confiança entre os pares através da rede social. As relações confiáveis permitem utilizar um nível mais baixo de replicação, o que implica em consumo menor de banda de comunicação e de área de armazenamento. Algumas implementações de sistemas de *backup f2f* utilizam a codificação por apagamento (*erasure coding*) [Plank, 1997] ao invés da replicação convencional, como meio de garantir a disponibilidade dos dados. A codificação por apagamento oferece uma maior disponibilidade dos dados armazenados e também consome menos banda de comunicação. No entanto, a reconstrução de um arquivo a partir da codificação por apagamento consome muito processamento; em sistemas com alta disponibilidade, a replicação torna-se então mais viável, conforme discutido

em [Rodrigues and Liskov, 2005, Oliveira et al., 2008]. Além da codificação por apagamento, para tolerar nós maliciosos, utilizam-se outras técnicas, como consultas periódicas a blocos aleatórios, visando certificar que os pares ainda detêm os arquivos, entre outras [Lillibridge et al., 2003].

Devido ao potencial dos pares desenvolverem táticas arbitrárias astuciosas para obter vantagens sem fornecer ao sistema igual parcela de seus recursos, não é suficiente verificar experimentalmente se um protocolo tolera uma coleção de ataques identificados pelo criador do protocolo. Faz-se necessário projetar protocolos que provavelmente irão atingir seus objetivos independentemente das estratégias que os nós possam tramar dentro do escopo do modelo.

3.4.10 O sistema *BAR*

O sistema *BAR – Bizantine-Altruistic-Rational* [Aiyer et al., 2005] foi concebido para operar em múltiplos domínios administrativos. Ele acomoda três classes de nós, Bizantinos, Altruístas (nós que executam o programa proposto, beneficiando-se dele ou não), e nós egoístas (que desenvolvem um comportamento ganancioso *greedy*). Provê garantias similares às dos protocolos tolerantes a faltas Bizantinas para todos os nós egoístas e altruístas. Para estender a tolerância a faltas Bizantinas para os nós egoístas, foram utilizadas ferramentas da teoria dos jogos tais como o *Nash Equilibrium* [Mailath, 1998], um protocolo para fornecer equilíbrio rígido onde todos os nós egoístas irão seguir o protocolo porque não têm nada a ganhar ao se desviar dele. Também foi proposto um protocolo para punir todos os nós egoístas por sua ganância, através da negação de acesso às funções do sistema que permitiriam que eles se beneficiassem dos serviços oferecidos pelo sistema (*Backup* distribuído).

3.5 Conclusão do Capítulo

Neste capítulo foram apresentados os conceitos de *backup* e as principais características dos sistemas de *backup* cooperativos. Em seguida, foram apresentados os trabalhos científicos mais recentes relacionados com a área de *backup* cooperativo. Todos os sistemas de backup cooperativos apresentados na seção anterior 3.4 podem ser classificados em relação à utilização do repositório como fornecendo *backup on-line* em conjunto com *backup off-line*. Na tabela 3.1 a seguir estes sistemas de backup cooperativos são classificados conforme exposto na seção 3.2.

Concluindo, o sistema *CBS* é o mais simples, o único com arquitetura centralizada. Já o sistema *Pastiche* é mais completo, incentivando trocas justas, além disto, busca tolerância a intrusões através de mecanismos probabilísticos. O sistema *Venti-DHash* utiliza um armazenamento completamente distribuído. O sistema *PeerStore* se diferencia de outros sistemas por separar a gerência de metadados da gerência de armazenamento de *backup*. Os sistemas *pStore* e *ABS* são inspirados nos sistemas de versionamento e propõem uma utilização melhor dos recursos. Porém o *pStore* trata o problema dos nós maliciosos através de replicação e da assinatura dos blocos de dados por seus donos, para impedir que um nó malicioso possa assumir a propriedade dos blocos de outros nós e eliminá-los ou alterá-los. O *ABS* preocupa-se sobretudo com o uso eficiente dos recursos, mas não com tolerância a intrusões. O sistema *Phoenix* apresentou uma abordagem para tolerância a intrusões inovadora, a *informed replication*. O sistema *DIBS* garante somente a privacidade dos dados armazenados, mas não considera ataques maliciosos contra o serviço. No sistema *friend-to-friend* os pares são formados por uma rede

Tabela 3.1: Características dos sistemas de *backup* cooperativos

Sistema	data	Modelo de repositório de dados	Utilização dos recursos	Integridade e consistência	Confidencialidade	privacidade	disponibilidade	sinergia	confiança	Tolerância a intrusão	principal característica
pStore	2001	incremental	single-storage instance, versionamento de arquivos	erasure codes[a]	criptografia de chave secreta		replicação	–	–	√	Replicação e assinatura dos blocos de dados por seus donos
Pastiche	2002	–	single-storage instance	convergent encription[b]	criptografia	–	periodic random challenges	distributed quota enforcement mechanism[c]	–	–	sistema de arquivos Chunkstore[d]
CBS	2003		–	erasure codes	criptografia de chave secreta	–	periodic random challenges	acordos entre pares	–	√	arquitetura centralizada
Phoenix	2003	–	informed replication	criptografia	criptografia	–	informed replication	–	–	–	Informed replication
Venti-DHash	2003	–	–	erasure codes	criptografia	–	–	–	–	√	guarda snapshots completos do sistema de arquivos
ABS	2004	–	single-storage instance, versionamento de arquivos, compressão	erasure codes	criptografia	–	periodic random challenges	–	–	√	versionamento baseado no Rsync
DIBS	2004	–	–	MD5, criptografia de chave secreta	criptografia de chave secreta	–	replicação	–	–	–	uso em LAN
PeerStore	2004	incremental	single-storage instance	–	criptografia	–	spot checks with a probabilistic punishment model[e]	symmetric trading scheme[f]	–	–	separação dos metadados dos dados para armazenamento
BAR	2005	–	compressão	replicated state machine protocol[g]	criptografia	–	erasure codes	periodic work protocol[h]	chave de criptografia publica	√	Nash Equilibrium
f2f	2006	–	–	–	–	–	redes de relacionamento	redes de relacionamento	redes de relacionamento	–	o sistema p2p é formado por uma rede social

[a] Ver [Plank, 1997].

[b] Permite que os hosts utilizem a mesma representação criptográfica para dados em comum sem compartilhar suas chaves.

[c] Mecanismo para garantir que um nó ocupe somente tanto espaço quanto o que ele contribui.

[d] Ver [Cox et al., 2002].

[e] Para cada verificação que o paraceiro falhar em responder, dentro de um período de tempo razoável, o par descarta um pequeno conjunto de des suas réplicas selecionadas de forma aleatória, com uma probabilidade exponencialmente crescente.

[f] Mecanismo para negociações de espaço de armazenamento.

[g] Protocolo para máquinas de estado replicadas baseados em Byzantine Fault Tolerance [Castro and Liskov, 2002].

[h] Sistemas cooperativos podem ter tarefas de manutenção que devem ser executadas periodicamente (periodic random challenges.). No entanto, pode não haver incentivos para que um nó execute estas tarefas. Com este protocolo (Periodic Work protocol), um nó pode verificar se esta tarefa tem sido feita.

social ao invés de pares anônimos. Finalmente o sistema *BAR* utilizou ferramentas da teoria dos jogos para fornecer equilíbrio rígido onde todos os nós egoístas irão seguir o protocolo porque não têm nada a ganhar ao se desviar dele. Dentre estes sistemas, alguns fornecem tolerância a intrusões através de *erasure codes* (*CBS, Venti-DHash, pStore e ABS*), já o sistema *BAR* provê tolerância a intrusões através de uma adaptação do protocolo de replicação de máquinas de estados, o BTF (*Byzantine Fault Tolerance* de [Castro and Liskov, 2002]). No próximo capítulo o tema tolerância a intrusões será apresentado com maior profundidade.

Capítulo 4

Tolerância a Intrusões

4.1 Introdução

No capítulo anterior foram apresentados os principais trabalhos na área de sistemas de *backup* baseados na tecnologia *p2p* e, ao final do qual, o assunto tolerância a intrusões foi introduzido. Neste capítulo a tolerância a intrusões será vista mais a fundo, iniciando pela sua conceituação e em seguida serão apresentadas tecnologias de replicação de máquinas de estados (RME), códigos de apagamento (*erasure codes*) e, por fim, os sistemas de quóruns bizantinos.

4.2 Segurança e Confiabilidade

Inicialmente os termos segurança e confiabilidade (*security e dependability*) devem ser compreendidos. Segurança está associado a prevenção de ações maliciosas para que estas não causem prejuízo a informação ou a prestação de serviços computacionais. As principais propriedades que a segurança pretende garantir são a confidencialidade, integridade e disponibilidade da informação e de serviços computacionais. Confiabilidade é a probabilidade de um sistema realizar e manter seu funcionamento de forma adequada, em circunstâncias de rotina, bem como em circunstâncias hostis e inesperadas, conforme especificado, durante um período de tempo pré-determinado. Ou seja, tanto a segurança quanto a confiabilidade tem como objetivo garantir que os sistemas computacionais funcionem corretamente.

4.3 Tolerância a Intrusões

O conceito de tolerância a intrusões foi introduzido por [Fraga and Powell, 1985] e, com a evolução deste tema, chegou-se a uma definição de serviços distribuídos tolerantes a intrusões. O objetivo destes consiste em garantir a integridade, disponibilidade e confidencialidade de serviços constituídos por diversos servidores ligados através de uma rede, mesmo que alguns desses servidores sejam atacados e controlados com sucesso por atacantes (hackers, crackers) ou por código nocivo (vírus, vermes, etc.) [Correia, 2005].

A tolerância a intrusões (TI) surge do encontro da segurança com a confiabilidade, e assim, aplicar a tolerância a faltas ao domínio da segurança [Veríssimo et al., 2003]. A TI assume e aceita que um sistema permanece sempre mais ou menos vulnerável, os componentes

do sistema podem ser atacados e que alguns desses ataques terão sucesso e garante que o sistema como um todo permanece seguro e operacional, ou seja, que não falha.

As intrusões, as vulnerabilidades e os ataques são faltas. Uma vulnerabilidade é uma falta de projeto ou de configuração, geralmente acidental, que pode ser explorada com fins maliciosos. Um ataque é uma falta intencional, maliciosa, que visa explorar uma ou mais vulnerabilidades. Uma intrusão é o resultado de um ataque que tem sucesso em explorar uma ou mais vulnerabilidades. As faltas maliciosas são consideradas como podendo ser de qualquer tipo, sendo englobadas na categoria de faltas mais geral: as faltas arbitrárias, também denominadas de faltas bizantinas [1], quando uma falta bizantina ocorre o sistema pode responder de uma forma imprevisível, a menos que este tenha sido projetado para ser tolerante a faltas bizantinas. Abrangem as faltas chamadas de "faltas por parada" e "faltas por omissão". Em tolerância a intrusões os termos intrusão e falta bizantina são usados geralmente como sinônimos [Correia, 2005].

Muitos dos trabalhos em serviços distribuídos tolerantes a intrusões são baseados em replicação. A replicação é muito usada em tolerância a faltas para garantir a disponibilidade e a confiabilidade de serviços distribuídos. A replicação consiste em distribuir cópias do código e dos dados de determinado serviço por um conjunto de servidores. Este tipo de solução permite garantir a disponibilidade e a integridade do serviço se houver intrusões num número limitado de réplicas. Os principais trabalhos nesta área podem ser classificados como os que fazem replicação de máquinas de estados [Castro and Liskov, 2002] e os que usam quóruns bizantinos [Malkhi and Reiter, 1997].

4.4 Replicação de Máquinas de Estados

A abordagem de replicação de máquinas de estados é um método genérico para a implementação de serviços tolerantes a faltas através da replicação dos servidores e da coordenação das interações dos clientes com estas réplicas. Um serviço oferece um conjunto de operações aos seus clientes, que os invocam através de pedidos. Um serviço e realizado através de um conjunto de réplicas. Cada servidor é uma máquina de estados, definida por variáveis de estado que definem o seu estado, e por comandos que modificam esse estado. Os comandos têm de ser atômicos, ou seja, não podem interferir uns com os outros. Todos os servidores seguem a mesma seqüência de estados, para o que é suficiente satisfazer quatro propriedades:

Estado inicial: Todos os servidores começam no mesmo estado.

Acordo: Todos os servidores executam os mesmos comandos.

Ordem total: Todos os servidores executam os comandos pela mesma ordem.

Determinismo: O mesmo comando executado no mesmo estado inicial gera o mesmo estado final.

Um parâmetro importante quando se fala de um serviço tolerante a faltas/intrusões é a resistência (*resilience*), é o número máximo de servidores que podem falhar para o serviço se manter correto. Em sistemas assíncronos TI baseados em replicação de máquinas de estado

[1] A denominação faltas bizantinas vem de um artigo clássico que apresenta um protocolo tolerante a faltas maliciosas através de um problema envolvendo generais bizantinos [Lamport86].

este limite é imposto pelo protocolo de difusão atômica, cuja resistência máxima é de que são necessários (pelo menos) $3f+1$ servidores para tolerar f servidores que falham [Correia, 2005].

4.5 Sistemas de Quóruns Bizantinos

Um sistema de quóruns bizantinos (BQS – *Byzantine Quorum System*) [Malkhi and Reiter, 1997] é definido como um sistema distribuído de armazenamento de dados replicados com garantias de consistência e disponibilidade, mesmo na ocorrência de faltas bizantinas em algumas de suas réplicas [Dantas et al., 2007].

Os trabalhos iniciais em sistemas de quóruns [Gifford, 1979] assumiam que os servidores falhavam de forma benigna, isto é, falhas por parada, por omissão e de desempenho. Pesquisas posteriores forneceram técnicas que permitiram a disponibilidade dos dados mesmo em presença de faltas arbitrárias. Os trabalhos mais recentes passaram a fornecer semânticas corretas mesmo em presença de faltas arbitrárias nos servidores e também alguns casos de clientes Bizantinos [Malkhi and Reiter, 1997].

Os sistemas de quóruns bizantinos não exigem a execução de acordo entre as réplicas para o seqüenciamento das suas operações. Isto significa que eles não são suscetíveis à impossibilidade *FLP* [2] [Fischer et al., 1985] e que podem ser implementados em sistemas assíncronos como a Internet. Assim a sua terminação fica garantida e, por isso, eles dispensam a implementação de protocolos de acordo para garantir a consistência entre réplicas.

4.5.1 Definição

Um sistema de quóruns tolerante a faltas bizantinas é implementado como um conjunto de subconjuntos de nós em um sistema distribuído. Cada sub-conjunto de nós é denominado um *quórum*, e faz interseção com todos os outros quóruns do sistema. Essa interseção tem propriedades especiais, pois ela define uma quantidade de servidores em comum, que garante que as transações feitas em quóruns diferentes mantenham a consistência do sistema (propriedade de consistência). Além disto, existe pelo menos um quórum que é formado somente por servidores corretos (propriedade de disponibilidade). O uso de quóruns é uma forma de aumentar a disponibilidade e eficiência em dados replicados, pois cada quórum pode agir em nome do sistema como um todo, provendo um ganho de disponibilidade e desempenho do sistema.

Os BQS também possuem bom desempenho e escalabilidade, porque os clientes acessam somente um quórum de servidores e não o sistema todo. Porém, eles somente suportam operações de escrita e leitura em seus registros, o que não representa uma limitação neste projeto. Cada registro guarda um valor e seu *time stamp* associado $\langle v,t\rangle$ que é definido no momento de sua escrita pelo cliente. Os registros também podem ser assinados pelo cliente, sendo então denominados *auto-verificáveis*. Assim, o cliente pode detectar modificações não-autorizadas do conteúdo dos registros por algum servidor malicioso ou corrupto.

[2] O consenso (um acordo geral entre os membros de um grupo ou comunidade) tem sido mostrado como impossível de ser solucionado em muitos modelos em computação distribuída. Em sistemas assíncronos, onde os processos não tem um relógio (*clock*) comum e executam a velocidades arbitrariamente variadas, o consenso é impossível de ser alcançado se um processo pode falhar por parada (*crash*) e os processos se comunicam através de trocas de mensagens. A técnica usada para provar este resultado é chamada algumas vezes de uma prova de impossibilidade FLP, recebendo este nome de seus criadores Michael J. Fischer, Nancy A. Lynch e Michael S. Paterson, que receberam o prêmio Dijkstra por este resultado.

Uma forma típica de configuração de sistemas de quóruns bizantinos, utilizando registros auto-verificáveis para sobreviver a f faltas é formar o sistema com $3f + 1$ réplicas com quóruns de tamanho $2f + 1$. A interseção entre os quóruns terá tamanho $f + 1$ e isso garantirá que quaisquer dois quóruns terão em sua interseção pelo menos uma réplica correta [Liskov and Rodrigues, 2006].

A figura 4.1 mostra uma representação formal de um sistema de quóruns Bizantinos $\mathbb{Q}$, onde Q_1, Q_2 e Q_3 representam um conjunto de quóruns e B um conjunto de nós passíveis de falhas.

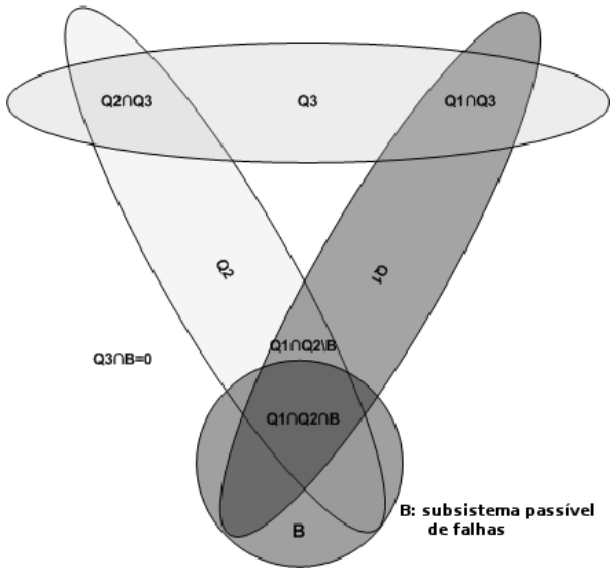

Figura 4.1: Representação de um sistema de quóruns Bizantinos.

Cada interseção contém uma quantidade suficiente de servidores corretos (por exemplo, o conjunto $Q_1 \cap Q_2 \setminus B$). Logo, se um cliente realizar duas operações em dois quóruns diferentes, necessariamente um mesmo grupo de servidores corretos será acessado, ainda que servidores Bizantinos sejam possivelmente acessados também (conjunto $B \cap Q_2$). A disponibilidade é garantida pela existência de, pelo menos, um quórum Q, onde todos os servidores são corretos (conjunto Q_3).

Os BQS permitem operações de leitura e escrita somente em um quórum dentre os servidores, desde que as propriedades das intersecções garantam que qualquer operação de leitura terá acesso aos dados mais recentes.

4.5.2 Operação

Cada cliente utiliza conjuntos distintos de *time stamps*. O protocolo de leitura tem usualmente uma única fase onde o cliente executa uma solicitação a um quórum de réplicas e retorna o valor com o maior *time stamp* fornecido com a assinatura válida, conforme mostrado na figura 4.2 a seguir. Uma extensão deste protocolo executa uma segunda fase na qual escreve de volta este valor em um quórum, isto garante uma semântica atômica.

São necessárias duas fases para a escrita dos dados, como mostrado na figura 4.3 a seguir. A primeira fase é igual a do protocolo de leitura. Primeiro o cliente contata um quórum para obter o maior *time stamp* produzido até então. O cliente então seleciona um *time stamp*

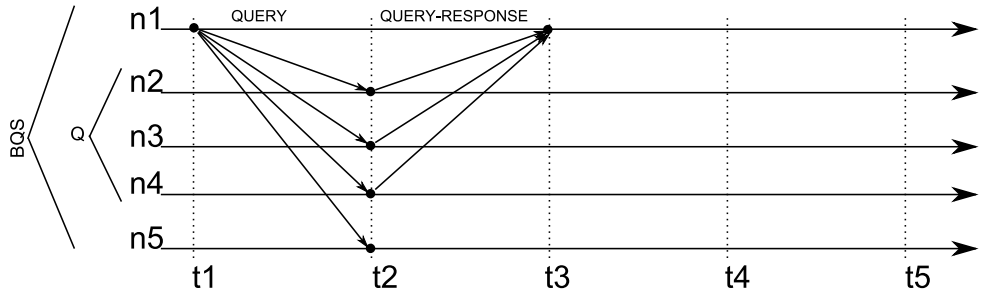

Figura 4.2: Fases para a leitura de dados em um sistema de quóruns Bizantinos [Dantas et al., 2007].

maior que o obtido na primeira fase, assina o novo valor e o *time stamp* e passa para a segunda fase, onde o novo valor é armazenado em um quórum de réplicas.

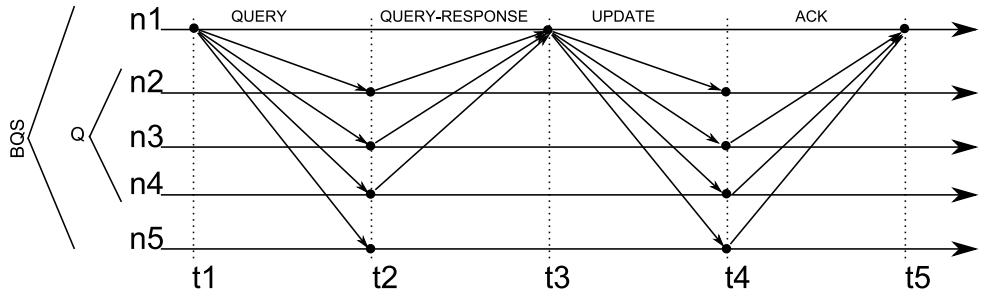

Figura 4.3: Fases para a escrita de dados em um sistema de quóruns Bizantinos [Dantas et al., 2007].

As réplicas permitem requisições de escritas somente de clientes autorizados. Uma réplica sobrescreve o que estava armazenado somente se o *time stamp* na requisição for maior que o armazenado.

Os quóruns de leitura e escrita podem ter tamanhos iguais ou diferentes (quóruns simétricos ou assimétricos). Pode haver diferentes níveis de acesso, quando somente um cliente acessa o sistema a semântica de acesso é chamada de "único escritor" caso sejam múltiplos clientes então a semântica de acesso é dita de "vários escritores". Também é possível classificar quanto à semântica de consistência. A semântica segura ocorre quando uma leitura com escrita concorrente pode retornar qualquer valor, caso contrário devolve o último valor escrito. A semântica regular garante a semântica segura e, quando houver uma leitura com escritas concorrentes retornará o último valor escrito ou um dos valores sendo escritos. Já a semântica atômica garante escritas e leituras dentro de uma semântica regular e a ordenação de leituras e escritas segundo uma relação causal, ou seja, uma leitura com escrita concorrente sempre irá retornar o último valor escrito.

Existem vários tipos de BQS, dentre eles podemos citar os sistemas *f-mascaramento*, *f-disseminação*, *a-mascaramento*, *a-disseminação* e *sistemas mínimos* [Dantas et al., 2007]. O sistema *f-mascaramento* armazena dados genéricos em quóruns simétricos, o *f-disseminação* armazena dados autoverificáveis em quóruns simétricos, *a-mascaramento* armazena dados genéricos em quóruns assimétricos, *a-disseminação* armazena dados autoverificáveis em quóruns assimétricos e os *sistemas mínimos* são em princípio um sistema *a-mascaramento*, porém, ao contrário destes, apenas os quóruns de escrita asseguram a propriedade de disponibilidade.

Existem vários algoritmos para escrita e leitura em BQS, estes algoritmos podem ser classificados em relação a vários pontos, tais como a organização dos quóruns, se os clientes são corretos e as propriedades de consistência envolvidas no armazenamento das réplicas. Como exemplo se pode citar o *MWMR seguro*, que descreve algoritmos de leitura e escrita em sistemas de quóruns *f-mascaramento* e a semântica de consistência alcançada é *multi-writer multi-reader* segura com clientes corretos. Assim como neste algoritmo, temos outros com clientes faltosos, *single-writer multi-reader*, semânticas regulares ou atômicas e também com tipos diferentes de BQS.

4.6 Conclusão

Enquanto a replicação de máquinas de estados (RME) é uma solução genérica para fornecer serviços tolerantes a faltas/intrusões, os quóruns geralmente são usados para construir repositórios de dados tolerantes a faltas/intrusões o que constitui um caso particular do RME. A principal diferença entre a RME e os sistemas de quóruns é que as operações na RME envolvem sempre todos os servidores, enquanto que nos sistemas de quóruns as operações são geralmente feitas sobre um quórum – um subconjunto dos servidores – o que torna os algoritmos mais escaláveis. No próximo capítulo será apresentada a proposta desta dissertação, um sistema de *backup* cooperativo tolerante a intrusões.

Capítulo 5

BackupIT – Um Sistema de *Backup* Distribuido Tolerante a Intrusões

5.1 Introdução

Os capítulos anteriores desta dissertação apresentaram os principais conceitos associados a sistemas de *backup*. Também foram descritas as bases da tecnologia *p2p*, e como esta pode ser usada para implementar sistemas de *backup* cooperativos com diversas vantagens sobre os sistemas convencionais. Este capítulo apresenta a proposta principal, que consiste em um sistema de *backup* cooperativo que usa o conceito de quóruns bizantinos para oferecer tolerância a intrusões.

5.2 Motivação

Como mencionado no início deste trabalho, a capacidade de armazenamento dos computadores duplica anualmente, no entanto esta mesma capacidade de armazenamento é consumida pela produção de novos dados. Como conseqüência, a necessidade de espaço para armazenar estes dados cresce na mesma proporção. Além disso, também foi citado que os sistemas *p2p* são uma das tecnologias em computação que mais cresce, o que pode ser verificado nos relatórios sobre o crescimento do tráfego *p2p* na Internet. Os sistemas de quóruns podem aumentar a disponibilidade e eficiência no armazenamento dos dados replicados em sistemas *p2p* através da execução de leituras e escritas de dados em diferentes conjuntos de servidores (quóruns) que mantêm réplicas (servidores) em comum.

5.3 A Proposta

O BacukpIT foi projetado visando usar os recursos disponíveis em uma *intranet*, isto é, um conjunto de redes de computadores sob o controle de uma única entidade administrativa, para armazenar cópias de segurança dos dados de seus usuários de forma colaborativa. Assim, cada nó dessa rede armazenará os *backups* de outros nós da mesma rede. Dessa forma, se obtém um uso mais racional do espaço em disco disponível e da capacidade de processamento e de comunicação do sistema. Além disso, este sistema oferece disponibilidade, tolerância a intrusões e consistência, pelo uso de replicação dos dados entre os nós da rede. A replicação é feita

usando sistemas de quóruns bizantinos que, por serem completamente distribuídos, também evitam problemas de ponto único de falha.

5.4 Arquitetura do Sistema

A arquitetura proposta consiste em um conjunto de nós interligados por uma rede *p2p* sobreposta que se utiliza de protocolos de BQS para dar suporte a operações de *backup* conforme exemplificado na figura 5.1 a seguir. Nesta figura temos *N* nós (n, n1, n2, n3...) interligados por uma *DHT* (representada pelo círculo). Os nós formam conjuntos que são os sistemas de quóruns bizantinos, por exemplo *BQS1* e *BQS2*, que serão tantos quantos forem a quantidade de nós no sistema (*N*), cada nó forma seu *BQS* a partir do seu *conjunto-folha*, informado pelo serviço de localização e roteamento. Os *BQS* se sobrepõem, como pode ser visto no exemplo da figura 5.1 a seguir.

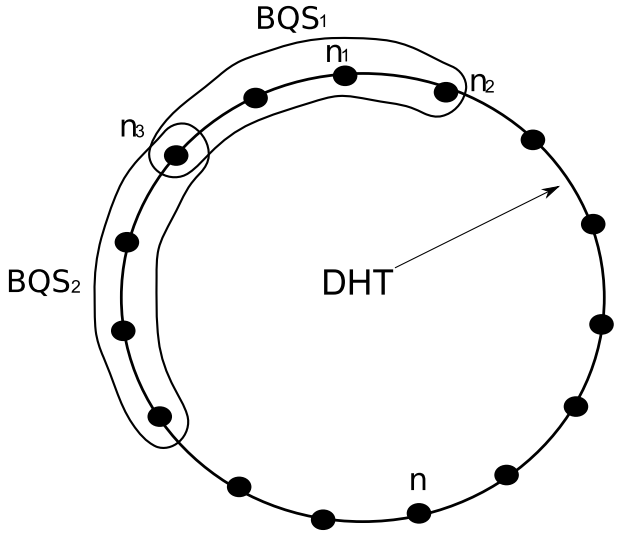

Figura 5.1: Distribuição dos nós e BQSs pelo sistema *BackupIT*.

Todos os nós do sistema têm as mesmas funções e interagem de forma cooperativa, formando uma rede *p2p* pura (não hierárquica). Cada nó do sistema é responsável por responder a requisições, armazenar e recuperar réplicas de/para outros nós e fornecer uma interface aos usuários. Cada nó tem cinco componentes locais: a *Gerência de Backup*, o *Serviço de Quórum*, o *Serviço de Localização e Roteamento*, o *Serviço de Compressão* e o *Serviço de Criptografia*, conforme apresentado na figura 5.2.

A *Gerência de backup* é responsável por tratar as solicitações de *backup* e de recuperação de arquivos vindas do usuário local. Ela é responsável por gerar assinaturas (*hashes* criptográficos) dos arquivos a armazenar e verificar sua integridade quando estes forem recuperados. O *Serviço de Quórum* provê tolerância a intrusões, disponibilidade, integridade e consistência de dados. Esse serviço é detalhado na seção 5.4.1 a seguir.

O *Serviço de Localização e Roteamento* é responsável por gerar os identificadores únicos de nós e de objetos (arquivos), pela localização de identificadores, pelo roteamento de mensagens e também por definir o grupo de nós que formam o sistema de quóruns de cada nó. Esse serviço é prestado pelo ambiente Pastry apresentado na seção 6.2. É importante observar

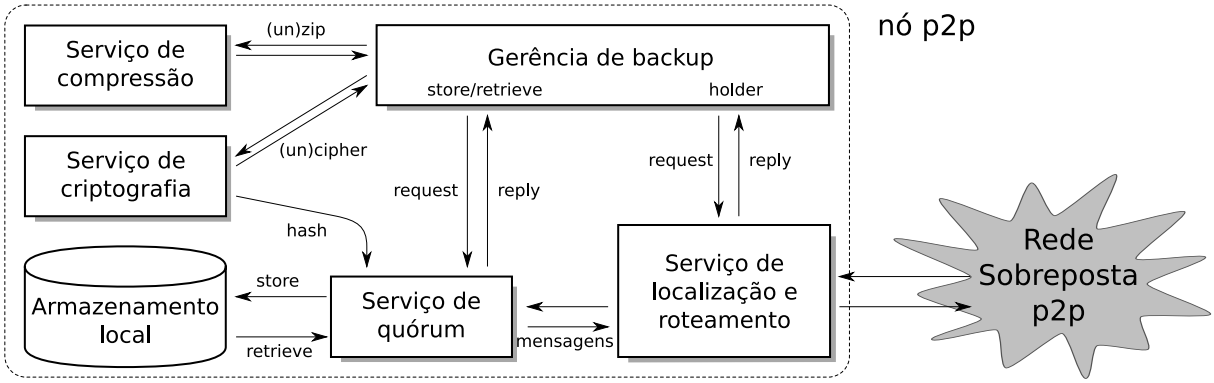

Figura 5.2: Arquitetura proposta para o sistema de *backup* cooperativo tolerânte a intrusões *BackupIT*.

que, uma vez localizado um nó, toda a comunicação subseqüente com ele é feita de forma direta, principalmente durante as transferências de arquivos.

O *Serviço de Compressão* tem por finalidade prover o uso eficiente do espaço de armazenamento e da banda de comunicação, através da compressão e descompressão dos arquivos a serem transferidos e armazenados. O *Serviço de Criptografia* visa garantir a confidencialidade e integridade dos dados. Antes de enviar um arquivo para armazenamento em seu sistema de quóruns, o nó o cifra, usando criptografa simétrica com uma chave secreta conhecida somente por ele.

Para evitar o consumo excessivo do espaço de armazenamento por dados obsoletos, foi adotada uma política de prazo de validade: cada arquivo armazenado tem um período de validade definido por seu proprietário. A preservação do arquivo é garantida durante esse período de tempo, após o qual ele poderá ser automaticamente removido do sistema de *backup*.

5.4.1 O Serviço de Quórum

Neste projeto foi utilizado o sistema de quórum do tipo *f-disseminação* descrito em [Malkhi and Reiter, 1997]. Esse sistema de quóruns trabalha com dados auto-verificáveis e quóruns simétricos (quóruns de escrita e de leitura com mesmo tamanho). Este tipo de sistema de quórum pode ser construído com um número menor de nós e seus protocolos demandam um número reduzido de trocas de mensagens, se adequando bem ao ambiente de uma *intranet*.

Ainda segundo [Malkhi and Reiter, 1997], a semântica de consistência alcançada por esse sistema de quóruns é regular, ou seja, se não houverem escritas concorrentes, a operação de leitura de um registro retornará o último valor escrito nele; caso contrário, retornará algum dos valores sendo escritos. Como sistemas de *backup* normalmente têm uma semântica de acesso de escritor único (pois somente um cliente irá executar escritas em uma determinada chave), não ocorrerão escritas concorrentes.

Cada nó p_i do sistema de *backup* cooperativo constrói um sistema de quóruns $\mathbb{S}_i$, com tamanho $|\mathbb{S}_i| \geq 3f + 1$ nós, onde f é o limite máximo de nós faltosos, definido durante a formação do sistema de quóruns. Os nós que compõem o sistema de quóruns de um nó são definidos a partir de seu *conjunto-folha* $\mathbb{L}_i$, informado pelo serviço de localização e roteamento. Os sistemas de quóruns dos diferentes nós se sobrepõem; cada nó do sistema será portanto membro de vários sistemas de quóruns simultaneamente. Isto é, se o sistema de quóruns tem

$3f+1$ membros dentre o total de nós do sistema de *backup*, cada nó possui outros $3f$ nós em seu sistema de quórum; se todos os sistemas de quóruns são distintos, então cada nó participará de $3f$ sistemas de quóruns distintos.

5.4.2 Modelo de Sistema

Assumimos um modelo de sistema assíncrono. O sistema de *backup* cooperativo é composto por um conjunto $\mathbb{P}$ com N nós $p_1 \dots p_N$. Considera-se f o limite de nós faltosos tolerados pelo sistema, com $3f+1 \leq N$. Cada nó p_i é identificado unicamente e localizado no sistema através de um identificador de nó $nodeId_i$. O conjunto-folha de um nó p_i, fornecido pelo substrato de localização e roteamento, é indicado como $\mathbb{L}_i$. Cada nó p_i constrói um sistema de quóruns bizantinos $\mathbb{S}_i$ com $3f+1$ nós, usando os $3f$ primeiros nós de seu conjunto-folha $\mathbb{L}_i$, além dele próprio. Um quórum $\mathbb{Q}_i^*$ é qualquer subconjunto desse sistema de quóruns bizantinos $\mathbb{S}_i$ com $2f+1$ nós, ou seja, $\mathbb{Q}_i^* \subset \mathbb{S}_i$ e $|\mathbb{Q}_i^*| = 2f+1$.

Um arquivo a ser armazenado no sistema de *backup* cooperativo é indicado como x, sendo $name(x)$ seu nome e $size(x)$ seu tamanho. Finalmente, considera-se que $hash(x)$ é uma função de assinatura criptográfica, $zip(x)$ é uma função de compressão de dados e $\{x\}_k$ representa a cifragem de x usando uma chave de criptografia simétrica k (cada nó p_i possui sua própria chave de criptografia k_i, mantida secreta).

Como modelo de falhas, assume-se que até f nós podem desviar de suas especificações, apresentando faltas bizantinas. Processos faltosos podem parar, omitir envio ou entrega de mensagens, enviar respostas incorretas e inundar outros processos com mensagens falsas. Porém, com o uso de assinatura criptográfica dos arquivos, as mensagens incorretas são detectáveis. Ataques por inundação não são tratados neste projeto. Assume-se também que o substrato de roteamento e localização é tolerante a faltas bizantinas. As faltas são consideradas independentes, ou seja, a probabilidade da ocorrência de uma falta em um nó é independente da ocorrência de uma falta em outro nó. As faltas por parada ou omissão e as perdas de mensagens são tratadas pelo sistema de quóruns bizantinos, que por sua construção tolera faltas destes tipos. Foram inseridos *time-outs* nos algoritmos para otimização do sistema, assim o cliente não precisa enviar a mensagem para todo o sistema de uma só vez, ele envia para um quórum de servidores e, após um determinado tempo, envia a cada novo *time-out* mais uma mensagem. A ocorrência de um *time-out* não define uma falha, somente significa um atraso (o que é correto pois a rede é considerada assíncrona). Assim se os pares responderem a tempo, haverá economia de banda. Finalmente, se for detectado algum nó desviando de sua especificação durante a execução do sistema, uma notificação de erro é gerada, permitindo intervenção externa.

5.4.3 Armazenamento de um Arquivo

O comportamento de um nó p_i para armazenar um arquivo x no sistema de *backup* é indicado no algoritmo 1. Inicialmente, p_i deve gerar uma chave (identificador único) para o arquivo, que será usada para seu armazenamento e localização futura. Para garantir a unicidade dessa chave, é usado o *hash* do identificador de nó $nodeId_i$ concatenado ao nome do arquivo (linha 1). A seguir, o serviço de localização e roteamento (*SLR*) é consultado para identificar o nó p_j responsável pela chave do arquivo, ou seja, o nó p_j cujo identificador $nodeId_j$ seja o mais próximo da chave do arquivo (linha 2). Esse nó p_j irá responder à consulta informando a p_i seu sistema de quóruns $\mathbb{S}_j$ (linha 3).

Na seqüência, p_i compacta e cifra o arquivo a armazenar usando sua chave de criptografia k_i (linha 4). O nó p_i deve então enviar o arquivo comprimido e cifrado (x') a pelo menos $2f+1$ nós p_k pertencentes a $\mathbb{S}_j$ (linha 8). Cada nó p_k retorna como resposta um *hash* h_k do arquivo recebido (linha 9). Se esse *hash* recebido coincidir com o *hash* do arquivo enviado, a resposta é considerada correta (linhas 10 e 16). Caso contrário, ou se p_k não responder dentro do prazo (*time-out*), é registrado um erro (linha 11). O nó p_i deve obter pelo menos $2f+1$ respostas corretas (linha 7), ou um número de erros maior que f (linha 12). Esta verificação de *hash* tem por finalidade detectar corrupções ou perdas de mensagens em circulação.

Algoritmo 1 Nó p_i armazena um arquivo x

```
 1: key ← hash(nodeId_i : name(x))
 2: envia holderReq(key) para o SLR
 3: recebe holderReply(S_j) de p_j
 4: x' ← {zip(x)}_{k_i}
 5: C = φ   // conjunto de nós com resposta correta
 6: E = φ   // conjunto de nós com erro
 7: while |C| < 2f + 1 do
 8:    envia storeReq(x') para p_k ∈ S_j
 9:    recebe storeReply(h_k) de p_k ou time-out
10:    if time-out ∨ (h_k ≠ hash(x')) then
11:       E ← E ∪ {p_k}
12:       if |E| > f then
13:          Erro: mais que f nós faltosos
14:       end if
15:    else
16:       C ← C ∪ {p_k}
17:    end if
18: end while
```

Deve-se observar que este não é exatamente o mesmo algoritmo proposto em [Malkhi and Reiter, 1997] para sistemas de quóruns f-disseminação com semântica regular *multi-writer/multi-reader*. Neste algoritmo foi introduzido um passo a mais, onde o cliente compara o *hash* calculado pelos servidores ao receber o arquivo com o *hash* local (linha 10). Este passo a mais tem a função de informar ao cliente que não houve nenhum problema na transação e o arquivo foi corretamente recebido pelos servidores. O cliente deve enviar o arquivo para outros servidores até completar um quórum ou constatar que o limite de faltas tolerado foi ultrapassado; neste caso, o sistema pode estar comprometido, sendo necessárias medidas de recuperação externas.

5.4.4 Recuperação de um Arquivo

O procedimento de recuperação de um arquivo x previamente armazenado está indicado no algoritmo 2. Inicialmente, o nó p_i deve localizar, através do substrato Pastry, o nó p_j responsável pela chave desse arquivo e obter seu sistema de quóruns $\mathbb{S}_j$, de forma similar ao algoritmo anterior (linhas 1 a 3 do algoritmo 2). A seguir, o nó p_i solicita o *hash* do arquivo identificado pela chave *key* a um quórum de nós pertencentes ao sistema de quóruns bizantinos

de p_j (linhas 4 a 8). O nó p_i deve então aguardar as respostas desses nós. Para cada resposta recebida, deve verificar se o hash h_k coincide com o hash de x previamente armazenado (linhas 9 a 12). Se algum nó não responder, o nó p_i deve escolher outros nós de $\mathbb{S}_j$ para solicitar o *hash* (linhas 13 a 16).

Uma vez recebidas $2f + 1$ respostas, o nó p_i solicita o arquivo a um dos nós que responderam corretamente, verifica se o *hash* do arquivo corresponde ao esperado e devolve o arquivo ao usuário (linhas 23 a 29). Caso contrário, um novo nó deve ser consultado. De acordo com o modelo de quóruns [Malkhi and Reiter, 1997], se não houverem mais de f faltas, o algoritmo irá recuperar corretamente o arquivo desejado.

Algoritmo 2 Nó p_i quer recuperar um arquivo x

1: $key \leftarrow hash(nodeId_i : name(x))$
2: envia $holderReq(key)$ para o SLR
3: recebe $holderReply(\mathbb{S}_j)$ de p_j
4: $\mathbb{P} \leftarrow \phi$ // conjunto de nós consultados
5: **for all** $p_k \in \mathbb{Q}_j^* \subset \mathbb{S}_j$ **do**
6: Envia $hashReq(key)$ a p_k
7: $\mathbb{P} \leftarrow \mathbb{P} \cup \{p_k\}$
8: **end for**
9: $\mathbb{R} \leftarrow \phi$ // conjunto de nós que responderam
10: $\mathbb{T} \leftarrow \phi$ // conjunto de nós com *time-out*
11: **while** $(|\mathbb{R}| < 2f + 1) \wedge (|\mathbb{T}| \leq f)$ **do**
12: recebe $hashReply(h_k)$ de $p_k \in \mathbb{P}$ ou *time-out*
13: **if** *time-out* **then**
14: $\mathbb{T} \leftarrow \mathbb{T} \cup \{p_k\}$
15: Envia $hashReq(key)$ a $p_m \in (\mathbb{S}_j - \mathbb{P})$
16: $\mathbb{P} \leftarrow \mathbb{P} \cup \{p_m\}$
17: **else**
18: $\mathbb{R} \leftarrow \mathbb{R} \cup \{p_k\}$
19: **end if**
20: **end while**
21: $\mathbb{C} \leftarrow \{p_k \in \mathbb{R} \mid h_k = hash(x)\}$ // Conjunto de respostas corretas
22: **while** $\mathbb{C} \neq \phi$ **do**
23: Envia $retrieveReq(key)$ a $p_k \in \mathbb{C}$
24: recebe $retrieveReply(x_k)$ de p_k ou *time-out*
25: **if** $time\text{-}out \vee (hash(x_k) \neq hash(x))$ **then**
26: $\mathbb{C} \leftarrow \mathbb{C} - \{p_k\}$
27: **else**
28: Decifra, descompacta e devolve x_k ao usuário
29: Fim do algoritmo
30: **end if**
31: **end while**
32: **Erro:** nenhuma resposta correta

5.4.5 Tratamento de Requisições

O comportamento de um nó p_i tratando requisições para armazenar ou recuperar um arquivo x no sistema de *backup* é bastante simples, como pode ser visto no algoritmo 3.

- O nó p_i, ao receber uma mensagem de p_j solicitando a identificação do nó responsável por uma determinada chave *key*, e sendo ele o responsável, responde a esta consulta informando a p_j seu sistema de quóruns $\mathbb{S}_i$ (linhas 1 e 2).
- Se p_i receber uma mensagem de p_j para armazenamento de um arquivo x_j, então ele deve armazenar este arquivo em sua área de armazenamento local e retornar como resposta o *hash* do arquivo recebido $h(x_j)$ (linhas 4 e 5).
- Se a mensagem recebida de p_j solicitar o *hash* do arquivo identificado por uma determinada chave *key*, então p_i deve calcular e retornar o hash deste arquivo $h(x_{key})$ (linhas 7 e 8).
- Finalmente, se p_i receber uma mensagem de p_j solicitando o arquivo identificado por uma determinada chave *key*, então p_i deve retornar o arquivo x_{key} a p_j (linhas 10 e 11).

Algoritmo 3 Nó p_i agindo como servidor.

```
1:  if recebe holderReq(key) de p_j then
2:      envia holderReply(S_j) para p_j
3:  end if
4:  if recebe storeReq(x_j) de p_j then
5:      envia storeReply(h_j) para p_j
6:  end if
7:  if recebe hashReq(key) de p_j then
8:      envia hashReply(h_key) para p_j
9:  end if
10: if recebe retrieveReq(key) de p_j then
11:     envia retrieveReply(x_key) para p_j
12: end if
```

5.4.6 Confiabilidade do sistema de *backup – BackupIT*

Como já dito antes, a confiabilidade é a probabilidade de um sistema realizar e manter seu funcionamento de forma adequada, em circunstâncias de rotina, bem como em circunstâncias hostis e inesperadas, conforme especificado, durante um período de tempo pré-determinado. Daí, pode-se dizer que a confiabilidade, no caso do sistema *BackupIT*, pode ser definida como sendo a probabilidade de não haver mais do que f nós faltosos pertencientes a um mesmo sistemas de quóruns bizantinos (*BQS*) do sistema de *backup*.

O número de nós no *BackupIT* não tem relação direta com o tamanho dos *BQS* que são criados a partir da definição do limite máximo de nós faltosos f do sistema. Sendo que a única restrição é o tamanho mínimo do sistema deve ser maior ou igual a $3f + 1$ nós. Assim, o *BackupIT* pode ter muitos nós e estes nós pertencerão a um ou mais *BQS*, dependendo do

tamanho do sistema e do valor de f. Como exemplificado na figura 5.1, o nó $n3$ pertence a dois *BQS* diferentes, *BQS* 1 e *BQS* 2, já os nós $n1$ e $n2$ pertencem ao mesmo *BQS* (*BQS* 1). Para que haja uma falha no *BackupIT* e ele seja considerado inseguro o número de nós faltosos dentro de um *BQS* deve ser superior a f, isto é, igual ou maior que $f + 1$.

A probabilidade de um nó falhar é dada por:

$$\mathbb{P}_n = \frac{b}{s} \tag{5.1}$$

onde:
Tamanho do sistema de quóruns bizantinos: $b = 3f + 1$;
Tamanho do sistema *BackupIT*: $\mathbb{S} = s$ nós;

Da teoria de probabilidade temos que a probabilidade de um nó n' e um nó n'' pertencerem a um mesmo *BQS* é dada pela probabilidade conjunta de dois eventos não independentes. Então, a probabilidade ($\mathbb{P}$) de que $f + 1$ nós do *BackupIT* ($\mathbb{S}$), com tamanho s nós, pertencerem a um mesmo *BQS* de tamanho $3f + 1$ nós, é dada pela fórmula:

$$\mathbb{P} = \frac{b^j}{\mathbb{A}^s_j} \tag{5.2}$$

onde:
$j = f + 1$;
$b = 3f + 1$;
$\mathbb{A}^s_j$ é o arranjo de s nós, tomados j a j;

Conseqüentemente a confiabilidade $\mathbb{C}$ deste sistema *BackupIT* é dada por:

$$\mathbb{C} = 1 - \mathbb{P} \tag{5.3}$$

Nas tabelas 5.1, 5.2, 5.3 e 5.4 a seguir são mostrados a variação da confiabilidade para diferentes tamanhos de sistemas (20, 100, 1000 e 10000 nós), com a variação do limite máximo de nós faltosos:

Tabela 5.1: Probabilidade de falha do sistema *BackupIT* para um sistema de 20 nós em relação ao limite máximo de nós faltosos.

tamanho do BQS (b)	quantidade de nós bizantinos no sistema (j)	porcentagem de nós bizantinos no sistema (%)	confiabilidade $\mathbb{C}$ (%)
4	**2**	**10,0**	**95,789**
7	3	15,0	94,985
10	4	20,0	91,400
13	5	25,0	80,043
16	6	30,0	39,882

Das tabelas 5.1, 5.2, 5.3 e 5.4 vemos que o sistema *BackupIT* pode alcançar um nível de confiabilidade superior a cinco noves (99,999%), dependendo da relação entre seu o tamanho e o seu limite máximo de nós faltosos, porém o tamanho mínimo do sistema deve ser igual ou superior a 90 nós, abaixo deste valor os cinco noves não são atingidos.

Tabela 5.2: Probabilidade de falha do sistema *BackupIT* para um sistema de 100 nós em relação ao limite máximo de nós faltosos.

tamanho do BQS (b)	quantidade de nós bizantinos no sistema (j)	porcentagem de nós bizantinos no sistema (%)	confiabilidade $\mathbb{C}$ (%)
4	2	2,0	99,838
13	5	5,0	99,996
19	**7**	**7,0**	**99,999**
49	17	17,0	99,998
55	19	19,0	99,993
61	21	21,0	99,970
67	23	23,0	99,845
73	25	25,0	98,982
79	27	27,0	91,754
85	29	29,0	18,190

Tabela 5.3: Probabilidade de falha do sistema *BackupIT* para um sistema de 1000 nós em relação ao limite máximo de nós faltosos.

tamanho do BQS (b)	quantidade de nós bizantinos no sistema (j)	porcentagem de nós bizantinos no sistema (%)	confiabilidade $\mathbb{C}$ (%)
4	2	0,2	99,998
811	**271**	**27,1**	**100,000**
841	281	28,1	99,338
844	282	28,2	97,886
847	283	28,3	93,218
850	284	28,4	78,132
853	285	28,5	29,142

Tabela 5.4: Probabilidade de falha do sistema *BackupIT* para um sistema de 10000 nós em relação ao limite máximo de nós faltosos.

tamanho do BQS (b)	quantidade de nós bizantinos no sistema (j)	porcentagem de nós bizantinos no sistema (%)	confiabilidade $\mathbb{C}$ (%)
8491	**2831**	**28,3**	**100,000**
8521	2841	28,4	99,852
8524	2842	28,4	99,521
8527	2843	28,4	98,450
8530	2844	28,4	94,979
8533	2845	28,5	83,725
8536	2846	28,5	47,216

5.4.7 Número de mensagens

Dos algoritmos apresentados podemos verificar que o número de mensagens e a quantidade de dados que transitam no sistema são proporcionais ao tamanho dos quóruns, ou seja, a f.

No algoritmo 1, para cada operação de armazenamento, temos uma mensagem *holderReq*() e sua resposta *holderReply*(), mais $(2f+1)$ mensagens *storeReq*() com sua respectiva resposta *storeReply*(), não havendo a ocorrencia de *time-out* ou hash errados a quantidade mínima de mensagens será $4f+4$. No caso de *time-out* deve-se acrescentar mais f mensagens ou no caso de resposta com hash errado (pior caso) acrescenta-se mais $2f$ mensagens perfazendo um total de $6f+4$ mensagens no pior caso. Para o algoritmo 2 a análise é similar.

A Tabela 5.5 indica os números mínimos e máximos de mensagens trocadas no sistema e relação ao limite de nós faltosos, para os processos de armazenamento e de recuperação de um arquivo. Comparando os valores apresentados na Tabela 5.5 com o tamanho do sistema de quóruns ($3f+1$ nós), verifica-se que aproximadamente duas mensagens por nó são trocadas no sistema para armazenar ou recuperar um arquivo.

Tabela 5.5: Números mínimos e máximos de mensagens trocadas no sistema *BackupIT* em relação ao limite de nós faltosos, para os processos de armazenamento e de recuperação de um arquivo.

Procedimento	melhor caso	pior caso
Armazenamento	$4f+4$	$6f+4$
Recuperação	$4f+6$	$6f+8$

A Tabela 5.6 apresenta uma estimativa do volume de dados trocados entre os nós do sistema em ambos os procedimentos. Para simplificar o cálculo, são consideradas somente as mensagens que transferem arquivos (*storeReq* e *retrieveReply*), pois as demais mensagens são geralmente muito pequenas e podem ser desprezadas. São considerados o pior e o melhor caso em ambos os procedimentos.

Tabela 5.6: Estimativa do volume de dados trocados entre os nós do sistema *BackupIT* para os processos de armazenamento e de recuperação de um arquivo.

Procedimento	melhor caso	pior caso
Armazenamento	$2f+1$	$3f+1$
Recuperação	1	$2f+1$

5.5 Escopo e Limitações

O sistema proposto tem como objetivo definir, construir e avaliar um sistema de *backup* cooperativo que faz uso da tecnologia de sistemas de quóruns bizantinos para oferecer tolerância a intrusões. Sendo que, o foco principal da proposta é a tolerância a intrusões às quais os sistemas de *backup* cooperativos estão sujeitos por serem utilizados em ambientes como a Internet. Os sistemas de quóruns bizantinos são uma forma de agregar disponibilidade e eficiência aos dados replicados em sistemas *p2p*.

Um sistema de *backup* cooperativo deve fornecer serviços para garantir: a confidencialidade e privacidade aos seus usuários; integridade, consistência e disponibilidade de dados; sinergia dos recursos e gerência de confiança aos pares. Pelo fato deste sistema ter sido planejado para ser utilizado em ambiente de Intranet, a gerência de confiança torna-se desnecessária, pois todos os pares estão sob um mesmo domínio administrativo, tornando os comportamentos

egoistas menos prováveis de ocorrer. Além do mais, a sinergia de recursos é muito maior neste caso, pois, como todos os pares estão sob um mesmo domínio administrativo, todos têm interesses em comum e devem colaborar entre si com boa vontade. Para implementar um sistema de *backup* que atenda a todos estes requisitos é necessário modelar todas as operações e situações possíveis para o sistema, o que não é o escopo deste trabalho. Assim, o sistema proposto tem seu foco em tolerância a intrusões através do uso de sistemas de quóruns bizantinos.

Além disto, o sistema proposto possui algumas limitações que poderão ser suprimidas no futuro. Uma delas é que a detecção de faltas ocorre somente quando o cliente precisa recuperar os seus *backups* do sistema. Se neste momento não houverem mais réplicas em quantidade suficiente para garantir a consistência dos dados o cliente pode ter perdido seus dados. Para solucionar esta limitação faz-se necessário implementar uma gerência de manutenção de réplicas de forma a garantir a disponibilidade e consistência a qualquer momento e não somente no momento da recuperação dos dados.

5.6 Conclusão

Foi apresentado neste capítulo o BackupIT, um sistema de *backup* cooperativo tolerante a intrusões que faz uso da tecnologia de sistemas de quóruns bizantinos. A proposta do BackupIT descreve a arquitetura do sistema, os algoritmos propostos, o escopo e suas limitações.

No próximo capítulo são apresentados os resultados obtidos com as experimentações do protótipo que podem ser consideradas como prova de conceito, bem como uma descrição de como foi implementado o protótipo.

Capítulo 6

Implementação e Avaliação do Protótipo

6.1 Introdução

Com a finalidade de avaliar a capacidade do sistema de *backup* cooperativo tolerante a intrusões proposto de tolerar faltas bizantinas foi implementado um protótipo para o BackupIT. Este protótipo fornece somente as funcionalidades necessárias para avaliar o sistema proposto.

A seguir é apresentada uma descrição do Pastry, o substrato selecionado para fornecer os serviços de localização e roteamento do sistema. Na sessão 6.3 é descrito como o protótipo foi implementado e como funciona. Finalmente, os experimentos e seus resultados são apresentados e discutidos.

6.2 O Ambiente Pastry

O Pastry [Rowstron and Druschel, 2001a] é um substrato para sistemas *peer-to-peer* que fornece serviços de localização e roteamento. Ele forma uma rede sobreposta robusta e auto-organizada, que pode ser usada por aplicações distribuídas. Qualquer nó que execute o Pastry com as credenciais apropriadas pode participar da rede sobreposta. É completamente descentralizado, resistente a faltas, escalável, confiável e tem boas propriedades de roteamento de mensagens.

Nós e objetos recebem identificadores únicos escolhidos de forma aleatória em um espaço de 128 bits. Esses identificadores são chamados respectivamente de *nodeId* e *key*. O Pastry roteia eficientemente uma mensagem para o nó cujo *nodeId* seja numericamente o mais próximo a uma determinada chave; o número de passos para o roteamento da mensagem é $O(\log N)$, onde N indica a quantidade de nós Pastry na rede [Haeberlen et al., 2005].

Cada nó Pastry mantém uma lista de nós imediatamente vizinhos dentro do espaço de identificadores e avisa a aplicação sobre nós que entram ou saem da rede sobreposta. Pelo fato dos *nodeIds* serem designados de forma aleatória, existe grande probabilidade de que o conjunto de nós com *nodeIds* adjacentes seja diversificado em relação a geografia, propriedade, jurisdição, etc. Uma heurística assegura que, entre um conjunto de nós com *nodeIds* próximos a uma determinada chave, seja provável que uma mensagem alcance primeiro um nó fisicamente próximo ao nó do qual a mensagem se originou.

A informação de roteamento mantida por cada nó consiste em um *conjunto-folha* (*leaf set* $\mathbb{L}$) e uma *tabela de roteamento*. Cada entrada nessas tabelas é denominada um *node handle*

e consiste em um *nodeId* e seu respectivo endereço IP. O conjunto-folha contém $|\mathbb{L}|/2$ *nodeIds* vizinhos de cada lado do *nodeId* do nó local, dentro do espaço de identificadores (normalmente $|\mathbb{L}|$ vale 32 ou 64 entradas), ordenados por proximidade de identificadores. Os nós do conjunto-folha são normalmente utilizados para guardar réplicas dos objetos enviados pela aplicação. O Pastry atualiza a informação de roteamento quando nós ingressam ou partem da rede sobreposta. As entradas defeituosas (nós que falharam ou saíram) são substituídas por outros nós, de forma a manter os custos de roteamento logarítmicos [Rowstron and Druschel, 2001a].

A interface de programação (API) do Pastry define várias funções, das quais as mais relevantes para este trabalho são:

- `pastryInit(Credenciais, Aplicação)`: faz com que o nó executante se associe a uma rede Pastry (ou inicie uma nova rede), inicializa o estado interno do nó e define seu identificador único na rede (*nodeId*). As credenciais são específicas para cada aplicação e contém as informações necessárias para autenticar o nó local. O argumento "Aplicação" é um identificador para a aplicação que fornece ao nó Pastry os procedimentos para invocar quando certos eventos acontecem, por exemplo a chegada de uma mensagem.

- `route(msg,key)`: envia uma mensagem ao nó cujo identificador *nodeId* seja aquele numericamente mais próximo à chave *key*, entre todos os nós Pastry ativos.

Por outro lado, as aplicações construídas sobre o ambiente Pastry devem exportar as seguintes operações:

- `deliver(msg,key)`: chamada pelo substrato Pastry quando for recebida uma mensagem destinada ao nó em questão;

- `forward(msg,key,nextId)`: chamada pelo substrato Pastry quando for recebida uma mensagem destinada a outro nó; essa mensagem será roteada pelo próprio Pastry para o nó com identificador *nextId*;

- `newLeafs(leafSet)`: chamada quando houver mudança na composição do *leafset* do nó local.

6.3 Implementação

O protótipo foi construído como uma aplicação executando sobre o *FreePastry* [Hoye, 2009] através de 1875 linhas de código Java. O *FreePastry* é uma implementação *open source* do *Pastry* escrita em Java que implementa o protocolo de roteamento e substrato *peer-to-peer*.

Foi implementado um conjunto de 18 classes em linguagem de programação Java responsável pelas funcionalidades do sistema de *backup* cooperativo tolerante a intrusões. Este conjunto de classes está dividido em cinco blocos funcionais como visto na figura 5.2 apresentada no capítulo 4. Os blocos são:

- A *Gerência de backup* (*Gerência*) é responsável por tratar as solicitações de *backup* e de recuperação de arquivos vindas do usuário local.

- O *Serviço de Quórum* responde pela replicação e distribuição dos dados.

- O *Serviço de Localização e Roteamento* baseado no *FreePastry* cuida da geração dos identificadores de nós e de arquivos, pela localização, pelo roteamento e também por definir o sistema de quóruns de cada nó.

- O *Serviço de Compressão* compacta e descompacta os arquivos a serem transferidos e armazenados.

- O *Serviço de Criptografia* cifra e decifra os dados além de fornecer assinatura criptográficas (*hash()*).

Este protótipo implementa as funções de armazenamento e recuperação de *backups*, apresentadas no capítulo anterior e que são as principais funções do sistema. As funções de remoção de arquivos, controle da validade de réplicas e manutenção do nível de replicação não foram implementadas pois não fazem parte do escopo deste projeto.

Os passos a seguir, representados na figura 6.1, são executados pelo protótipo agindo como cliente quando é realizada uma operação de armazenamento de um arquivo x no sistema de *backup*:

- A *Gerência* solicita ao *Serviço de Criptografia* a geração da chave para o arquivo x (passo 1).

- Em seguida, solicita ao *Pastry* a identificação o nó responsável pela chave deste arquivo (passos 2 e 3).

- Na seqüência, solicita a compactação e cifragem de x (resultando em x') (passos 4 e 5).

- A *Gerência* solicita o envio de x' a pelo menos $2f + 1$ nós (passo 6).

- Cada nó que recebeu o x' retorna como resposta o hash dos dados recebidos (*hash(y)*) (passo 7).

- A *Gerência* compara *hash(x')* com *hash(y)*, se forem iguais então a resposta é considerada correta. Caso contrário, ou se não houver resposta dentro do prazo um erro é registrado. Devem ser obtidas pelo menos $2f + 1$ respostas corretas para considerar-se encerrado o processo.

A seguir é mostrado na figura 6.2 uma operação de recuperação de um arquivo x do sistema de *backup* pelo pelo protótipo agindo como cliente:

- A *Gerência* solicita ao *Serviço de Criptografia* a geração da chave *key* para o arquivo x (passo 1).

- Em seguida, solicita ao *Pastry* a identificação do sistema de quóruns responsável por deste arquivo (passos 2 e 3).

- A seguir, o nó solicita o *hash* do arquivo x identificado pela chave *key* a um quórum de nós (passos 4 e 5) pertencente ao sistema informado no item anterior.

- Uma vez recebidas $2f + 1$ respostas, o nó solicita x a um dos nós que responderam corretamente, verifica se o *hash* do arquivo corresponde ao esperado (passos 6 e 7).

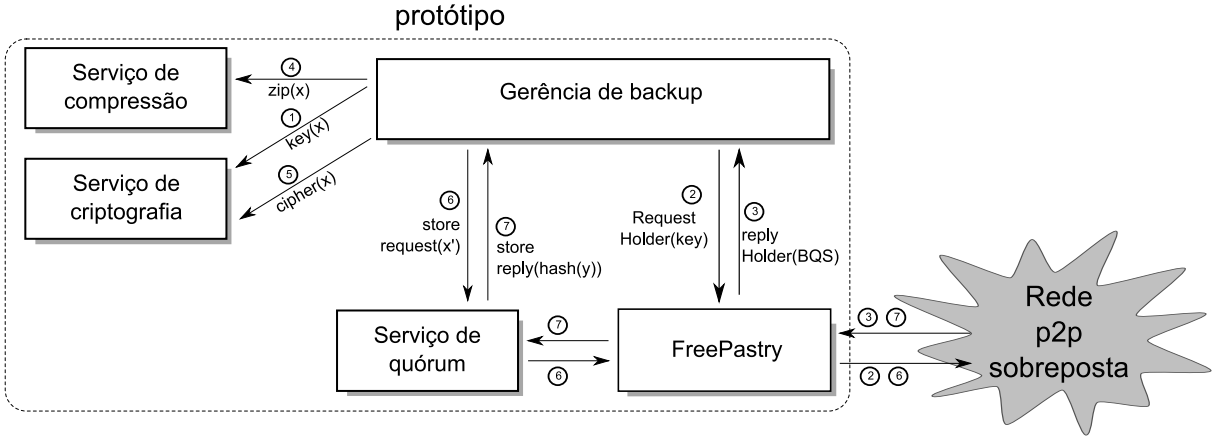

Figura 6.1: Seqüência de operações para o armazenamento de um arquivo no *BackupIT*.

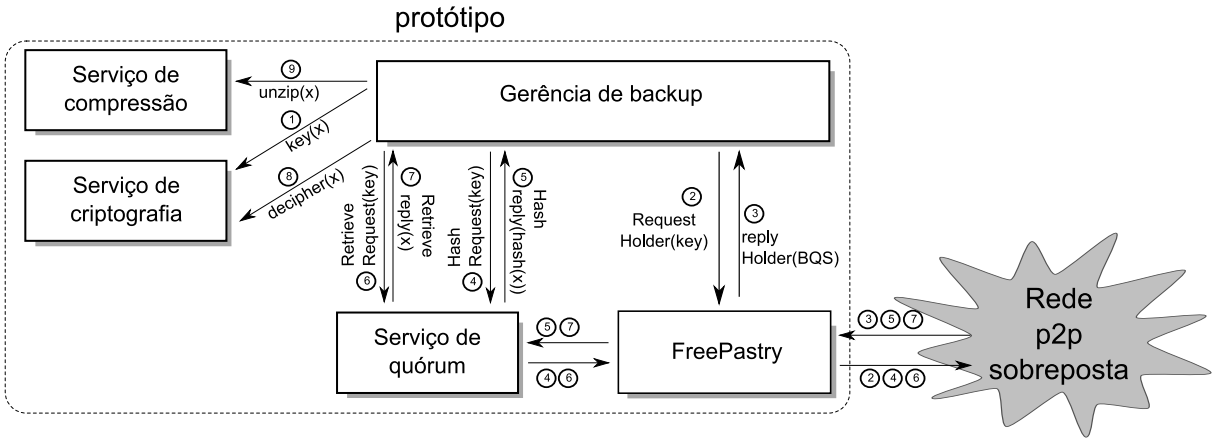

Figura 6.2: Seqüência de operações para a recuperação de um arquivo no *BackupIT*.

- Por último, solicita a descompactação e decifragem de x e devolve o arquivo ao usuário (passos 8 e 9).

A seguir é mostrado na figura 6.3 a execução do protótipo no atendimento das requisições de armazenamento e recuperação de um arquivo x do sistema de *backup*:

- A *gerência de backup* ao receber uma solicitação de identificação de nó responsável por uma determinada chave *key* e, sendo ele o responsável, responde informando todos os elementos dos seu sistema de quóruns ($\mathbb{S}_i$).

- Caso receba uma solicitação de armazenamento de um arquivo x, armazena este arquivo e responde com o *hash(x)*. A seguir, o nó solicita o *hash* do arquivo identificado pela chave *key* a um quórum de nós (passos 4 e 5).

- Uma vez recebidas $2f + 1$ respostas, o nó solicita x a um dos nós que responderam corretamente, verifica se o *hash* do arquivo corresponde ao esperado (passos 6 e 7).

- Por último, solicita a descompactação e decifragem de x e devolve o arquivo ao usuário (passos 8 e 9).

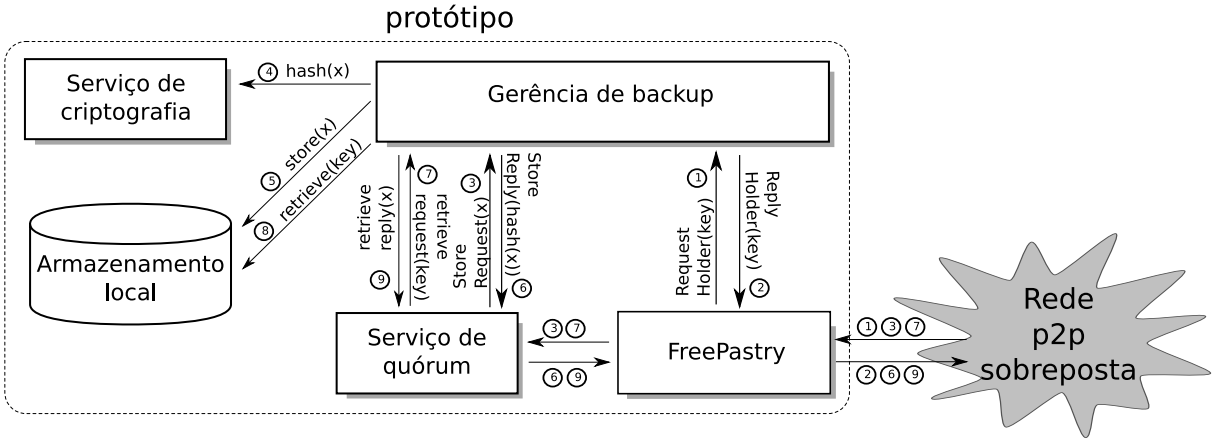

Figura 6.3: Seqüência de operações para o atendimento das requisições de armazenamento e recuperação de um arquivo no *BackupIT*.

6.4 Avaliação do Protótipo

A avaliação do protótipo do BackupIT visa verificar o seu comportamento em presença de faltas bizantinas. Espera-se que o sistema proposto seja capaz de tolerar até f faltas bizantinas sem perder a consistência ou a disponibilidade, ou seja, mantendo sua confiabilidade em 100%, além deste limite a confidencialidade do sistema pode sofrer degradação, conforme mostrado nas tabelas 5.1, 5.2, 5.3 e 5.4. Os algoritmos de armazenamento e recuperação de arquivos são avaliados, por sua vez, com relação à quantidade de mensagens trocadas no sistema, sendo que o seu comportamento esperado deverá estar em conformidade com a tabela 5.5.

6.4.1 Ambiente de Avaliação

Os algoritmos apresentados no capítulo 5 foram implementados como uma aplicação sobre o *FreePastry*, uma implementação aberta do ambiente *Pastry*. O ambiente de simulação foi um computador equipado com uma CPU Intel® Pentium® Dual E2140, 1.60GHz, 1GB de RAM, sistema operacional Linux Ubuntu 8.10 com *kernel* 2.6.19, máquina virtual SUN Java versão 1.6.0-07. Para a compressão dos arquivos foi adotado o algoritmo de compressão ZIP fornecido pela API Java. O algoritmo de criptografia simétrica adotado foi o *Triple DES*. Para os *hashes* criptográficos foi usado o algoritmo MD5. Esses algoritmos foram adotados por sua disponibilidade na plataforma Java, sem outro critério específico. Os nós integrantes da rede *peer-to-peer* são executados em uma única máquina e também compartilham a mesma máquina virtual java.

6.4.2 Metodologia

Três experimentos foram executados para avaliar o comportamento do sistema proposto. O primeiro avalia o número de mensagens trocadas no sistema durante os processos de armazenamento e recuperação de dados na presença de até f nós maliciosos. O segundo experimento avalia o número de mensagens trocadas no sistema durante os processos de armazenamento e recuperação de dados em relação ao tamanho do sistema sem a presença de

nós maliciosos. O último experimento avalia a confiabilidade do sistema com relação ao crescimento do número de nós maliciosos ultrapassando o limite de nós faltosos no sistema para valores diferentes de f. Para esses testes foi executado o mesmo procedimento de preparação e finalização do ambiente, para que os resultados fossem o confiáveis. O sistema de *backup* cooperativo é iniciado, e os nós são iniciados e conectados através de uma rede *peer-to-peer overlay* estabelecida pelo Pastry. Cada operação de armazenamento ou recuperação de arquivo foi executada 100 vezes, tendo sido observado um coeficiente de variação inferior a 5% nos resultados. O tamanho dos arquivos utilizados foi de 1 KB, sendo que cada arquivo foi gerado com um conteúdo aleatório para garantir que cada arquivo era diferente do outro. Não foram experimentados outros tamanhos de arquivo, pois em um ambiente simulado isto somente informaria a capacidade de processamento e de transferência de dados com o disco rígido do computador que executou os testes. Mesmo em um ambiente real, a variação do tamanho dos arquivos não mostraria a capacidade do sistema de tolerar faltas bizantinas. Após a execução dos testes, e a coleta das medições, todo o sistema é desligado. Isto é, os nós são eliminados, a máquina java virtual é removida.

6.4.3 Resultados e Avaliação

Para avaliar o sistema de *backup* cooperativo implementado, foram feitas três avaliações. Para a primeira avaliação, que mostra a quantidade de mensagens trocadas pelo sistema nos processos de armazenamento e de recuperação de dados sem a presença de nós maliciosos, a quantidade de nós do sistema varia de um mínimo de 4 até 22 e o valor de f variou de 1 até 7 conforme a relação $(3f + 1)$ conforme mostrado na figura 6.4 mostrada a seguir. Os dados obtidos confirmam o valores previstos na tabela 5.5.

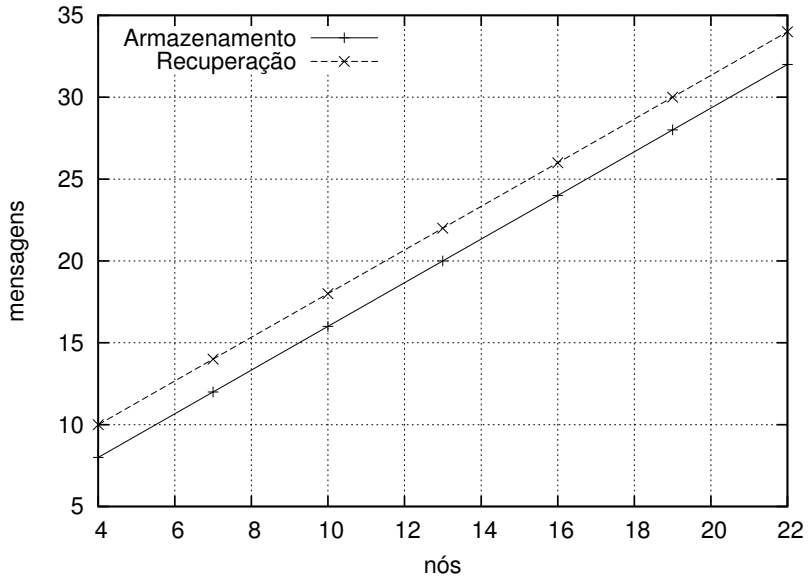

Figura 6.4: Número de mensagens trocadas no sistema durante os processos de armazenamento e recuperação de dados em relação ao tamanho do sistema sem a presença de nós maliciosos no sistema *BackupIT*.

Para a segunda avaliação, que analisa o número de mensagens trocadas pelo sistema nos processos de armazenamento e de recuperação de dados quando exposto a nós com com-

portamento bizantino até um limite de $f = 7$, escolheu-se um sistema com 22 nós e variou-se o número dos maliciosos entre 0 (zero) e 7 nós (pois $3f + 1 = 22$). Mediu-se o número de mensagens trocadas entre os nós para o armazenamento e para a recuperação de um arquivo. Os dados apresentados na figura 6.5 mostram que o número de mensagens varia de forma linear com o número de nós maliciosos, conforme previsto na tabela 5.5.

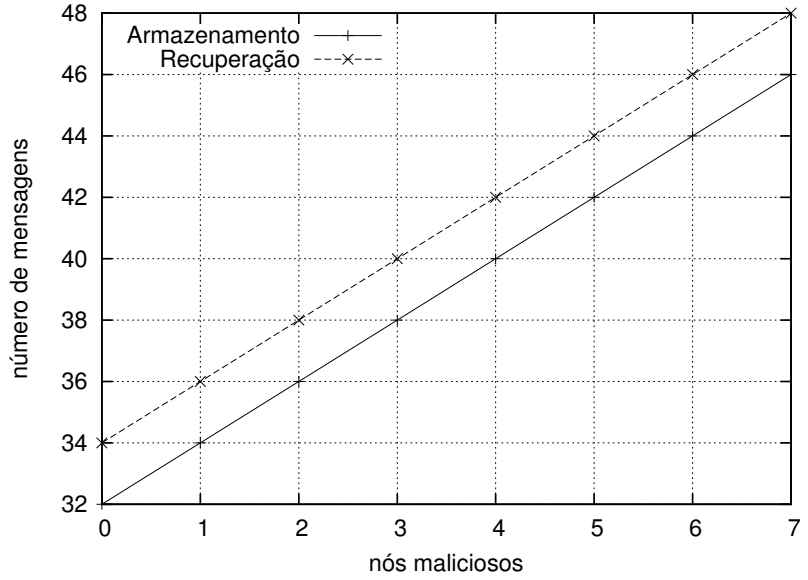

Figura 6.5: Número de mensagens trocadas no sistema durante os processos de armazenamento e recuperação de dados na presença de até f nós maliciosos no sistema *BackupIT*.

No terceiro experimento, que avalia a confiabilidade do sistema com relação ao crescimento do número de nós maliciosos presentes no sistema, a quantidade de nós foi fixada em 22 e o número de nós faltosos variou de 0 até 21 com o limite de faltas variando de 1 até 7 para cada valor diferente do número de nós faltosos no sistema. A figura 6.6, a seguir, apresenta os resultados desta avaliação. Nesta figura (6.6) verifica-se que o sistema suporta a quantidade de nós maliciosos conforme projetado, mantendo sua confiabilidade mesmo quando o número de nós maliciosos ultrapassa o limite de faltas definido. Assim, para um sistema com 22 nós, até 7 nós podem falhar de forma bizantina sem que haja perda de dados, e mesmo que mais nós falhem a confiabilidade do sistema ainda se mantém.

6.4.4 Comparação com projetos correlatos

O sistema *pStore* ([Batten et al., 2002]) apresenta uma disponibilidade de 95% para uma situação em que 23% de seus nós deixaram a rede e restando apenas quatro réplicas para de seus arquivos para recuperação. Comparando este resultado com o apresentado na figura 6.6, é possível verificar que o sistema de *backup* distribuído *BackupIT* apresenta 100% de confiabilidade com 76% de seus nós presentes para qualquer limite de faltas e começando a apresentar uma queda na confiabilidade a partir de 52% de nós maliciosos presentes no sistema e isto somente para o valor do limite de faltas f igual a um. Somente após 66% de nós maliciosos no sistema é que observou-se a redução da confiabilidade para o limite de faltas igual a dois.

O sistema *ABS* ([Cooley et al., 2004]), conseguiu uma disponibilidade de 90% mesmo em presença de 75% de nós faltosos (seis em oito nós) o que foi igualado pelo *BackupIT* nas

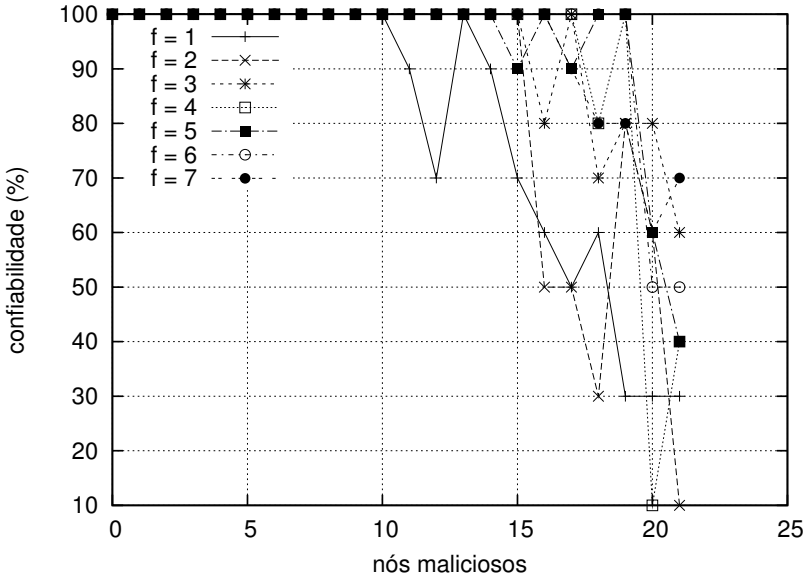

Figura 6.6: Variação da confiabilidade do sistema com relação ao crescimento do número de nós maliciosos no sistema *BackupIT*.

situações em que o limite de faltas é maior que um (sete réplicas ou mais distribuídas no sistema).

O sistema *DIBS* [Hsu et al., 2004] alcançou uma disponibilidade de 100% para todas as suas réplicas desde que o tempo para que em um período de 3 segundos as réplicas indisponíveis sejam substituídas. O *BackupIT* não precisa de tempo para remanejamento de suas réplicas para garantir uma alta disponibilidade e conseqüentemente, sua confiabilidade.

6.5 Conclusão do Capítulo

Neste capítulo foi descrita a implementação do protótipo e o seu funcionamento foi explicado. Foram mostrados os resultados dos experimentos realizados para comprovar o funcionamento do sistema e a sua tolerância a intrusões. Foi analisado, também, o número de mensagens gasto no armazenamento e recuperação de arquivos com e sem a presença de nós maliciosos.

Os resultados apresentados comprovam que o sistema funciona conforme a sua especificação, mantendo sua confiabilidade em 100% mesmo nos casos em que o número de nós faltosos ultrapassa o limite de faltas f especificado. Para garantir este nível de confiabilidade são necessários $2f + 1$ réplicas distribuídas pelo sistema e com um custo de aproximadamente duas mensagens por nó trocadas no sistema durante os processos de armazenamento ou recuperação de um arquivo.

Capítulo 7

Conclusão

Esta dissertação descreve o projeto e implementação do BackupIT, um sistema de *backup* cooperativo. O BackupIT é o primeiro sistema de *backup* cooperativo que fornece tolerância a intrusões através do uso de algoritmos para sistemas de quóruns bizantinos. Desta forma o serviço de *backup*, fornecido pelo BackupIT, garante disponibilidade e consistência para os dados armazenados, além de eficiência no uso do espaço de armazenamento. Este capítulo inicia contextualizando o trabalho, em seguida expondo o problema o qual este projeto se propos resolver. Dando seqüência, é descrito o desenvolvimento do projeto e os resultados obtidos. Por último são discutidas as limitações do trabalho e as possibilidades para trabalhos futuros.

7.1 O Problema

As tecnologias tradicionais de *backup* e recuperação são soluções baseadas em gravações em fitas magnéticas e visam primeiramente cenários de desastres para centros de dados com tempos de recuperação medidos em dias. No entanto, a continuidade dos negócios e o atendimento a requisitos além dos tradicionais para recuperação de desastres estão levando as organizações a demandar soluções que possam reduzir o potencial de tempo de interrupção dos seus serviços e perdas de dados para horas, minutos ou até segundos. Dependendo da quantidade de dados críticos que devem ser preservados nas organizações, o espaço necessário para armazenar os *backups* pode chegar facilmente a centenas de terabytes. Ao mesmo tempo, uma corporação pode ter uma grande capacidade de armazenamento ociosa, pois em média 50% dos discos rígidos de cada computador de sua rede interna estão livres. Além disso, existe também uma grande capacidade de processamento ociosa e disponível nessa mesma rede, pois, a atividade mais comum de um computador pessoal é esperar por entradas do usuário.

7.2 O BackupIT

Neste trabalho foi apresentado o BackupIT, um sistema de *backup* cooperativo baseado em sistemas de quóruns bizantinos. Este sistema oferece tolerância a intrusões, uso eficiente dos recursos da rede, além de consistência e disponibilidade, características básicas dos sistemas de quóruns. Também foram utilizadas técnicas criptográficas para fornecer confidencialidade e integridade ao sistema.

Os resultados obtidos das avaliações experimentais mostram que o sistema é capaz de tolerar intrusões até o limite de faltas determinado sem apresentar perda dos dados, além de manter sua confiabilidade muito elevada mesmo quando a quantidade de nós maliciosos é maior que o limite de faltas. Nas avaliações de tolerância a intrusões um sistema com 22 nós foi simulado com as ferramentas fornecidas pelo ambiente do FreePastry. Nós maliciosos foram introduzidos no sistema até o limite de faltas do sistema e além deste.

Também foi avaliado o número de mensagens trocadas na rede para executar os processos de armazenamento e recuperação de *backups*. Nestas avaliações o tamanho do sistema foi incrementado até um limite de 22 nós. Assim como a avaliação da tolerância a intrusões, nesta avaliação o sistema também foi simulado. Os resultados mostram que o número de mensagens trocadas na rede é diretamente proporcional ao tamanho do sistema de quóruns. Este resultado se deve principalmente aos algoritmos de escrita e leitura para sistemas de quóruns bizantinos que apresentam esta relação do número de mensagens proporcional ao tamanho do sistema. Em presença de nós maliciosos no sistema, o BackupIT apresentou um aumento no número de mensagens trocadas de duas vezes o número de nós faltosos no sistema. Assim como nas experimentações anteriores, este aumento se deve justamente ao aumento da complexidade dos algoritmos para poder fornecer disponibilidade e consistência ao sistema. O sistema *BackupIT* foi comparado com os sistemas *pStore*, *ABS* e *DIBS*. Nestas comparações o *BackupIT* mostrou ser vantajoso com relação a sua confiabilidade e uso de espaço de armazenamento.

7.3 Limitações e Trabalhos Futuros

Um sistema de *backup* cooperativo deve fornecer serviços para garantir: a confidencialidade e privacidade aos seus usuários; integridade, consistência e disponibilidade de dados; sinergia dos recursos e gerência de confiança aos pares. Pelo fato deste sistema ter sido planejado para ser utilizado em ambiente de Intranet a gerência de confiança torna-se desnecessária pois todos os pares estão sob um mesmo domínio administrativo tornando os comportamentos egoistas menos prováveis de ocorrer. Além do mais, a sinergia de recursos é muito maior neste caso, pois como todos os pares estão sob um mesmo domínio administrativo implica em que todos tem interesses em comum e devem colaborar entre si com maior boa vontade. Para implementar um sistema de *backup* que atenda a todos os requisitos para poder ser utilizado na Internet faz se necessário modelar todas as operações e situações possíveis para o sistema, o que não é o escopo deste trabalho. Assim, o sistema proposto tem seu foco em tolerância a faltas através do uso de sistemas de quóruns bizantinos.

Além disto, o sistema proposto possui algumas limitações que poderão ser suprimidas no futuro. Uma delas é que a detecção de faltas ocorre somente quando o cliente precisa recuperar os seus *backups* do sistema. Se neste momento não houverem mais réplicas em quantidade suficiente para garantir a consistência dos dados o cliente corre o risco de ter perdido seus dados. Para solucionar esta limitação faz-se necessário implementar uma gerência de manutenção de réplicas de forma a garantir a disponibilidade e consistência a qualquer momento e não somente no momento da recuperação dos dados.

O sistema proposto pode ser aperfeiçoado em diversos aspectos. Em primeiro lugar, o registro adicional dos *hashes* dos arquivos armazenados no lado do cliente (o nó que solicitou o armazenamento) permite diminuir o tamanho dos sistemas de quóruns utilizados, uma vez que o cliente pode usar essa informação para verificar a integridade dos valores recebidos. Como o sistema de *backup* cooperativo usa uma semântica de acesso *single-writer/single-reader*

regular, seria possível trabalhar com quóruns de tamanho $f + 1$, diminuindo o tráfego de rede e o consumo de área de armazenamento. Outra possibilidade de trabalho futuro seria o estudo de um protocolo de quórum que também considere clientes maliciosos como, por exemplo, o sistema com semântica *single-writer/multi-reader* segura usando o sistema de quóruns de f-mascaramento apresentado em [Dantas et al., 2007].

Referências Bibliográficas

[Aiyer et al., 2005] Aiyer, A., Alvisi, L., Clement, A., Dahlin, M., Martin, J., and Porth, C. (2005). BAR fault tolerance for cooperative services. *SIGOPS Operating Systems Review*, 39(5):45 – 58.

[Alima et al., 2003] Alima, L., El-Ansary, S., Brand, P., and Haridi, S. (2003). Dks(n, k, f) a family of low-communication, scalable and fault-tolerant infrastructures for p2p applications. In *3rd International workshop on Global and P2P Computing on Large Scale Distributed Systems*, Tokyo, Japan.

[Anderson et al., 1998] Anderson, R., Needham, R., and Shamir, A. (1998). The steganographic file system. In *Second International Workshop on Information Hiding*, pages 73–82, London, UK. Springer-Verlag.

[Androutsellis-Theotokis and Spinellis, 2004] Androutsellis-Theotokis, S. and Spinellis, D. (2004). A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys*, 36(4):335 – 371.

[AOL, 2009] AOL (2009). Aol instant messenger. `http://www.aim.com`. Acesso em: 1 de jun. 2009.

[Apache, 2008] Apache (2008). Apache tapestry. `http://tapestry.apache.org`. Acesso em: 1 de jun. 2009.

[Batten et al., 2002] Batten, C., Barr, K., Saraf, A., and Treptin, S. (2002). pstore: A secure peer-to-peer backup system. Technical Report TM – 632, MIT Laboratory for Computer Science.

[Bolosky et al., 2000a] Bolosky, W. J., Corbin, S., Goebel, D., and Douceur, J. R. (2000a). Single instance storage in windows 2000. In *4th Usenix Windows System Symposium*, Seattle, WA, USA.

[Bolosky et al., 2000b] Bolosky, W. J., Douceur, J. R., Ely, D., and Theimer, M. (2000b). Feasibility of a serverless distributed file system deployed on an existing set of desktop pcs. In *International Conference on Measurement and Modeling of Computer Systems*, pages 34 – 43, Santa Clara, CA.

[Castro et al., 2002] Castro, M., Druschel, P., Ganesh, A. A. R., and Wallach, D. (2002). Secure routing for structured peer-to-peer overlay networks. *ACM SIGOPS Operating Systems Review*, 36(SI):299–314.

[Castro and Liskov, 2002] Castro, M. and Liskov, B. (2002). Practical byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems*, 20(4):398–461.

[Chen et al., 2000] Chen, Y., Katz, R., and Kubiatowicz, J. (2000). Scan: A dynamic, scalable and efficient content distribution network. In *First International Conference on Pervasive Computing*, pages 282 – 296, London, UK. Springer-Verlag.

[Chervenak et al., 1998] Chervenak, A. L., Vellanki, V., and Kurmas, Z. (1998). Protecting file systems: A survey of backup techniques. In *Joint NASA and IEEE Mass Storage Conference*, Maryland, USA.

[Cipar et al., 2007] Cipar, J., Corner, M., and Berger, E. (2007). Tfs: A transparent file system for contributory storage. In *USENIX Conference on File and Storage Technologies*.

[Clark et al., 2000] Clark, I., Sandberg, O., Wiley, B., and Hong, T. (2000). Freenet: A distributed anonymous information storage and retrieval system. In *Workshop on Design Issues in Anonymity and Unobservability*, Berkeley, CA.

[Cooley et al., 2004] Cooley, J., Taylor, C., and Peacock, A. (2004). Abs: the apportioned backup system. Technical report, Massachusetts Institute of Technology, Cambridge, Massachuesetts.

[Correia, 2005] Correia, M. (2005). Servicos distribuídos tolerantes a intrusões: resultados recentes e problemas abertos. Technical Report TR-05-18, Faculdade de Ciências da Universidade de Lisboa, Campo Grande, 1749 – 016 Lisboa Portugal.

[Cox et al., 2002] Cox, L. P., Murray, C. D., and Noble, B. D. (2002). Pastiche: making backup cheap and easy. *SIGOPS Operating Systems Review*, 36:285 – 298.

[Dantas et al., 2007] Dantas, W., Bessani, A., Fraga, J., and Correia, M. (2007). Evaluating byzantine quorum systems. In *IEEE Symposium on Reliable Distributed Systems*, pages 253 – 264, Beijing, China.

[DCT, 2009] DCT (2009). distributed.net. `http://www.distributed.net`. Acesso em: 1 de jun. 2009.

[Dingledine et al., 2000] Dingledine, R., Freedman, M., and Molnar, D. (2000). The freehaven project: Distributed anonymous storage service. In *Workshop on Design Issues in Anonymity and Unobservability*, pages 67 – 95, Berkeley, CA, USA.

[Douceur and Bolosky, 1999] Douceur, J. R. and Bolosky, W. J. (1999). A large-scale study of file-system contents. In *ACM Conference on Measurement and Modeling of Computer Systems*, pages 59 – 70, Atlanta, Georgia, USA.

[Edutella, 2004] Edutella (2004). Edutella - p2p for the semantic web. `http://www.edutella.org/edutella.shtml`. Acesso em: 1 de jun. 2009.

[Fischer et al., 1985] Fischer, M. J., Lynch, N. A., and Paterson, M. S. (1985). Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374 – 382.

[Fraga and Powell, 1985] Fraga, J. S. and Powell, D. (1985). A fault- and intrusion-tolerant file system. *3rd International Conference on Computer Security*, pages 203–218.

[Gifford, 1979] Gifford, D. K. (1979). Weighted voting for replicated data. *7th ACM Symposium on Operating Systems Principles*, pages 150 – 162.

[Groove, 2007] Groove (2007). Microsoft office groove. `http://office.microsoft.com/en-us/groove/FX100487641033.aspx`. Acesso em: 1 de jun. 2009.

[Growchowski, 1988] Growchowski, E. (1988). Emerging trends in data storage on magnetic hard disk drives. *Datatech*, 32(2):11 – 16.

[Haeberlen et al., 2005] Haeberlen, A. H., J. Mislove, A., and Druschel, P. (2005). Consistent key mapping in structured overlays. Technical Report TR05 – 456, Rice CS Department, Houston TX.

[Hand and Roscoe, 2002] Hand, S. and Roscoe, T. (2002). Mnemosyne: Peer-to-peer steganographic storage. In *1st International Workshop on Peer-to-Peer Systems (IPTPS'02)*, Cambridge, MA.

[Hoye, 2009] Hoye, J. (2009). Freepastry. `http://freepastry.rice.edu`. Acesso em: 1 de jun. 2009.

[Hsu et al., 2004] Hsu, E., Mellen, J., and Naresh, P. (2004). DIBS: distributed backup for local area networks. Technical report, Parallel & Distributed Operating Systems Group, MIT.

[Junqueira et al., 2003] Junqueira, F., Bhagwan, R., Marzullo, K., Savage, S., and Voelker, G. M. (2003). The phoenix recovery system: rebuilding from the ashes of an internet catastrophe. In *Ninth Workshop on Hot Topics in Operating Systems*, Lihue, Hawaii.

[Kamienski et al., 2005] Kamienski, C., Souto, E., Rocha, J., Domingues, M., Callado, A., and Sadok, D. (2005). Colaboracao na internet e a tecnologia peer-to-peer. In *XXV Congresso da Sociedade Brasileira de Computacao*, Sao Leopoldo, RS.

[Killijian and Courtes, 2006] Killijian, M. and Courtes, L. (2006). A survey of cooperative backup mechanisms. Technical Report 06472, LAAS, Toulouse France.

[Kirk, 2003] Kirk, P. (2003). The gnutella protocol specification 0.6. `http://rfc-gnutella.sourceforge.net`. Acesso em: 1 de jun. 2009.

[Kubiatowicz et al., 2000] Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Wells, C., and Zhao, B. (2000). Oceanstore: an architecture for global-scale persistent storage. In *ASPLOS-IX: Proceedings of the ninth International conference on Architectural Support for Programming Languages and Operating Systems*, pages 190 – 201, New York, NY, USA.

[Lamport et al., 1982] Lamport, L., Shostak, R., and Pease, M. (1982). The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401.

[Landers et al., 2004] Landers, M., Zhang, H., and Tan, K.-L. (2004). Peerstore: better performance by relaxing in peer-to-peer backup. *IEEE International Conference on Peer-to-Peer Computing*, pages 72 – 79.

[Leibowitz et al., 2003] Leibowitz, N., Ripeanu, M., and Wierzbicki, A. (2003). Deconstructing the kazaa network. In *WIAPP '03: Proceedings of the The Third IEEE Workshop on Internet Applications*, page 112, Washington, DC, USA. IEEE Computer Society.

[Li and Dabek, 2006] Li, J. and Dabek, F. (2006). F2F: Reliable storage in open networks. In *Workshop on Peer-to-Peer Systems*, Santa Barbara, CA, USA.

[Lillibridge et al., 2003] Lillibridge, M., Elnikety, S., Birrell, A., Burrows, M., and Isard, M. (2003). A cooperative internet backup scheme. In *USENIX Annual Technical Conference*, San Antonio, Texas, USA.

[Liskov and Rodrigues, 2006] Liskov, B. and Rodrigues, R. (2006). Tolerating byzantine faulty clients in a quorum system. In *IEEE Conference on Distributed Computing Systems*, Lisbon, Portugal.

[Lua et al., 2005] Lua, E., Crowcroft, J., Pias, M., Sharma, R., and Lim, S. (2005). A survey and comparison of peer-to-peer overlay network schemes. *Communications Surveys & Tutorials, IEEE*, pages 72 – 93.

[Mailath, 1998] Mailath, G. J. (1998). Do people play nash equilibrium? lessons from evolutionary game theory. *Journal of Economic Literature*, 36(3):1347 – 1374.

[Malkhi and Reiter, 1997] Malkhi, D. and Reiter, M. (1997). Byzantine quorum systems. In *ACM Symposium on Theory of Computing*, pages 569 – 578, El Paso, Texas, USA.

[Malkhiy et al., 2000] Malkhiy, D., Piercez, E., Reiter, M. K., Wright, R. N., and Alvisi, L. (2000). Dynamic byzantine quorum systems. In *2000 International Conference on Dependable Systems and Networks (formerly FTCS-30 and DCCA-8)*, Washington, DC, USA. IEEE Computer Society.

[Manber, 1994] Manber, U. (1994). Finding similar files in a large file system. In *USENIX Winter 1994 Conference*, pages 1 – 10, San Francisco, CA.

[Maymounkov and Mazieres, 2002] Maymounkov, P. and Mazieres, D. (2002). Kademlia: A peer-to-peer information system based on the xor metric. In *1st International Workshop on Peer-to-peer Systems*, Cambridge, MA, USA.

[McCoy, 2000] McCoy, J. (2000). Mojo nation. `http://sourceforge.net/projects/mojonation`. Acesso em: 1 de jun. 2009.

[Mondejar et al., 2005] Mondejar, R., Garcia, P., and Pairot, C. (2005). Bunshin: Dht para aplicaciones distribuidas. In *I Congresso Espanhol de Informatica*, Granada, Spain.

[MSN, 2009] MSN (2009). Microsoft msn messenger. `http://messenger.msn.com`. Acesso em: 1 de jun. 2009.

[Oliveira et al., 2008] Oliveira, M., Cirne, W., Brasileiro, F., and Guerrero, D. (2008). On the impact of the data redundancy strategy on the recoverability of friend-to-friend backup systems. In *26th Brazilian Symposium on Computer Networks and Distributed Systems*, Rio de Janeiro, RJ.

[Plank, 1997] Plank, J. S. (1997). A tutorial on reed-solomon coding for fault-tolerance in RAID-like systems. *Software, Practice and Experience*, 27(9):995 – 1012.

[Pouwelse et al., 2004] Pouwelse, J., Garbacki, P., Epema, D., and Sips, H. (2004). A measurement study of the bittorrent peer-to-peer file-sharing system. In *19th IEEE Annual Computer Communications Workshop*, Bonita Springs, Florida.

[Rabin, 1989] Rabin, M. O. (1989). Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the ACM*, 36(2):335–348.

[Ratnasamy et al., 2001] Ratnasamy, S., Francis, P., Handley, M., Karp, R., and Schenker, S. (2001). A scalable content-addressable network. In *Conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161 – 172, New York, NY, USA.

[Rhea et al., 2005] Rhea, S., Godfrey, B., Karp, B., Kubiatowicz, J., Ratnasamy, S., Shenker, S., Stoica, I., and Yu, H. (2005). Opendht: A public dht service and its uses. In *ACM Special Interest Group on Data Communication*, Philadelphia, PA, USA.

[Rodrigues and Liskov, 2005] Rodrigues, R. and Liskov, B. (2005). High availability in dhts: Erasure coding vs. replication. In *Workshop on Peer-to-Peer Systems*, Ithaca, NY, USA.

[Rowstron and Druschel, 2001a] Rowstron, A. and Druschel, P. (2001a). Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM Conference on Distributed Systems Platforms*, pages 329 – 350, Heidelberg, Germany.

[Rowstron and Druschel, 2001b] Rowstron, A. and Druschel, P. (2001b). Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *18th ACM Symposium on Operating Systems Principles*, Lake Louise, Alberta, Canada.

[Sandvine, 2008] Sandvine (2008). Online entertainment applications dominate peak evening hours. `http://www.sandvine.com/news/pr_detail.asp?ID=203`. Acesso em: 1 de jun. 2009.

[Saroiu et al., 2003] Saroiu, S., Gummadi, K. P., and Gribble, S. D. (2003). Measuring and analyzing the characteristics of napster and gnutella hosts. *Multimedia Syst.*, 9(2):170–184.

[Shamir, 1979] Shamir, A. (1979). How to share a secret. *Communications of the ACM*, 22(11):612–613.

[Sit et al., 2003] Sit, E., Cates, J., and Cox, R. (2003). A dht – based backup system. In *1st IRIS Student Workshop*, Cambridge, Massachusetts, USA.

[Stoica et al., 2001] Stoica, I., Morris, R., Liben-Nowell, D., Karger, D., Kaashoek, M., Dabek, F., and Balakrishnan, H. (2001). Chord: A scalable peer-to-peer lookup protocol for internet applications. In *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, San Diego, CA, USA.

[Sung et al., 2006] Sung, L. G. A., Ahmed, N., Blanco, R., Li, H., Soliman, M. A., and Hadaller, D. (2006). A survey of data management in peer-to-peer systems. Technical report, David R. Cheriton School of Computer Science, University of Waterloo, Canada.

[TorrentFreak, 2008] TorrentFreak (2008). Shocking 61% of all upstream internet traffic is p2p. http://torrentfreak.com/shocking-61-of-all-upstream-internet-traffic-is-p2p-081021. Acesso em: 1 de jun. 2009.

[UCA, 2009] UCA (2009). Seti@home. http://setiathome.ssl.berkeley.edu/. Acesso em: 1 de jun. 2009.

[Veríssimo et al., 2003] Veríssimo, P. E., Neves, N. F., and Correia, M. P. (2003). Intrusion-tolerant architectures: Concepts and design. In *Architecting Dependable Systems, volume 2677 of Lecture Notes in Computer Science*, pages 3–36. Springer-Verlag.

[Waldman et al., 2000] Waldman, M., Rubin, A., and Cranor, L. (2000). Publius: A robust, tamper-evident, censorship-resistant web publishing system. In *9th USENIX Security Symposium*, Denver, Colorado, USA.

[WindowsMeetingSpace, 2009] WindowsMeetingSpace (2009). Windows meeting space. http://www.microsoft.com/windows/windows-vista/features/meeting-space.aspx. Acesso em: 1 de jun. 2009.

[Yahoo, 2009] Yahoo (2009). Yahoo! messenger. http://messenger.yahoo.com. Acesso em: 1 de jun. 2009.