TIAGO VIGNATTI

ARQUIVAMENTO DIGITAL A LONGO PRAZO BASEADO EM SELEÇÃO DE REPOSITÓRIOS EM REDES PEER-TO-PEER

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.

Orientador: Prof. Dr. Luis Carlos Erpen De Bona

CURITIBA

2009

Resumo

A importância da informação digital é cada vez maior na sociedade atual. Bibliotecas digitais, aplicações na Internet tais como *email*, compartilhamento de fotos e arquivos de página Web frequentemente precisam ser preservados por um longo período de tempo e cabem aos sistemas de arquivamento digital tal responsabilidade. Esta dissertação propõe um modelo de replicação confiável de conteúdo digital para ser utilizado em sistemas de arquivamento a longo prazo. O sistema de arquivamento é modelado como um conjunto de repositórios de armazenamento aos quais são atribuídas probabilidades independentes de falha a cada um deles. Itens são inseridos neste sistema com uma confiabilidade que é satisfeita replicando-os em subconjuntos de repositórios. Este modelo implica em não considerar um grau de replicação fixa para cada item armazenado no sistema, permitindo a criação de estratégias que otimizem a utilização dos recursos dos repositórios. Através de simulação, avaliamos três diferentes estratégias para a criação das réplicas. Também é proposto um sistema de arquivamento totalmente distribuído utilizando este modelo sobre uma rede *peer-to-peer* (P2P). A comunicação dos nodos (repositórios) desta rede é organizada em uma tabela *hash* distribuída e múltiplas funções *hashs* são utilizadas para selecionar os repositórios que irão guardar as réplicas de cada item armazenado. O sistema, junto com seus algoritmos para a criação das réplicas, foi avaliado através de experimentos em um ambiente real. Acreditamos que o modelo e os algoritmos propostos, combinados à escalabilidade das redes P2P estruturadas, viabilizam a construção de sistemas de arquivamento digital confiáveis e totalmente distribuídos.

Abstract

The importance of digital information is increasing constantly in today's society. Digital libraries, Internet applications such email, photo sharing and Web site files often need to be preserved for a long-term and this is a responsability of digital archiving systems. This dissertation proposes a reliable replication model of digital content to be used in long-term archiving systems. The archiving system is modeled as a set of storage repositories in which independent fail probabilities are assigned for each one of them. Items are inserted in this system with a reliability that is satisfied replicating them in subsets of repositories. This model implies in not considerer a fixed degree of replication for each item stored in the system, allowing to create strategies to optimize the utilization of repositories resources. Through simulation, we evaluated three different strategies to create replicas. It is also proposed a totally distributed archiving system using this model over a peer-to-peer (P2P) network. The communication between the nodes (repositories) of such network is organized in a distributed hash table and multiple hash functions are used to select repositories that will keep the replicas of each stored item. The system and its algorithms to create replicas are evaluated through experiments in a real environment. We believe that this model and the proposed algorithms, combined with the structured P2P scalability, make possible to construct a reliable and totally distributed digital archiving systems.

à minha família
aos meus melhores amigos

Agradecimentos

Agradeço primeiramente a minha maravilhosa família por ter sempre incentivado a minha educação, inclusive me presenteando com um computador *TK 2000* ainda quando criança (em meados de 1993). Sem este computador eu não estaria aqui. Assim, agradeço a minha mãe amada Elusa, que além de ter me motivado na informática com o *TK 2000*, sempre ajudou e cobrou a minha educação (por exemplo, "tomando" rigorosamente as lições de matemática já no jardim de infância) e que também me aguentou nos momentos de ansiedade e nervosismo durante o período da escrita desta dissertação. Agradeço o meu pai Olir, que sempre deu os melhores conselhos e pela nossa amizade indescritível (a propósito, no começo ele queria que eu fosse jogador de futebol – como todo pai brasileiro – mas agora ele está conformado e feliz com a minha carreira de *nerds* da computação). Agradeço também o André, meu irmão (também computeiro), meu ídolo e maior companheiro que sempre estará desfrutando as belezas da vida do meu lado. Um agradecimento muito especial a minha falecida avó Terezinha que participou da minha educação como ninguém.

Agradeço ao capitalismo selvagem que me motivou escolher computação ao invés da faculdade de música, pensando que a carreira na informática iria me tornar um homem rico (ahh, quanta ingenuidade!). Da mesma maneira, meus votos de agradecimento vão a toda sociedade chata e medíocre, onde alega que música e arte não trazem o ganha pão diário e são vistas com maus olhos. Estou aqui por causa de vocês. Obrigado!

Além dos fatos que fizeram eu chegar neste importante marco na minha vida, também estavam presentes algumas pessoas que cooperaram para alcancá-lo. Assim, agradeço o meu amigo e orientador Prof. Luis de Bona, que agiu como um "paizão": deu broncas,

incentivou, apoiou e parabenizou sempre na medida necessária. A todos os professores do C3SL, em especial os "caciques" que estiveram mais próximos: o próprio Bona, o Castilho e o Fabiano. A vocês três eu devo muito do meu conhecimento que tenho hoje. Ao Prof. Alexandre Direne, pelas sábias e oportunas palavras (e que deve ser o professor do departamento que mais está presente nos agradecimentos de dissertações – obviamente, por ótimas razões!). Não posso deixar de agradecer outros professores que me tocaram de alguma forma em todo o tempo dentro da universidade: Jair (me instruindo com o "kit básico de sobrevivência" da teoria da computação), Roberto (com seu jeito "Dr. House" de ensinar e ser) e Sunye (exemplo de um sujeito carismático). Os bons amigos que fiz dentro do C3SL e da universidade, em especial agradeço aqueles com quem eu passei mais tempo: no começo o Marcinho Miranda, depois o Bruno Russo Ribas, o Ander Conselvan, o Antonio H. Neto e o Juliano Picussa. Também o Henrique Brieta Barbieri, que esteve presente nos 6 anos e meio dentro da universidade junto comigo (e por ter revisado esta dissertação. Valeu!).

Agradeço muito a meus grandes amigos de longa data Gustavo, Su, Eduardo, Rafael, Wagner, Riche, Luizinho, Batata, Tim, Leon e Paulinho; agradeço o Igor também que infelizmente está ausente em São Paulo. Também ao Rodrigo, parceiro demais na minha adolescência. Um agradecimento especial à Gi que sempre me apoiou, incentivou, deu dicas, esteve ao meu lado e proporcinou os melhores momentos. Pessoal, vocês todos são os melhores e não conseguiria nada sem as suas presenças!

Por fim, gostaria de agradecer alguns bons passatempos (ou seriam procrastinações?) que fizeram com que as horas chatas da dissertação se tornassem mais harmoniosas. Primeiramente a bela cidade de Curitiba por proporcionar os belos butecos e bares de badalação. Agradeço também o Pink Floyd, o Frank Zappa, Hermeto Pascoal, Arrigo Barnabé e tantos outros bons músicos que fizeram maravilhosas sinfonias para os meus ouvidos. Agradeço as minhas bandas que nasceram e morreram no tempo da universidade e os ótimos músicos com quem toquei nelas. Também, um grande agradecimento a toda comunidade de software livre por proporcionar tantos projetos e problemas a serem resolvidos (vide o *X Window System*).

Sumário

Lista de Figuras

Capítulo 1

Introdução

Ao longo dos últimos anos assistimos um crescimento acelerado da importância das informações armazenadas em meios digitais. A questão da vulnerabilidade do ambiente digital é cada vez mais preocupante, motivando a necessidade de sistemas que preservem estas informações a longo prazo. Um Sistema de Preservação Digital (PD) tem como objetivo garantir o armazenamento e a integridade de grandes quantidades de dados por longos períodos de tempo [1]. Acervos de bibliotecas, museus e até documentos importantes de organizações governamentais são alguns dos principais candidatos à PD.

Uma das preocupações da preservação digital é a maneira com a ao qual as informações são armazenadas. Cabem aos sistemas de arquivamento tal responsabilidade. Uma boa estratégia de arquivamento digital deve contar com sistemas de baixo custo que permitam replicar grandes volumes de dados e com mecanismos de auditoria para verificar a integridade dos mesmos. A replicação da informação em múltiplos repositórios de armazenamento é uma alternativa comumente utilizada para garantir o seu arquivamento a longo prazo [2].

Neste contexto, as redes *Peer-to-Peer* (P2P) aparecem como uma abordagem promissora, visto que são altamente escaláveis para a distribuição e recuperação de dados. Entretanto, o modelo P2P de computação por si não trata questões relacionadas a alta disponibilidade e confiabilidade de dados da maneira necessária neste tipo de ambiente. Para isso, é necessário estabelecer estratégias de replicação que irão garantir estes requisitos.

Um dos desafios da replicação confiável de dados em P2P é na seleção dos repositórios de armazenamento necessários para a criação das réplicas da informação a ser preservada. Nem sempre um determinado conjunto de repositórios garante a confiabilidade necessária para esta informação. Naturalmente os repositórios falham por diversas razões, comprometendo a preservação a longo prazo dos dados. Além disso, diferentes repositórios têm diferentes confiabilidades. O LOCKSS (*Lots of Copies Keep Stuff Safe*) [3] e o BRICKS [4] são exemplos de sistemas de arquivamento confiável dos dados através de P2P. Ambos sistemas tratam seus repositórios com uma única probabilidade de falha, o que não modela exatamente a realidade.

1.1 Proposta deste Trabalho

Neste trabalho propomos um modelo de replicação confiável de dados para sistemas de arquivamento digital. No modelo proposto, para cada repositório é associada uma métrica de confiabilidade referente à probabilidade de que os dados não sejam perdidos ou danificados em um determinado período de tempo. Assim, considerando que cada item (informação digital) é replicado em um subconjunto de repositórios, é possível determinar a confiabilidade deste item. Conforme a necessidade pelo tempo de preservação, é desejável que itens tenham diferentes confiabilidades para o arquivamento.

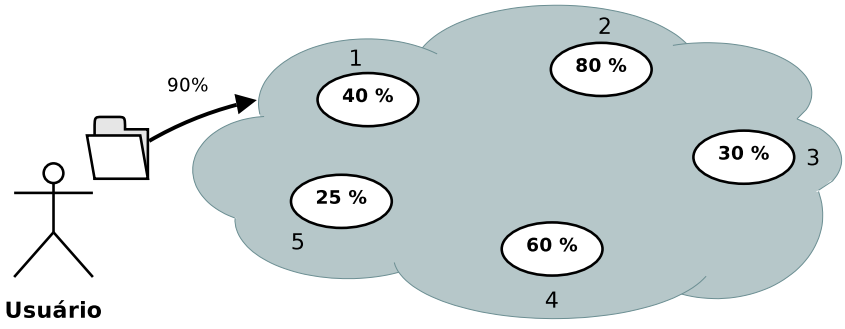

Figura 1.1: Repositórios rotulados com probabilidades de confiabilidade.

Como exemplo, a Figura 1.1 ilustra uma rede hipotética com cinco repositórios, identificados por 1, 2, 3, 4 e 5, e com confiabilidade respectivamente de 40%, 80%, 30%, 60%

e 25%. Digamos que um usuário queira inserir um item na rede com uma confiabilidade desejada de 90%. Uma estratégia ingênua seria replicar este item na ordem dos identificadores dos repositórios até atingir a confiabilidade desejada. Com uma única réplica no repositório 1, o item teria uma confiabilidade de 40%. Adicionando mais uma réplica no repositório 2, a confiabilidade desejada seria de $1-(0.6\times0.2) = 88\%$ e ainda não atingiria a confiabilidade desejada de 90%. Ao replicar o item no repositório 3, o mesmo estaria com uma confiabilidade de $1-(0.6\times0.2\times0.7) = 91.6\%$, chegando na confiabilidade desejada. Uma outra estratégia de replicação poderia começar selecionando os repositórios mais confiáveis e criar réplicas neles até atingir a confiabilidade desejada pelo usuário.

O modelo proposto implica em não considerar um grau de replicação fixa para cada item armazenado no sistema, permitindo a criação de algoritmos que otimizem a utilização dos recursos dos repositórios. Assim, propomos três algoritmos para determinar o subconjunto de repositórios e atender a confiabilidade desejada de um item inserido. O algoritmo Guloso é motivado pela minimização do número de réplicas geradas com adaptações para gerar um bom balanceamento da carga (réplicas) entre os repositórios; o algoritmo Aleatorizado é motivado pelo balanceamento de carga e justificado por resultados teóricos de *ball-and-bins* [5], e; o algoritmo do Subconjunto Ideal resolve o problema sem priorizar a minimização de réplicas ou o balanceamento. Os três algoritmos foram avaliados através de simulações.

Acreditamos que é possível implementar um sistema de arquivamento digital totalmente distribuído aplicando o modelo e os algoritmos de criação de réplicas nas redes P2P. Portanto, este trabalho também apresenta a concepção de um sistema de arquivamento digital sobre uma rede P2P estruturada. Neste sistema a comunicação dos nodos (repositórios) é organizada em uma tabela *hash* distribuída [6, 7, 8, 9] e múltiplas funções *hash* são utilizadas para selecionar os repositórios que irão guardar as réplicas de cada item armazenado. O sistema foi avaliado através de experimentações em um ambiente real.

1.2 Organização do Texto

Esta dissertação está organizada da seguinte maneira. O capítulo 2 apresenta uma introdução aos conceitos de preservação digital e sistemas de arquivamento. O capítulo 3 apresenta as redes P2P e as principais estratégias de replicação existentes nestas redes. O modelo de replicação para sistemas de arquivamento, os algoritmos de criação de réplicas e a simulação destes algoritmos são apresentados no capítulo 4. O capítulo 5 apresenta um sistema de arquivamento digital, baseado no modelo do capítulo 4, e implementado sob a arquitetura das redes P2P estruturadas. Neste mesmo capítulo apresentamos uma experimentação do sistema de arquivamento em um ambiente real. No capítulo 6 é feita a conclusão e os trabalhos futuros são apresentados.

Capítulo 2

Preservação Digital

Um *objeto digital* pode ser definido como toda e qualquer informação que possa ser representada através de uma sequência de dígitos binários [10]. Documentos de texto, imagens, bases de dados, vídeo, áudio e páginas *Web* são alguns exemplos do que podemos considerar um objeto digital. A *Preservação Digital* (PD) é o conjunto de atividades ou processos responsáveis por garantir o acesso continuado a longo prazo aos objetos digitais e a todo o patrimônio cultural existente em formato digital [1]. Neste contexto, o tempo em que as informações devem ser preservadas torna-se um problema chave, uma vez que mídias de armazenamento e componentes de *hardware* e *software* possuem tempo de vida pré-determinados.

Os termos *arquivamento*, *bibliotecas* e *preservação* têm significados próximos porém não idênticos [11]. Quando nos referimos a arquivamento estamos falando do lugar utilizado para guardar dados públicos, enquanto ao nos referirmos a bibliotecas falamos do lugar em que pessoas têm acesso aos materiais. Preservação, por sua vez, é o ato de manter objetos longe da destruição e de danos. Desta maneira, definimos um sistema de preservação digital como um sistema capaz de preservar e dar o acesso a objetos digitais por um longo período de tempo.

Sistemas de preservação digital possuem algumas características que os diferenciam dos sistemas tradicionais de *backup* ou de armazenamento de dados. Sistemas de PD devem operar de maneira correta durante décadas ou séculos. Seus objetos devem estar acessíveis

de maneira legível para o usuário mesmo que o *software*/*hardware* originalmente utilizado para escrevê-lo esteja extinto. Objetos digitais nestes sistemas devem ter identificadores únicos que façam a distinção entre eles a longo prazo. Tais objetos não devem ser apagados e raramente são atualizados. Por fim, é desejável que estes sistemas sejam baratos de construir e manter.

Um sistema de PD utiliza métricas de *confiabilidade* para avaliar seus objetos [12]:

1. **Confidencialidade**: Indica que objetos digitais não podem ser acessados por usuários não autorizados. O primeiro passo para garantir a confidencialidade é identificar usuários propriamente com autenticação, mas outras técnicas também devem ser consideradas, como por exemplo a criptografia dos objetos digitais para previnir ataques maliciosos.

2. **Integridade**: sob hipótese nenhuma usuários de sistemas de PD devem ler objetos sujeitos a modificações não autorizadas. O objetivo da métrica de integridade é avaliar objetos digitais consistentes mesmo em face de modificações inadvertidas.

3. **Autenticidade**: a autenticidade de um objeto digital é definido como a habilidade de definir a *identidade* ao objeto e validar sua integridade. A identidade de um objeto consiste do nome de um arquivo, nome de um autor, a data de criação e etc. Metadados de identidades de objetos digitais são frequentemente utilizados e organizados de modo que uma busca pelo objeto seja acelerada. Portanto, autenticidade é o estado de um objeto digital sem corrompimento e modificações inesperadas.

Nenhuma dessas métricas é exclusiva e todas devem ser consideradas sob um determinado grau de importância conforme o objetivo do sistema. As métricas de confiabilidade são sobrecargas para o sistema como um todo e podem degradar seu desempenho. Entretanto, sistemas de PD não necessariamente precisam operar de maneira rápida, fazendo com que os arquitetos destes sistemas geralmente façam um balanço entre a necessidade pelo desempenho ou a confiabilidade de dados no sistema.

O restante deste capítulo está organizado da seguinte maneira. A seção 2.1 expõem as ameaças constantes em sistemas de PD. Na seção 2.2 são apresentadas algumas estratégias

e técnicas para o projeto destes sistemas. Na seção 2.3 apresentamos os modelos de repositórios comumente utilizados nos sistemas de PD.

2.1 Ameaças à Preservação Digital

Sistemas de preservação digital estão sujeitos a várias falhas e ameaças em seus conteúdos. Apresentamos a seguir os tipos de falhas comumente encontradas em ambientes de arquivamento a longo prazo [13, 14]:

- **Desastre em larga escala**: Durante o período de vida de um objeto digital é necessário estar atento a desastres naturais de larga escala como inundações, fogo, terremótos e guerras. Replicações remotas podem ser realizadas para reduzir o dano provocado por desastres deste tipo.

- **Erro humano**: O erro humano pode afetar o *hardware* utilizado para a preservação (por exemplo, perdendo as mídias de armazenamento), o *software* (por exemplo, desinstalando um *driver* necessário para o sistema) ou a própria infraestrutura (por exempo, removendo uma fonte de energia da máquina utilizada para preservação).

- **Falha nos componentes**: Deve-se esperar falhas em componentes de *hardware*, tal como a falta de energia ou simplesmente a falha numa placa controladora. Também deve-se esperar todos os tipos de falhas em componentes de *software* (desde simples *bugs* até falhas na *firmware* de alguma controladora) que comprometam os dados armazenados. Componentes de rede pertencentes a empresas externas podem não existir após décadas. Portanto, sistemas de preservação devem também se precaver e antecipar eventuais falhas transientes ou irreversíveis a serviços externos, como por exemplo DNS ou URL persistentes.

- **Falha na mídia**: Mídias de armazenamento são componentes vitais para sistemas de PD. Elas estão sujeitas a uma deterioração gradual dos dados, conhecida como *bit rot* [14], causando danos irreversíveis. Em particular, *bit rot* é um grande problema

pois o dado pode estar corrompido sem ser notado pelo fato do setor magnético do disco que o armazena estar inacessível por algum período.

- **Obsolecência de *hardware***: mídias e componentes de *hardware* podem se tornar obsoletos no sentido de não serem mais capazes de se comunicarem uns com os outros. Em particular, discos removíveis têm um grande histórico neste quesito de obsolescência, uma vez que a mídia ainda pode estar funcional porém dificilmente dispositivos de leitura serão encontrados no mercado para utilizá-la.

- **Obsolecência de *software***: Similarmente, componentes de *software* podem tornar-se obsoletos. Todo o tipo de material digital tem obrigatoriamente de respeitar as regras de um determinado formato em que foi codificado. À medida que o *software* vai evoluindo, os formatos por ele produzidos também vão sofrendo alterações. Quando os dados codificados (formatados) ainda podem ser acessíveis mas não decodificados do formato de armazenamento numa forma legível então é caracterízado uma obsolescência de formato [15]. Um exemplo típico é o editor de textos *Microsoft Word*. Documentos de texto escritos em versões antigas deste editor são conhecidos por não funcionarem em versões mais novas.

- **Ataques**: Todos sistemas conectados a uma rede pública (Internet, por exemplo) estão vulneráveis a ataques maliciosos, vírus ou *worms*. Por razões emocionais ou econômicas, operadores internos que têm acesso autorizado podem causar danos no sistema.

- **Falha econômica**: Quando a informação está no formato digital existe um custo contínuo de energia, banda de rede, administração do sistema, dentre outros. Deve ser antecipado um orçamento necessário para realizar a preservação digital e esperar variações do mesmo.

- **Falha organizacional**: Além da tecnologia empregada, a preservação digital deve também cuidar dos fatores organizacionais. Organizações podem ir a falência ou simplesmente mudar sua missão comprometendo os objetos preservados.

Um sistema de preservação digital deve considerar estas ameaças com o objetivo de garantir a confiabilidade dos seus objetos. A próxima seção apresenta algumas das estratégias mais utilizadas para realizar a preservação de objetos na forma digital.

2.2 Estratégias para Preservação Digital

Um sistema de preservação digital emprega a combinação de várias técnicas para beneficiar a duração a longo prazo de seus objetos. Tais técnicas podem ser enquadradas em três diferentes estratégias: 1) redundância, utilizada para aumentar a confiabilidade dos dados do sistema; 2) migração e emulação com o objetivo de solucionar problemas de obsolescência de *software* e formatos, e; 3) auditoria, que garante a autenticidade e integridade dos objetos digitais.

2.2.1 Redundância

Redundância é a estratégia utilizada para a multiplicação de componentes críticos de um sistema com a intenção de aumentar a confiabilidade do mesmo. Geralmente ela é aplicada através da criação de múltiplas cópias ou a partir de mecanismos que detectam falhas. A redundância é utilizada em diferentes níveis na prática: 1) dentro da mídia de armazenamento; 2) dentro do repositório de armazenamento, ou; 3) geograficamente distribuída e replicadas em vários repositórios [12]. Existem várias técnicas para fazer a redundância tais como espelhamento, paridade, ECC (*Error Correction Code*), CRC (*Cyclic Redundancy Check*), assinatura e *erasure code*. Porém a técnica mais utilizada é a replicação.

Chamamos de replicação a técnica utilizada para guardar cópias de objetos, as *réplicas*, em lugares diferentes com o objetivo de melhorar a confiabilidade, a tolerância a falhas e/ou o desempenho do sistema. Esta técnica pode ser empregada em um bloco de dados ou em um pedaço de dados de tamanho fixo. Para realizar a preservação de objetos no formato digital é necessário que réplicas sejam colocadas em diferentes dispositivos de armazenamento ou, preferivelmente, em diferentes repositórios.

A replicação é amplamente utilizada devido à sua simplicidade de implementação. Alguns sistemas de PD utilizam um número fixo de réplicas e outros ajustam este número dinamicamente.

2.2.2 Migração e Emulação

Uma das propriedades da preservação digital é manter os objetos preservados acessíveis mesmo em eventuais obsolescências de *software* e formatos. Existem duas estratégias para garantir estes requisitos: migração e emulação.

Podemos definir *migração* como o conjunto de atividades projetadas com o intuito de realizar transferências periódicas de materiais digitais de uma geração tecnológica para uma geração subsequente [16]. Uma importante característica é que nesta estratégia não é necessário manter a aplicação original. A utilização de *formatos padrões*, principalmente *formatos abertos*, é uma estratégia comum em migração. Alguns exemplos de padrões são ASCII (texto puro), PDF (*Portable Document Format*), XML (*Extensible Markup Language*) e mais recentemente o ODF (*OpenDocument Format*). No entanto, mesmo com a utilização de padrões, formatos continuam se desenvolvendo e muitas vezes fabricantes tendem a seguir seus próprios estilos de codificação.

O maior problema da técnica de migração é a necessidade por uma especializada investigação no formato original e no formato alvo do objeto a ser preservado. Outro problema é a grande probabilidade de acontecer perdas de informação na migração de formato. A razão disto se deve às incompatibilidades existentes entre os formatos de origem e destino e/ou também à utilização de conversores incapazes de realizar as suas tarefas adequadamente [15]. Por exemplo, um documento do *Microsoft Word* transformado numa imagem *bitmap* perde informações relacionadas a diagramação da página.

Emulação é vista como uma alternativa para migração quando o objetivo é garantir a obsolescência de *software* e formato. Nesta técnica, o ambiente de operação em que o objeto digital foi originalmente criado é recriado através do uso de emuladores capazes de reproduzir o comportamento de uma plataforma de *hardware* ou *software*. A vantagem nesta técnica é que não existe perda de informações pois o formato do objeto digital não

é modificado.

Existem essencialmente dois tipos de emuladores: emuladores de sistemas operacionais e de *hardware*. Os primeiros estão focados na reprodução de um sistema operacional por completo permitindo a execução de diversas aplicações no contexto de um único emulador (por exemplo, o Wine [17]). Os segundos reproduzem o comportamento de uma plataforma de *hardware*, possibilitando que vários sistemas operacionais e aplicações correspondentes possam ser executados (por exemplo, VMware Workstation [18]). A estratégia de emulação não é amplamente utilizada justamente devido a dificuldade na construção dos ambientes mencionados.

2.2.3 Auditoria

A *auditoria* em um sistema de arquivamento é necessária para garantir a autenticidade e integridade de objetos digitais. A técnica de *auditoria periódica* faz a auditoria em objetos digitais em um intervalo fixo de tempo [19]. Nesta técnica, para um sistema de preservação digital baseado em repositórios providos de disco rígido, uma auditoria iria requerer uma operação de acesso no disco (*Power On Cycle*), ao qual é confirmada que danifica a confiabilidade do disco com o passar do tempo [20]. Assim, é desejável que seja aproveitada a carga de trabalho normal de um disco para realizar a auditoria.

Chamamos de *auditoria oportunística* a técnica que se aproveita de outras operações comuns cotidianas no disco para fazer a auditoria e evitar adicionais acessos no mesmo. Quando o disco é acessado, uma porção do disco com intervalo de tempo excedente o tempo médio de falhas é auditada imediatamente. Para mídias com acesso frequente ao disco, esta técnica não acarreta nenhum custo adicional. Entretanto, para discos com acesso não frequente, esta técnica não pode ser utilizada para garantir a integridade dos dados.

Sistemas de preservação digital baseado em comunidades se aproveitam da natureza distribuída dos seus objetos para fazer a auditoria e descobrir eventuais falhas. Por exemplo, o LOCKSS [3] emprega um mecanismo baseado em votação para identificar objetos falhos e informar outros nodos sobre isto. Entretanto, o tempo de auditoria para

cada cópia local não é levado em conta: cada vez será mais demorado fazer uma varredura completa de auditoria no disco rígido [14]. Por exemplo, é necessário 20 dias para auditar um único disco rígido de 1 Petabyte com uma taxa de 600 MB/s de acesso.

2.3 Modelos de Repositórios

Existem duas categorias diferentes relacionadas ao lugar em que as réplicas são armazenadas e a maneira ao quais repositórios são dispostos em um sistema de arquivamento digital [11]. A categoria de modelos *centralizados* é caracterizada principalmente pela existência de um pequeno número de repositórios robustos e altamente controlados. Nesta categoria, cada repositório é responsável por realizar todo trabalho de preservação. Os repositórios requerem *hardware* potente e suporte técnico especializado, consequentemente, altos custos para manutenção são necessários. Soluções tradicionais de *backup* ou armazenamento de dados, tais como sistemas de arquivos replicados e discos RAID [21] se enquadram nesta categoria.

Em contrapartida, na categoria dos *descentralizados* o conteúdo é armazenado em um grande número de repositórios "livres", ou seja, com um controle escasso sobre objetos. Nesta categoria, cada repositório é responsável apenas por uma parte de todos objetos do sistema o que torna o custo da preservação compartilhado entre os vários participantes. Consequentemente o *hardware* utilizado tem um custo relativamente baixo, pelo fato da informação ser distribuída. Além disso, é necessário um conhecimento técnico baixo para manter o *hardware* e o *software*.

Na categoria dos sistemas de arquivamento descentralizados, uma alternativa comumente utilizada para garantir a confiabilidade dos dados é através da replicação dos dados em múltiplos repositórios executados por organizações autônomas. Esta é a forma encontrada por bibliotecas físicas para preservar seus conteúdos e que tem sido estendida para preservação de dados digitais através de diversos projetos [3, 22, 4]. Sistemas deste tipo são classificados como *sistemas de replicação baseado em comunidades* [23].

Sistemas de replicação baseado em comunidades se beneficiam com a redundância herdada de sistemas distribuídos para manter os dados seguros. A replicação garante

que a informação seja recuperada a partir de diferentes localidades, o que aumenta a confiabilidade e a disponibilidade do conteúdo. O LOCKSS, o SAV e o BRICKS são típicos exemplos de sistemas baseados em comunidades.

O LOCKSS, por exemplo, constrói uma rede de computadores que cooperam entre si com o intuito de detectar e reparar eventuais problemas de integridade do material digital. O LOCKSS organiza seus nodos a participarem de pesquisas de opinião *opinion polls*, fornecendo informações – os votos – sobre as condições atuais dos seus objetos digitais. Se o conteúdo em um nodo estiver corrompido ou incompleto, este irá "perder" a pesquisa. Sendo assim, seu conteúdo será corrigido baseado no material dos nodos "vencedores".

Capítulo 3

Redes Peer-to-Peer

No modelo de computação cliente/servidor existem dois papéis bem distintos das máquinas que compõem sua rede [24]. Os clientes são os responsáveis por iniciar a comunicação e fazer requisições com um ou mais servidores. Os servidores esperam, tratam e retornam as requisições aos clientes. Este modelo é um exemplo de arquitetura centralizada onde a rede depende de pontos centrais para prover os serviços. Atualmente a maioria das aplicações da Internet, tais como *e-mail*, banco de dados e navegação *Web* são baseadas neste modelo.

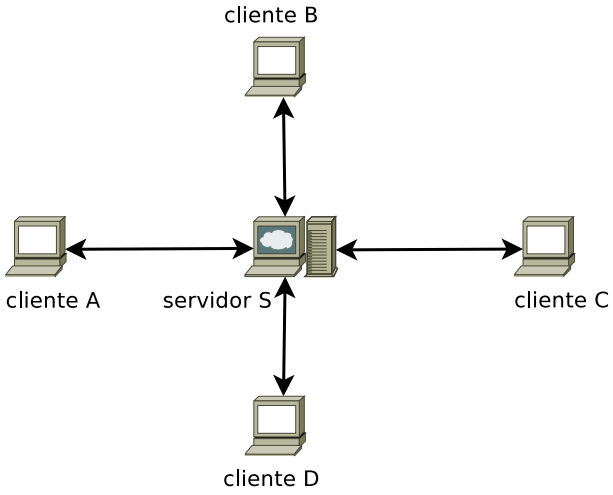

Figura 3.1: Modelo de computação cliente/servidor.

A Figura 3.1 ilustra o modelo cliente/servidor no qual os clientes A, B, C e D se comunicam com o servidor S. Podemos observar que quanto mais a rede cresce, ou seja,

mais clientes são adicionados, a pressão pelo ponto central (o servidor) também aumenta. Isto torna o servidor um ponto de contenção e sua falha comprometeria a rede como um todo.

O termo *peer-to-peer* (P2P) se refere ao modelo de computação distribuída onde dois ou mais participantes são capazes de espontaneamente colaborar sem a necessidade de um coordenador central através do uso de sistemas de comunicação [25]. Neste modelo, todos os participantes são ao mesmo tempo clientes e servidores. Tais participantes são geralmente chamados de nodos ou *peers*.

Diversas definições de sistemas baseados no modelo P2P podem ser encontradas na literatura [26]. Uma delas define sistemas P2P puros, que tem como característica serem totalmente descentralizados. Em um sistema P2P puro, existe uma equivalência com relação a funcionalidade e a execução de tarefas em todos os nodos da rede. No entanto, este tipo de definição não é totalmente abrangente, já que não engloba sistemas que utilizam "supernodos" (nodos que funcionam como mini-servidores, responsáveis por funções específicas), nem sistemas que utilizam algum servidor central para realizar tarefas secundárias.

Uma definição amplamente aceita diz que sistemas P2P são uma classe de aplicações que levam vantagem de recursos computacionais, como armazenamento, processamento e conteúdo disponíveis na Internet [27]. Porém, devido seu alto nível de abstração, esta definição engloba outros tipos de aplicações, como por exemplo computação em Grid [28], e sistemas que contam com servidores centralizados.

De acordo com Androutsellis-Theotokis e Spinellis [29] a definição de um sistema P2P deve priorizar as seguintes características: (i) o compartilhamento de recursos computacionais de forma direta; (ii) o fato de servidores centralizados serem utilizados apenas para tarefas específicas e não para operações básicas; (iii) ter a habilidade para tratar instabilidade da rede e conectividade variada e; (iv) tolerância a falhas e auto-organização como um todo. Neste contexto, pode-se dizer que sistemas *peer-to-peer* são sistemas distribuídos que consistem de nodos inter-conectados em uma rede, aptos a se auto-organizar, com propósito de compartilhamento de recursos computacionais e capacidade de se adaptar a

falhas e nodos transitórios, mantendo conectividade e desempenho aceitáveis, sem requerer intermediação de um servidor centralizado [29]. Tais características motivaram a implantação do modelo P2P em diversos sistemas distríbuidos notórios, tais como Gnutella [30], SETI@home [31], Freenet [32], Jabber [33], eMule [34], entre outros.

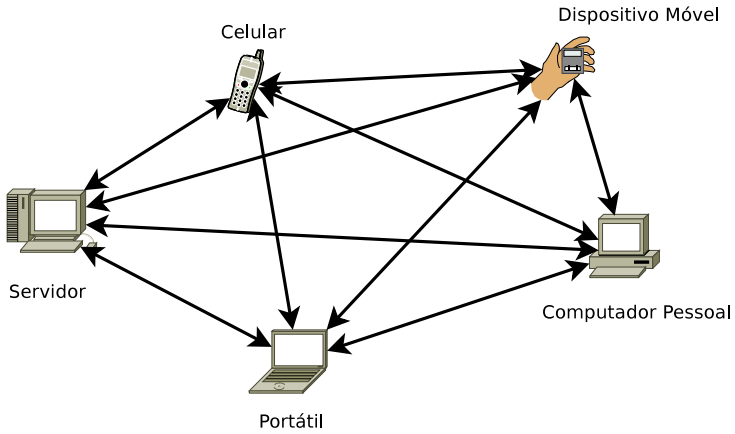

Figura 3.2: Modelo de computação *peer-to-peer*.

A Figura 3.2 ilustra o modelo P2P, na qual podemos observar vários participantes interagindo uns com os outros, cada qual contribuindo com uma determinada quantidade de recursos. Quanto mais nodos entram na rede, mais a capacidade da rede aumenta. Dessa maneira, o modelo P2P garante um fator de *escalabilidade* impossível de conseguir no cliente/servidor.

Um problema em redes P2P diz respeito à natureza transiente dos nodos (*churn*). Os nodos, bem como os roteadores existentes entre eles, estão sujeitos a falhas e frequentemente entram e saem da rede. Assim, é desejável que uma rede P2P seja capaz de adaptar-se a esta instabilidade e variações das conexões.

Sistemas no modelo P2P frequentemente utilizam uma rede virtual para implementar serviços não disponíveis na rede física. Denominamos esta rede virtual como rede de sobreposição (*overlay network*). A Figura 3.3 mostra uma rede de sobreposição construída em cima de uma rede física. Na figura as mensagens entre os nodos da rede de sobreposição (círculos pretos) seguem a topologia lógica em anel desta rede, porém fisicamente passam através das conexões existentes entre nodos e roteadores que formam a rede física (círculos brancos).

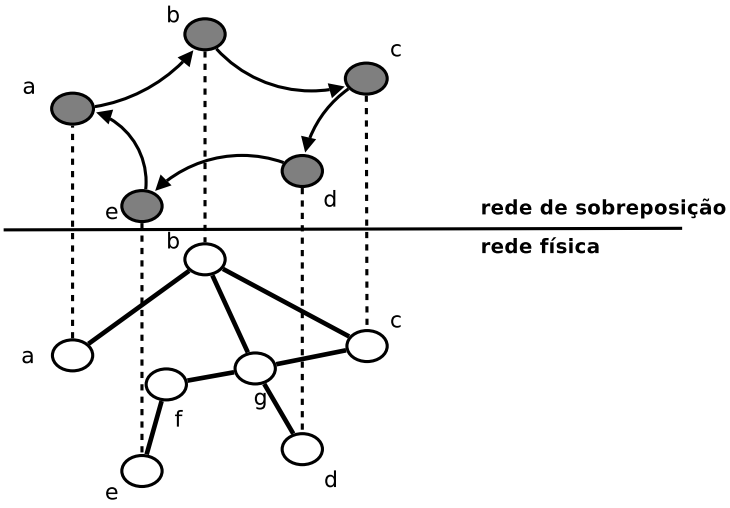

Figura 3.3: Uma rede de sobreposição construída sobre uma rede física.

Nas próximas seções mostraremos a classificação das aplicações nas redes P2P e detalharemos a arquitetura em relação a sua topologia, estrutura e grau de centralização. Além disso, apresentaremos as estratégias comumente utilizadas para realizar a replicação neste tipo de rede.

3.1 Classificação das Aplicações

As redes P2P são popularmente conhecidas pelo seu uso na transferência de arquivos de músicas, videos, imagens e também no uso por jogos de entretenimento. Entretanto, estas são somente algumas das aplicações das redes P2P. Conforme a classificação de Androutsellis-Theotokis e Spinellis [29], podemos dividir as redes P2P nas seguintes categorias de aplicações:

- **Comunicação e Colaboração:** Nesta categoria estão sistemas que provêm uma infraestrutura para a realização de comunicação e colaboração direta entre nodos da rede. Exemplos incluem mensageiros instantâneos e salas de bate-papo, tais como IRC, ICQ, MSN e Jabber [33].

- **Computação Distribuída:** Inclui sistemas ao quais objetivam tirar o proveito do poder de processamento (CPU) de nodos na rede. Isto é feito ao "quebrar" uma tarefa muito grande em pequenas unidades que são distribuídas em diferentes

nodos, que por sua vez as computam e retornam o valor. SETI@home [31] é uma das aplicações mais conhecidas nesta categoria.

- **Suporte a Serviços na Internet:** Inclui várias aplicações baseadas em *peer-to-peer* com suporte a serviços na Internet. Exemplo destas aplicacões incluem alguns sistemas multicast [35] e serviços de segurança por exemplo contra ataques *denial of service* ou vírus [36].

- **Sistemas de Banco de Dados:** Nesta categoria estão sistemas de banco de dados distribuídos apoiados na arquitetura de redes P2P. O Modelo Relacional Local [37] e o PIER (*Peer-to-Peer Information Exchange and Retrieval*) [38] são alguns exemplos desta categoria.

- **Distribuição de Conteúdo:** De maneira simples, um sistema P2P de distribuição de conteúdo cria uma mídia de armazenamento distribuída que permite a publicação, consulta e recuperação de conteúdo pelos membros da sua rede. Atualmente, a maior parte dos sistemas P2P existentes enquadram-se nesta categoria, onde pode-se encontrar desde aplicações de compartilhamento de arquivos, até sistemas que disponibilizam uma rede de distribuição de conteúdo segura e com funcionalidades de publicação, organização, indexação, consulta, recuperação e alteração de dados de forma eficiente. Como exemplo desta categoria pode-se citar sistemas como Gnutella [30], Kazaa [39], eMule [34], Freenet [32], entre outros.

3.2 Arquitetura e Roteamento

As redes P2P são geralmente separadas em duas categorias conforme sua topologia, grau de centralização e roteamento das mensagens.

3.2.1 Não-estruturada

Na categoria das redes *não-estruturadas* não existe nenhum controle sobre a topologia da rede ou sobre como as informações são inseridas nela. Cada nodo apenas conhece seus

vizinhos e um mecanismo de força bruta é utilizado para fazer a busca pela informação, quando necessário.

O mecanismo de inundação (*flooding*) é o mais utilizado para realizar a busca pela informação, onde a busca é propagada para toda a vizinhança de um determinado nodo com uma certa profundidade. Os algoritmos de inundação tendem a causar graves implicações na escalabilidade da rede pela necessidade de muitas mensagens serem enviadas até que a busca seja completada. Além disso, muitas mensagens duplicadas são recebidas pelos nodos em uma busca e mesmo utilizando algumas técnicas para diminuí-las, como o *random walk* [40], a escalabilidade acaba sendo comprometida, aparecendo como um grande entrave para algumas aplicações P2P.

A incompleteza do resultado em redes não-estruturadas também é um problema. Frequentemente um dado raro não é encontrado devido a dificuldade de buscar informações em nodos nesta topologia. Por outro lado, a topologia não-estruturada é ótima quando os conteúdos da rede seguem uma distribuição do tipo *Zipf* [41], isto é, com poucos objetos muito populares. As redes não-estruturadas também são apropriadas para acomodar população de nodos transientes pois não existe nenhuma regra utilizadas na entrada/saída de um nodo. Napster [42], Gnutella [30] e Kazaa [39] são exemplos clássicos de sistemas nesta categoria.

3.2.2 Estruturada

As redes P2P *estruturadas* são uma alternativa para suprir os problemas de escalabilidade na busca das não-estruturadas. Nestas redes a topologia é bem definida e o conteúdo sempre colocado em locais determinados, resultando em buscas eficientes.

Redes P2P estruturadas frequentemente utilizam tabelas *hash* distribuídas (DHT – *distributed hash table*) como substrato em suas operações básicas para prover sistemas distribuídos escaláveis e auto-gerenciáveis [6, 7, 8, 9]. Da mesma maneira que em uma tabela *hash* tradicional, a DHT contém pares chave/valor referenciados como itens ou objetos, porém a tabela de roteamento na DHT é distribuída. Em um cenário típico, o usuário da rede tem uma chave para a qual deseja buscar seu valor associado. O usuário

informa a chave para qualquer um dos nodos, que executa a operação de busca e retorna o valor associado àquela chave. A representação do par chave/valor pode ser arbitrária. Por exemplo, a chave pode ser uma palavra ou um objeto. Similarmente, o valor pode ser uma palavra, um número ou uma representação binária de um objeto arbitrário. A representação dependerá da particularidade de cada aplicação [43].

Para realizar a entrega de mensagens, as DHTs possuem mecanismos de roteamento. Estes mecanismos permitem que o nodo responsável por uma determinada chave possa ser encontrado em $O(log\ n)$ na maioria das DHTs, onde n é o número de nodos na rede. O CAN [7] tem um desempenho um pouco diferente, na ordem de $O(\frac{d}{4}n^{\frac{1}{d}})$, onde d representa o número de dimensões empregadas na implementação do protocolo.

Sistemas estruturados têm problemas com populações extremamente transientes. Manter a estrutura das tabelas de roteamento é relativamente custoso quando acontece uma entrada ou saída de um nodo em todos os sistemas que empregam DHT. Outro problema é relacionado as consultas, que são restritivas. O conteúdo pesquisado nas DHTs deve corresponder exatamente ao valor da chave e portanto buscas complexas não são suportadas de maneira simples [44].

Um dos protocolos DHT mais conhecido é o Chord [8], que utiliza um espaço circular de identificadores, com tamanhos de *m-bits*, para fazer a associação entre nodos e chaves. O comprimento do identificador deve ser grande o suficiente para tornar negligível a probabilidade de uma mesma *hash* ser utilizada por diferentes chaves. Desta maneira, o espaço circular tem tamanho 2^m. O número de saltos esperados para rotear uma mensagem no Chord é $\frac{1}{2}\ log_2 n$, em um sistema com n nodos.

A Figura 3.4 ilustra o Chord e mostra um círculo de identificadores com $m = 3$ e três nodos conectados (círculos brancos). O conjuto de chaves é $\{1, 2, 6\}$ e elas estão associadas respectivamente com os nodos 0, 1 e 3. Neste exemplo, pelo fato do sucessor da chave 1 entre os nodos da rede ser o próprio nodo 1, a chave 1 é associada com o nodo 1. Similarmente, o sucessor da chave 2 é o 3, o primeiro nodo movendo no sentido horário a partir do 2, no círculo de identificadores. Para a chave 6, o sucesssor é encontrado no nodo 0.

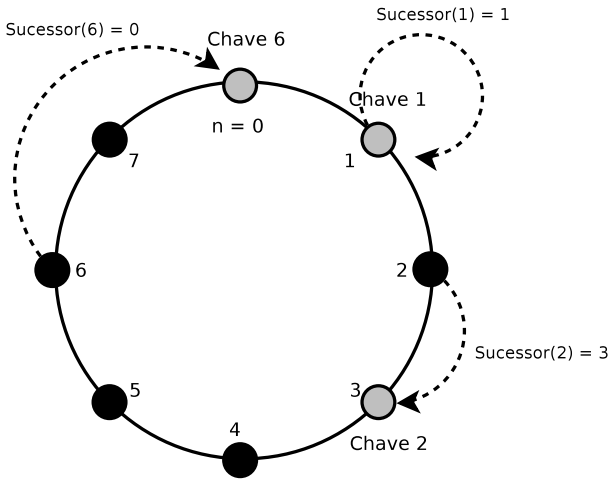

Figura 3.4: Círculo de identificadores do Chord.

Para localizar as chaves de maneira mais eficiente, as mensagens no Chord são encaminhadas no espaço circular de identificadores com a ajuda de uma tabela de roteamento chamada *finger table*. Cada nodo n mantém uma *finger table* com m entradas contendo os identificadores e os endereços IP de seus vizinhos. A i-ésima entrada da tabela de n aponta ao sucessor do identificador $[(n + 2^{i-1}) \bmod 2^m]$, onde $1 \leq i \leq m$. Como exemplo, consideremos o nodo 0 ($n = 0$) na Figura 3.5. As entradas na *finger table* do nodo 0 são computadas da seguinte maneira:

- $i = 1:$ sucessor $[(0 + 2^0) \bmod 2^3 \rightarrow$ sucessor $(1) = 1$
- $i = 2:$ sucessor $[(0 + 2^1) \bmod 2^3 \rightarrow$ sucessor $(2) = 3$
- $i = 3:$ sucessor $[(0 + 2^2) \bmod 2^3 \rightarrow$ sucessor $(4) = 0$

Suponha que o nodo 1 procura pela chave $k = 6$, no qual é armazenada no nodo 0. O nodo que inicia a busca encontra em sua *finger table* o nodo n' com o maior identificador tal que n' seja localizado entre n e k no círculo. Em outras palavras, dado que 3 é o maior identificador apontado pela *finger table* de 1, localizado entre 1 e 6 no círculo, então $n' = 3$. Se tal nodo existe, a busca é encaminhada para n', que por sua vez se torna n e executa a mesma operação de busca. Caso contrário, o nodo que atualmente mantém a busca retorna seu sucessor no círculo como sucessor(k).

Outros exemplos de DHTs são o CAN [7], que utiliza uma arquitetura no espaço de

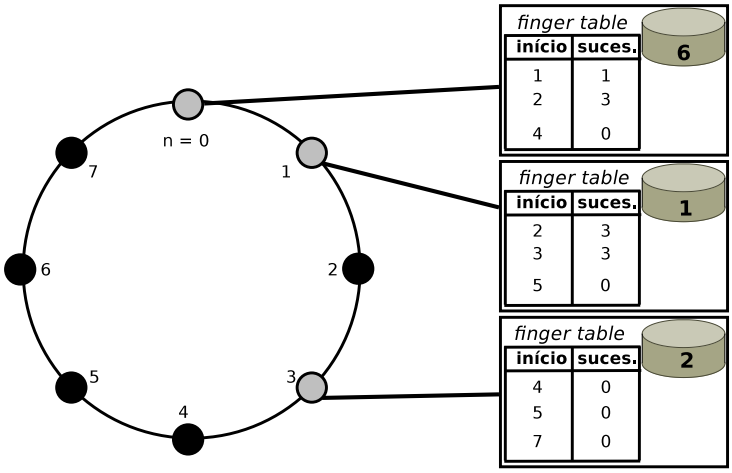

Figura 3.5: *Finger table* do Chord.

coordenadas Cartesiano para organizar os nodos; o Tapestry [9], que de maneira parecida com o Chord utiliza uma abordagem baseada em prefixos de m-bits para determinar os nodos; e o Pastry [6], muito similar ao Tapestry, diferenciando principalmente na sua abordagem de replicação dos objetos.

3.3 Estratégias de Replicação

As redes *peer-to-peer* garantem aos sistemas a escalabilidade em relação a capacidade crescente do número de usuários, automaticamente se adaptando a falhas nos nodos e nas conexões existentes entre eles. Entretanto, não podemos assumir a disponibilidade dos serviços caso um nodo responsável por uma tarefa específica seja desconectado. Uma maneira de contornar este problema é fazendo a cópia das informações deste nodo em outro, ao qual assume as tarefas do nodo desconectado.

A replicação de dados consiste em manter múltiplas cópias de objetos, chamados réplicas, em computadores ou repositórios separados [45]. A replicação é muito importante em sistemas distribuídos pois (i) aumenta a disponibilidade dos objetos; (ii) aumenta o desempenho ao reduzir a sobrecarga de comunicação e aumentando a vazão; e (iii) aumenta a confiabilidade dos dados.

Alguma soluções de replicação assumem dados estáticos e somente de leitura, por exemplo arquivos de música. Outras soluções consideram atualizações, mas simplificam

o gerenciamento das réplicas assumindo que não existam conflitos de atualizações e que somente uma cópia aceita operações de escritas. Aplicações mais avançadas precisam de abordagens mais sofisticadas para tratar operações de escritas em todas as réplicas. Em ambientes de preservação digital as réplicas não são modificadas [46]. Feita uma réplica, ela se torna imutável no sistema. Portanto não estamos interessados em estratégias e mecanismos de atualizações das réplicas.

Sistemas P2P não-estruturados tais como Napster e Gnutella frequentemente utilizam replicação para aumentar a disponibilidade dos arquivos e o desempenho das buscas. A replicação acontece "naturalmente" pelo fato de nodos requisitarem e copiarem cópias de arquivos uns nos outros. Este esquema é chamado de *replicação passiva*. Por sua vez, as P2P estruturadas geralmente empregam esquemas mais complexos de replicação. No restante desta seção descrevemos estratégias de replicação comumente utilizadas em redes P2P estruturadas. Dividimos as estratégias em três categorias: replicação na vizinhança, replicação no caminho e replicacão com múltiplas chaves.

3.3.1 Replicação na Vizinhança

Nas DHTs, nodos guardam e mantêm uma lista de outros nodos na sua tabela de roteamento com informações necessárias para fazer a transmissão das mensagens. Estas listas podem ser úteis para realizar a estratégia de replicação baseada na vizinhança (*neighbor replication*). Podemos separar a replicação na vizinhança em dois tipos:

- **Lista de sucessores** (*sucessor-list*). O Chord e outros protocolos DHT que mantêm uma lista de sucessores podem se beneficiar deste tipo de replicação. A DHT guarda uma réplica em cada um dos r nodos imediatamente localizados após o nodo responsável pela primeira cópia. Se o nodo responsável pela primeira cópia falhar, então o dado estará disponível nos seus r sucessores.

- **Conjunto folha** (*leaf-set*). O roteamento de alguns protocolos, como o Pastry [6] e o Kademlia [47], acontece alternando a direção dos nodos sucessores e antecessores. Assim, de maneira similar a técnica da lista de sucessores, a replicação do conjunto

folha guarda itens do nodo responsável pela primeira cópia nos seus $\lfloor r/2 \rfloor$ próximos sucessores e $\lfloor r/2 \rfloor$ próximos antecessores.

Uma desvantagem desta estratégia é que qualquer requisição feita para uma réplica r específica deve ser primeiro roteada para o nodo responsável pela primeira cópia e somente depois pode ser encaminhada para os r vizinhos. Isto faz com que o primeiro nodo se torne um ponto de contenção, desacelerando algumas operações. Outra desvantagem é que o número de réplicas não pode exceder o tamanho da lista de vizinhos, dado que o número de vizinhos é atrelado ao protocolo DHT. Neste caso um eventual sistema de arquivamento digital poderia não alcançar o fator de replicação desejado para preservar suas informações.

Se um nodo não tem capacidade suficiente para armazenar uma determinada cópia de um objeto, então outros nodos da vizinhança poderiam ser explorados para conseguir um fator de replicação maior [48]. Neste caso algumas heurísticas tais como *random-fit*, *first-fit*, *best-fit* e *worst-fit* poderiam ser aplicadas para escolher o melhor conjunto da vizinhança.

3.3.2 Replicação no Caminho

Replicação no caminho (*path replication*), replica o objeto pelo caminho que uma mensagem de busca da DHT é roteada. Buscas geradas por diferentes nodos tendem a convergir para alguns nodos específicos antes de chegar no nodo que contém a cópia original. A replicação no caminho faz com que as buscas passem por menos nodos e consequentemente sejam mais rápidas.

O propósito dessa estratégia é primordialmente garantir um bom desempenho, pois a antecipação dos objetos em uma busca pode ser encarada como uma espécie de *cache*. O Tapestry [9] se beneficia da replicação no caminho com este objetivo.

Da mesma maneira que na replicação na vizinhança, esta estratégia também é atrelada ao protocolo DHT e o fator desejado de replicação pode não ser alcançado dependendo do número de nodos que estão no caminho.

3.3.3 Replicação com Publicação de Múltiplas Chaves

Nessa estratégia uma chave de um objeto é replicada em r nodos distintos da rede [49]. Nodos precisam conhecer o esquema de replicação para inserir um dado corretamente e ser capaz de buscá-lo mais tarde. Separamos esta estratégia em três grupos.

- **Hash Correlacionado** No *hash* correlacionado (*correlated hashing*), cada objeto com identificador ID é inserido na DHT por uma *hash* do tipo $h(i; ID)$, onde i indica o número da réplica ($1 \leq i \leq r$). A função *hash* é definida da seguinte maneira: $h(1; ID) = ID;\ h(i; ID) = h(i + ID)$, onde "+" é a concatenação, i é o número da réplica e h é a *hash*. Em outras palavras, todas as chaves de um determinado objeto são correlacionadas a primeira (ou alguma outra) chave. O BRICKS [4] usa *hash* correlacionado.

- **Replicação Simétrica**. Outra estratégia proposta para criar múltiplas chaves a partir de único objeto é a replicação simétrica (*symmetric replication*) [50]. A principal idéia é que cada identificador de nodo no sistema seja associado com r outros identificadores. Se o identificador i é associado com j, então o nodo responsável pelo item i deve guardar ambos itens i e j. Similarmente, o nodo responsável por j deve guardar ambos itens de i e j. Por exemplo, se o identificador 0 é associado com os identificadores 0, 4, 8 e 12, então qualquer nodo responsável por qualquer um dos itens 0, 4, 8 ou 12 tem que guardar todos os itens 0, 4, 8 e 12. Portanto, para recuperar o item 0, uma busca pode ser enviada a qualquer um dos nodos responsáveis pelos itens 0, 4, 8 e 12.

  O grande problema desta estratégia é que o número de réplicas sempre tem que ser o mesmo para todos os itens no sistema.

- **Múltiplas Funções Hash**. A idéia no uso de múltiplas funções *hash* é que se r réplicas de um par chave/valor são necessárias, então é aplicado r funções nesse par, obtendo r identificadores e publicando as cópias em r nodos distintos no sistema. CAN [7] e Tapestry usam essa abordagem.

Um dos desafios nesta estratégia é como escolher uma função *hash* ainda não utilizada na hora de criar uma nova réplica e como escolher uma função *hash* já existente na hora de fazer a requisição de um item [51]. Uma maneira simples para criar as múltiplas *hashs* é anexando um "sal" i no final da chave do objeto, onde i indica o número da réplica a ser criada. Suponha que o nome do objeto é *foo*, então $h(foo1), h(foo2), \ldots, h(foor)$ nos daria r *hashs* para este objeto.

Capítulo 4

Um Modelo de Replicação Confiável para Arquivamento Digital

"In wildness is the preservation of the world." –
Henry David Thoreau, *Walking (1862)*

Replicar a informação em *quais* repositórios? Esta é uma pergunta frequente de organizações e sistemas preocupados com o arquivamento digital. Nem sempre um determinado conjunto de repositórios garante a confiabilidade suficiente para a informação, que deve ser preservada por décadas ou até centenas de anos. É comum repositórios falharem por várias razões, comprometendo o arquivamento a longo prazo destas informações. Além disso, é desejável que o sistema de arquivamento seja financeiramente viável e o custeio da estratégia de armazenamento das informações seja praticável. Portanto, um grande desafio dos sistemas de arquivamento digital é como selecionar os repositórios que irão armazenar as réplicas, preservando-as de modo que os recursos do sistema sejam otimizados.

Neste capítulo apresentamos um modelo de replicação confiável de conteúdo para sistemas de arquivamento digital. No modelo proposto, repositórios de arquivamento são organizados como um sistema baseado em comunidades com o objetivo de garantir a confiabilidade a longo prazo do conteúdo digital. Tal modelo se caracteriza pela autonomia dos repositórios aos quais são associadas probabilidades independentes de falha. A asso-

ciação de probabilidades de falha aos repositórios permite que o conteúdo replicado seja organizado utilizando de maneira racional os recursos de armazenamento do sistema de arquivamento.

Este capítulo está organizado da seguinte maneira. A seção 4.1 apresenta uma visão geral dos componentes do modelo e as razões de sua escolha. A seção 4.2 apresenta a formalização teórica do modelo, bem como sua definição equivalente, útil para a construção dos algoritmos de replicação apresentados na seção 4.3. A avaliação empírica do modelo e dos algoritmos é mostrada através de simulação na seção 4.4.

4.1 Visão Geral

O modelo de replicação proposto utiliza a abordagem baseada em comunidades autônomas para organizar os repositórios de armazenamento. O modelo contém os seguintes componentes:

- Uma *rede de repositórios de arquivamento* responsável pelo armazenamento das réplicas. Repositórios irão cooperar uns com os outros para prover a preservação a longo prazo das informações. Cada repositório possui uma *capacidade de armazenamento* limitada.

- *Itens* ou *objetos digitais*, que são as unidades básicas dos dados preservados. Os objetos digitais geralmente são agrupados em coleções, que podem ser qualquer conjunto de objetos, por exemplo um grupo de fotos, uma coleção de revistas digitais, um conjunto de documentos de texto e etc.

- O modelo também possui *clientes*, que são usuários ou aplicações responsáveis por depositar e recuperar objetos digitais e operar o sistema.

Para cada repositório existe um valor associado que indica a probabilidade de seus dados não serem perdidos em um determinado período de tempo. Para determinar esta probabilidade, podemos considerar alguns parâmetros como a quantidade de *bugs*, as

vulnerabilidades no sistema, fatores humanos como a falta de responsabilidade do administrador do sistema, a taxa que máquinas são reparadas, dentre outros. Denominaremos esta probabilidade como a *confiabilidade do repositório*. Embora a confiabilidade do repositório possa mudar com o passar do tempo, assumimos neste primeiro trabalho que ela é constante.

A *confiabilidade do objeto* pode ser determinada a partir da confiabilidade dos repositórios que armazenam réplicas deste objeto. Podemos interpretar a confiabilidade de um objeto como a probabilidade dele não ser perdido em um intervalo particular de tempo [2]. O modelo também permite que o usuário escolha a *confiabilidade desejada* de um objeto a ser inserido na rede, criando as réplicas e definindo a quantidade delas conforme a confiabilidade de cada repositório. Através da confiabilidade desejada o usuário tem a capacidade de especificar quais objetos são mais importantes e quais merecem um tempo maior para a preservação. Assim, ao não considerar um grau de replicação fixo, o número de réplicas é reduzido de uma maneira geral e portanto também o custo total da rede de arquivamento digital.

A Figura 4.1 ilustra os componentes do modelo. Na figura o usuário está inserindo um objeto com confiabilidade desejada de 90% na rede de repositórios. Para atingir esta confiabilidade, uma estratégia seria criar réplicas nos repositórios 1, 2 e 3, totalizando uma confiabilidade de $1 - (0.6 \times 0.2 \times 0.7) = 91.6\%$ do objeto.

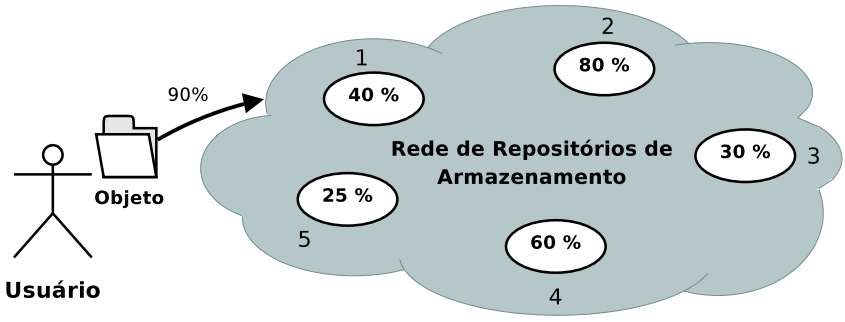

Figura 4.1: Principais componentes do modelo de replicação.

Neste capítulo estudamos três diferentes estratégias de como selecionar repositórios que satisfaçam a confiabilidade desejada de um objeto inserido e otimize a utilização

dos recursos de armazenamento da rede. Tais estratégias se deram de maneira intuitiva, porém foram formuladas com embasamentos teóricos para justificar sua utilização. Nas próximas seções apresentaremos a definição formal do modelo, as estratégias e a avaliação delas.

4.2 Definição Formal do Modelo

Formalmente, o modelo é composto de um conjunto N de $|N| = n$ itens (objetos digitais). Cada item i tem um *tamanho* s_i e um número $0 < r_i < 1$, dito a *confiabilidade desejada* deste item. Além disso, temos um conjunto $M = \{\ell_1, \ldots, \ell_m\}$ de $|M| = m$ repositórios, onde cada repositório ℓ_j tem associado um número $0 < p_j < 1$, dito a *confiabilidade do repositório* e uma *capacidade de armazenamento* de itens c_j. Definimos a *capacidade de armazenamento livre* de ℓ_j como $f_j = c_j - \sum_{k \in \ell_j} s_k$. A confiabilidade do subconjunto $S \subseteq M$ é definida como $1 - \prod_{\ell_j \in S}(1 - p_j)$, que denota a probabilidade de pelo menos um repositório em S não perder dados.

Definimos o problema da criação de réplicas da seguinte maneira. Itens vão chegando um a um *on-line*, ou seja, inicialmente a entrada é vazia e os itens vão chegando no decorrer do tempo. Após um item i chegar, devemos escolher um subconjunto de repositórios tal que a confiabilidade desse subconjunto seja pelo menos a confiabilidade desejada de i. Em outras palavras, devemos selecionar $S_i \subseteq M$ tal que $1 - \prod_{\ell_j \in S_i}(1 - p_j) \geq r_i$. Além disso, cada repositório em S_i deve suportar s_i. Ou seja, $\forall \ell_j \in S_i$, $s_i \leq f_j$ tem que valer. Em cada repositório de S_i colocamos então uma cópia do item i (*réplica*) para satisfazer r_i.

Neste modelo, temos o objetivo de selecionar subconjuntos de repositórios que minimizem o "desperdício" de suas confiabilidades ao satisfazer a confiabilidade desejada de um objeto inserido. Em outras palavras, para um item i queremos selecionar $S_i \subseteq M$ que forneça a confiabilidade que está mais próxima de r_i. Assim, esperamos que uma quantidade maior de itens possa ser inserida em M. Acreditamos que minimizar o número de réplicas geradas e balancear a criação destas réplicas entre os repositórios sejam boas heurísticas para alcançarmos tal objetivo.

A seguir, é feito algumas observações relacionadas à viabilidade do problema de criação

de réplicas no modelo proposto.

Observação 1. *Atribuir duas ou mais réplicas de um mesmo item em um único repositório não traz vantagens.*

Pela Observação 1, no pior caso criaremos m réplicas para cada item.

Observação 2. *Existem instâncias cujas soluções são inviáveis para o problema.*

Para ilustrar a Observação 2, suponha que todos repositórios tenham baixa confiabilidade, e.g. $1/(m+1)$. Pela Observação 1, não temos vantagem em criar mais que m réplicas. Assim, ao criarmos réplicas de um item em todos m repositórios, podemos garantir uma confiabilidade de $1 - \left(1 - \frac{1}{m+1}\right)^m \leq 1 - \frac{1}{e} \leq 2/3$. Ou seja, é impossível satisfazer a confiabilidade desejada de um item que a deseje maior que 2/3. Em face da Observação 2, gostaríamos de identificar condições que tornam uma instância do problema viável/inviável. Uma *caracterização* (uma condição do tipo se e somente se) seria o ideal neste caso.

Observação 3. *Devido à natureza* on-line *da entrada, não há como saber se uma instância do problema é viável.*

Pela Observação 3, só podemos afirmar que uma instância é inviável se o último item que chegou tornar a instância inviável, impossibilitando a caracterização das instâncias inviáveis. No entanto, sem "olharmos para o futuro" e olhando somente os itens que já chegaram, o problema é viável se e somente se $1 - \prod_{\ell_j \in S_i}(1 - p_j) \geq r_i$, $\forall i \in N$.

4.2.1 Definição Equivalente do Problema

A seguir, reescrevemos a restrição de confiabilidade desejada de forma a facilitar o manuseio do problema de replicação. Mais especificamente, apresentamos uma definição equivalente da restrição de confiabilidade desejada, substituindo o produtório por um somatório. Podemos reescrever a restrição de confiabilidade desejada da seguinte forma:

$$1 - \prod_{\ell_j \in S_i} (1 - p_j) \geq r_i \equiv \prod_{\ell_j \in S_i} (1 - p_j) \leq 1 - r_i$$
$$\equiv \prod_{\ell_j \in S_i} e^{\ln(1-p_j)} \leq e^{\ln(1-r_i)}$$
$$\equiv e^{\sum_{\ell_j \in S_i} \ln(1-p_j)} \leq e^{\ln(1-r_i)}.$$

Mas

$$e^{\sum_{\ell_j \in S_i} \ln(1-p_j)} \leq e^{\ln(1-r_i)} \iff \sum_{\ell_j \in S_i} \ln(1 - p_j) \leq \ln(1 - r_i).$$

Como $0 < p_j, r_i < 1$, então o valor da função logaritmo é negativa. Assim, para clareza de notação, vamos redefinir as variáveis do problema. Seja $a_j = -\ln(1 - p_j)$ e $b_i = -\ln(1 - r_i)$, então a restrição de confiabilidade desejada poder ser reescrita de maneira equivalente como

$$\sum_{\ell_j \in S_i} a_j \geq b_i$$

Ou seja, para um item i devemos selecionar $S_i \subseteq M$ tal que a soma de a_j ultrapasse o valor b_i. Essa definição, além de ser mais fácil de manusear, é equivalente à definição anterior.

4.3 Estratégias de Criação das Réplicas

A seguir, iremos propor três estratégias para solução do problema de replicação, que serão analisadas empiricamente na seção 4.4. Na subseção 4.3.1, a motivação principal da estratégia apresentada é a minimização das réplicas geradas, mas adaptações são feitas para gerar um bom balanceamento entre as réplicas (carga) nos repositórios. Na estratégia da subseção 4.3.2, temos como motivação principal o balanceamento da carga, justificada por resultados teóricos de *ball-and-bins*. Além disso, obtivemos um limitante teórico para o número de réplicas geradas. A estratégia da subseção 4.3.3 resolve o problema sem priorizar a minimização de réplicas ou o balanceamento, agindo como um "meio-termo"

entre as motivações das outras duas estratégias.

4.3.1 Gulosa sobre Subconjuntos

É possível acontecer situações nas quais os repositórios estão cheios no momento da criação de alguma réplica. Logicamente que explorar outros repositórios e não falhar naquele primeiro que esteja cheio, resulta em uma taxa de inserção maior de itens, e este é o nosso objetivo. Portanto, no restante do artigo e na avaliação que se segue, os algoritmos sempre exploram outros repositórios quando alguns não têm mais capacidade para armazenar uma determinada réplica. Ao tentar criar a réplica no repositório, é verificado se o espaço de armazenamento livre dele é suficiente para alocá-la e caso não seja outro repositório aleatório do subconjunto é explorado. Por razões de simplicidade e clareza não expomos esses detalhes nos algoritmos a seguir.

Vamos supor que estamos querendo minimizar o número total de réplicas, a princípio sem nos preocupar com o balanceamento de carga. Então, usando a definição equivalente da Seção 4.2.1, basta resolvermos o programa linear inteiro (PLI) a seguir

$$\begin{aligned} \min & \sum_{\ell_j \in M} x_j \\ \text{s.t.} & \sum_{\ell_j \in M} a_j x_j \geq b_i \\ & x_j \in \{0,1\} \qquad \forall \ell_j \in M \end{aligned}$$

O PLI descrito pode ser resolvido na otimalidade ordenando os valores a_j de maneira não-crescente e incluindo na solução do problema os valores nesta ordem até a soma dos valores selecionados alcançar b_i, como mostra o Algoritmo 1.

```
input: set M, desired reliability b_i of item i
begin
    sort M in non-increasing order according to the values a_j. S = ∅
    t = 1
    while Σ_{ℓ_j ∈ S} a_j < b_i do
        S = S ∪ {ℓ_t}
        t = t + 1
    return S
end
```

Algoritmo 1: NaïveGreedy

Esta estratégia gulosa, além de muito simples e eficiente, é suficiente para resolver o problema na otimalidade em relação ao número de réplicas pois em nenhum momento é mais vantajoso incluir outro valor fora da ordem. Contudo, note que esta estratégia acumula as réplicas nos repositórios com maiores confiabilidades e isto é ruim para o balanceamento de carga, já que as réplicas serão atribuídas, em sua maior parte, ao mesmo subconjunto de repositórios. Para resolver isso, executamos o Algoritmo 1 sob um subconjunto de repositórios escolhido aleatoriamente, como explicado a seguir.

Seja f um número inteiro. Para cada item i que chega, primeiramente selecionamos um conjunto $F \subseteq M$ de maneira aleatória, tal que $|F| = f$. Após isso, executamos o algoritmo guloso para selecionar o menor subconjunto $S \subseteq F$ que atinja a confiabilidade desejada de i.

```
input: set M, desired reliability b_i of item i
begin
    select f random sites from M, call this set F
    sort F in non-increasing order according to the values a_j
    S = ∅
    t = 1
    while Σ_{ℓ_j ∈ S} a_j < b_i do
        S = S ∪ {ℓ_t ∈ F}
        t = t + 1
    return S
end
```

Algoritmo 2: Greedy

No Algoritmo 2, note que se $f = |M|$, então recaímos no problema do acúmulo de réplicas nos repositórios com maiores confiabilidades. Por outro lado, se f for pequeno, não existem muitas opções de escolha sobre F ou não há repositórios suficientes para satisfazer a confiabilidade desejada de um item. O desafio é encontrar um valor f que não sobrecarregue os repositórios, mas que ao mesmo tempo criemos poucas réplicas. Na Seção 4.4, são avaliados resultados para vários valores de f.

4.3.2 Aleatorizada

A seguir, apresentamos a estratégia que utiliza aleatoriedade para a resolução do problema de criação de réplicas.

```
input: set M, desired reliability r_i of item i
begin
    S = ∅
    while reliability of S < r_i do
        choose ℓ_j ∈ M uniformly at random
        S = S ∪ {ℓ_j}
    return S
end
```

Algoritmo 3: Randomized

O Algoritmo 3 apresenta a estratégia Aleatorizada. Nesta estratégia, consideramos que as confiabilidades dos repositórios estão distribuídas uniformemente em um inter-

valo. Formalmente, estamos assumindo que os valores p_j são escolhidos de acordo com a *distribuição uniforme contínua* no intervalo $[a, b]$, onde $a > 0$ e $b < 1$. Sem perda de generalidade, a confiabilidade de um repositório pode ser interpretada como a probabilidade daquele repositório estar disponível. Desta forma interpretamos também a confiabilidade desejada de um item como a probabilidade dele estar em pelo menos um repositório disponível. Assim, consideramos também que o número esperado de repositórios disponíveis é pelo menos $8 \ln m$; note que isto não é uma suposição forte para o problema, visto que a fração esperada $\frac{8 \ln m}{m}$ de repositórios disponíveis tende a 0 quando m tende ao infinito. Por exemplo, para $m = 1000$, estamos supondo que a fração esperada de repositórios disponíveis seja $\frac{8 \ln 1000}{1000} \approx 0,05$, ou seja, estamos supondo que na média 5% das repositórios estejam disponíveis em um determinado instante.

Seja X_j a váriável aleatória que é igual a 1 se o repositório ℓ_j está disponível e 0 caso contrário. Seja $X = \sum_{j \in M} X_j$. Pelo fato de termos assumido que as confiabilidades dos repositórios são distribuídas de acordo com a distribuição uniforme contínua, então $E[X] = \frac{m(a+b)}{2}$. Como esse valor é esperado, pode acontecer de num determinado instante um número menor de repositórios estarem disponíveis. No entanto, o número de repositórios disponíveis não pode ser muito menor que a média, como mostra o Lema 4.3.1

Lema 4.3.1. *Seja X_j uma variável aleatória que é igual a 1 se o repositório ℓ_j está disponível, 0 caso contrário. Seja $X = \sum_{\ell_j \in M} X_j$ a v.a. do número total de repositórios disponíveis. Então $Pr(X < \frac{1}{4}E[X]) < \frac{1}{m^2}$.*

Demonstração. Note que X é uma somatória de v.a. de Poisson. Portanto, podemos aplicar o limitante de Chernoff, obtendo $Pr(X < (1 - \frac{3}{4})E[X]) \leq e^{-\frac{9E[X]}{32}} \leq m^{-2}$, onde a última desigualdade segue do fato que $E[X] \geq 8 \ln m$. □

O Lema 4.3.1 nos diz que, ao escolher um repositório aleatoriamente de maneira uniforme, com alta probabilidade (ou seja, probabilidade maior que $1 - \frac{1}{m^2}$), uma fração de $\frac{a+b}{8}$ dos repositórios estão disponíveis. Assim, ao escolhermos k repositórios aleatoriamente de maneira uniforme, a probabilidade de um ou mais repositórios estarem disponíveis é

pelo menos

$$1 - \left(1 - \frac{a+b}{8}\right)^k,$$

e queremos que isso seja maior que a confiabilidade desejada r_i, para um item i. Assim, ao escolher $k = \left\lceil \frac{8}{a+b} \ln\left(\frac{1}{1-r_i}\right) \right\rceil$ repositórios aleatoriamente de maneira uniforme para colocar as réplicas, a confiabilidade desejada do item i é satisfeita com alta probabilidade, como afirma o Teorema 4.3.2.

Teorema 4.3.2. *Basta um item i colocar $k = \left\lceil \frac{8}{a+b} \ln\left(\frac{1}{1-r_i}\right) \right\rceil$ réplicas em k repositórios escolhidos aleatoriamente de maneira uniforme para que, com alta probabilidade, a confiabilidade desejada de i seja satisfeita.*

Como exemplo de aplicação do Teorema 4.3.2, suponha que as confiabilidades dos repositórios estão distribuídas uniformemente entre 50% e algo próximo de 100%. Assim, um item que deseja uma confiabilidade de 95%, precisa escolher $\left\lceil \frac{8}{0,5+1} \ln\left(\frac{1}{1-0,95}\right) \right\rceil = 16$ repositórios aleatoriamente de maneira uniforme para colocar suas réplicas.

Com relação ao balanceamento de carga, podemos notar que o Algoritmo 3 faz com que uma réplica sempre escolha um repositório aleatoriamente de maneira uniforme. Ou seja, o Algoritmo 3 simula o processo de *balls and bins*, bastante estudado na área de algoritmos probabilísticos. Assim, os resultados já existentes se encaixam de maneira direta no nosso problema. Os resultados dos Teoremas 4.3.3 e 4.3.4 podem ser facilmente obtido a partir dos resultados sobre *balls and bins* [5], as demonstrações serão omitidas por questões de espaço.

Teorema 4.3.3. *Se n bolas são jogadas aleatoriamente de maneira independente e uniforme em m recipientes, então a carga máxima é limitada por $2e\frac{n}{m} + 2\log m$ com alta probabilidade.*

Teorema 4.3.4. *Seja $n \geq m \log m$. Se n bolas são jogadas aleatoriamente de maneira independente e uniforme em m recipientes então a carga máxima é limitada por $\frac{n}{m} + \sqrt{8\frac{m}{n}\log m}$ com alta probabilidade.*

Respectivamente, os Teoremas 4.3.3 e 4.3.4 dizem respeito a quando temos um número pequeno e grande de bolas (no nosso problema, são os itens). No entanto, em situações

práticas, é provável que $n \geq m \log m$, e assim o Teorema 4.3.4 nos diz que, com alta probabilidade, o repositório mais cheio terá um número constante de itens a mais que o balanceamento ótimo.

4.3.3 Subconjunto Ideal

A idéia desta estratégia é selecionar um subconjunto aleatório $S \subseteq M$ de repositórios, e então selecionar o subconjunto $F \subseteq S$ que forneça a confiabilidade que está mais próxima da confiabilidade desejada do item. Ou seja, queremos selecionar $F \subseteq S \subseteq M$ que minimize $1 - \prod_{\ell_j \in F}(1 - p_j) - r_i$. Usando a definição equivalente do problema, o que queremos é resolver o seguinte PLI

$$\begin{aligned} \min & \sum_{\ell_j \in S} a_j x_j - b_i \\ \text{s.t.} & \sum_{\ell_j \in S} a_j x_j \geq b_i \\ & x_j \in \{0, 1\} \qquad \forall \ell_j \in S \end{aligned}$$

Ao resolvermos o PLI, se o valor ótimo for igual a 0 então existe um subconjunto de valores que somados dão exatamente b_i; se o valor ótimo for maior que 0, então não existe tal subconjunto. Desta forma poderíamos resolver o problema de decisão SUBSET-SUM que é NP-COMPLETO [52]. Portanto, o PLI é NP-DIFÍCIL e, assumindo P $\neq$ NP, não existe algoritmo polinomial que resolva este problema. No entanto, existe um algoritmo de *programação dinâmica* que resolve esse problema em tempo pseudo-polinomial e na prática é satisfatório[1] para a grande maioria das instâncias [52]. Os detalhes do algoritmo são discutidos no Apêndice A.

A solução do PLI acima não necessariamente irá nos fornecer uma solução boa para minimizar o número de réplicas geradas. Apesar disso, a estratégia do Subconjunto Ideal é motivada pelo fato que em situações práticas é esperado não criar muitas réplicas e ao

[1] No nosso simulador a versão do algoritmo por força bruta (computando todos os possíveis subconjuntos) demorou 13.16 **minutos** para ser executado contra 1.1 **segundos** na versão em programação dinâmica para um $|S| = 22$. Este teste foi realizado num processador Intel Core 2 Duo, 2.00 GHz.

mesmo tempo selecionar subconjuntos diferentes para cada item, para que a distribuição das réplicas nos repositórios seja feita de modo a balancear a carga.

4.4 Avaliação e Simulação de Experimentos

Através de experimentos avaliamos as três estratégias de criação de réplicas utilizados no nosso modelo de replicação. Foi utilizado um simulador próprio que implementa os algoritmos e insere uma massa de dados. Todos os experimentos foram executados 10 vezes e a média foi coletada. Em todos os experimentos consideramos um conjunto de 100 repositórios.

4.4.1 Número de Réplicas Inseridas

Para a primeira série de experimentos, o simulador foi executado para repositórios com a confiabilidade média variando entre 50%, 60%, 70% e 80%, com o desvio padrão de 6% em cada um deles. Realizamos a inserção de 1000 itens aleatórios, todos com a confiabilidade desejada de 99%. Neste experimento desconsideramos o tamanho dos itens e a capacidade dos repositórios.

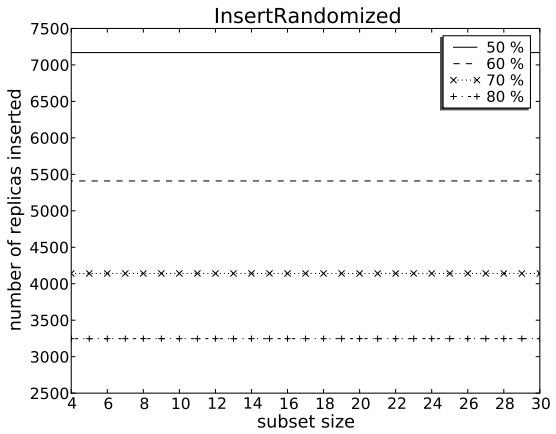

Figura 4.2: Estratégia Aleatorizada: número de réplicas inseridas em função do tamanho do subconjunto.

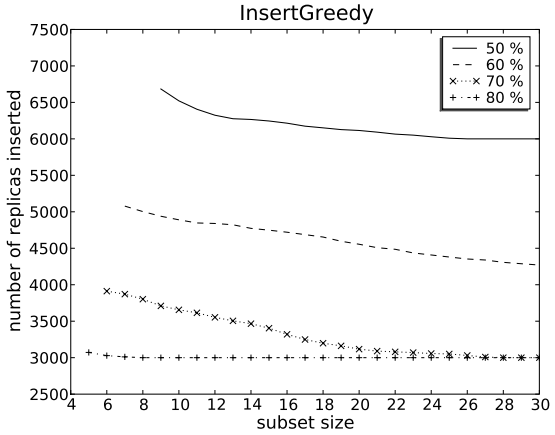

Figura 4.3: Estratégia Gulosa: número de réplicas inseridas em função do tamanho do subconjunto.

As Figuras 4.2, 4.3 e 4.4 ilustram o primeiro experimento, ao qual as réplicas são geradas em função do tamanho do subconjunto nas três estratégias. Obviamente que para a estratégia Aleatorizada não faz sentido variar o tamanho do subconjunto pois

esta estratégia não depende de tal parâmetro, mas o gráfico é ilustrado como forma de comparação com as outras estratégias. Vale observar também que nas outras duas estratégias, dependendo da confiabilidade média dos repositórios, a linha inicia somente após um determinado tamanho do subconjunto de repositórios selecionados. O motivo é que para um tamanho muito pequeno do subconjunto, a confiabilidade desejada não é satisfeita na inserção de todos os itens. Por exemplo, em todos os gráficos o eixo x inicia no valor 4, pois para valores menores que isso a estratégia do Subconjunto Ideal e Gulosa não são capazes de inserir todos os 1000 itens. Portanto, os resultados obtidos nestes gráficos somente plotam linhas quando todos os itens são inseridos com sucesso.

Conforme o tamanho do subconjunto de repositórios cresce, a quantidade de réplicas suficientes para alcançar a confiabilidade desejada diminui. Esta diminuição é mais acentuada na estratégia Gulosa (Figura 4.3), a qual tem o objetivo explícito de diminuir as réplicas, sempre criando-as em repositórios com maior confiabilidade. A alternativa do Subconjunto Ideal (Figura 4.4) também apresenta um declive, porém menor que na estratégia Gulosa e maior que na Aleatorizada (Figura 4.2).

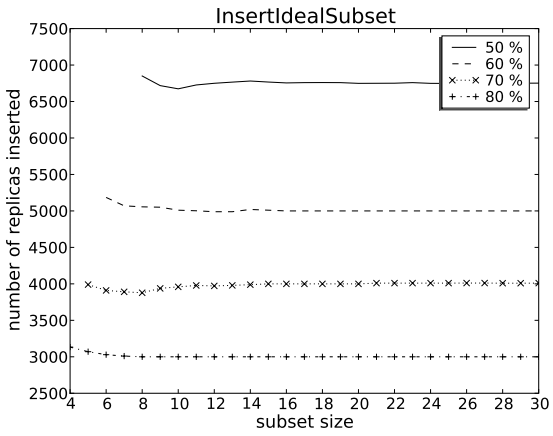

Figura 4.4: Estratégia do Subconjunto Ideal: número de réplicas inseridas em função do tamanho do subconjunto.

4.4.2 Balanceamento da Carga

O desvio padrão da carga dos repositórios (em termos de armazenamento de réplicas) é uma medida suficiente para observar o balanceamento após a inserção das réplicas. Dessa maneira, a Figura 4.5 ilustra o segundo experimento que mostra o desvio padrão da carga dos repositórios variando em função do tamanho dos subconjuntos. Foi utilizado uma confiabilidade média de 60% para os repositórios com 6% de desvio padrão (o mesmo

experimento foi repetido para repositórios com outras confiabilidades e o resultado obtido foi proporcionalmente o mesmo).

Da mesma maneira que no experimento anterior, a alternativa Aleatorizada não sofre influência sobre o tamanho do subconjunto de repositórios selecionados. Para as outras estratégias, podemos observar que quanto maior o tamanho do subconjunto, maior o desvio padrão da carga. A estratégia Gulosa é a que tem o pior balanceamento.

Alterar a variância da confiabilidade dos repositórios acarreta um resultado não-intuitivo a primeira vista para as estratégias Gulosa e do Subconjunto Ideal. Tais estratégias dependem de um subconjunto de tamanho mínimo para garantir a inserção de todos itens. Aumentando o desvio padrão da confiabilidade dos repositórios, aumenta a probabilidade de selecionar um subconjunto "ruim", ou seja, que não satisfaça a confiabilidade desejada de algum item. Portanto, aumentar o desvio padrão impacta em também aumentar o tamanho mínimo necessário do subconjunto de repositórios para realizar a inserção de *todos* os itens, como ilustram as Tabelas 4.2 e 4.1. O *makespan* – carga da máquina mais carregada – também é analisado nas tabelas.

Na estratégia Gulosa (Tabela 4.1), para um conjunto de repositórios com desvio padrão de 6% nas suas confiabiliaddes, o menor tamanho dos subconjuntos que garante a inserção de 1000 itens é 9; o *makespan* é 109. Em um conjunto com desvio padrão de 28% (o máximo i.e., variando a confiabilidade dos repositórios de 1% até 99%), o menor tamanho de subconjuntos que garante a inserção de 1000 itens é 11; o *makespan* é 125.

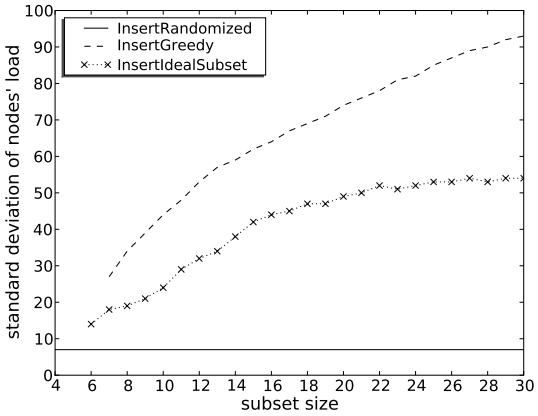

Figura 4.5: Desvio padrão da carga dos repositórios em função do tamanho dos subconjuntos.

Na estratégia do Subconjunto Ideal (Tabela 4.2), para um conjunto com desvio padrão na confiabilidade dos repositórios de 6%, o menor tamanho dos subconjuntos

que garante a inserção de 1000 itens é 8; o *makespan* é 96. Enquanto isso, em um conjunto com desvio padrão na confiabilidade dos repositórios de 28% o menor tamanho dos subconjuntos que garante a inserção de 1000 itens é 11; o *makespan* é 114.

σ	Subset Size	Makespan
6 %	9	109
28 %	11	125

Tabela 4.1: Variando o desvio padrão da confiabilidade dos repositórios na estratégia Gulosa.

σ	Subset Size	Makespan
6 %	8	96
28 %	11	114

Tabela 4.2: Variando o desvio padrão da confiabilidade dos repositórios no Subconjunto Ideal.

4.4.3 Makespan

As Figuras 4.6, 4.7 e 4.8 ilustram o terceiro experimento no qual a carga da máquina mais carregada, ou o *makespan*, é plotado em função do tamanho do subconjunto de repositórios (repare que o eixo x mostra mais valores que nas figuras anteriores). Conforme o tamanho do subconjunto de repositórios selecionados cresce, o *makespan* também cresce nas estratégias Gulosa e do Subconjunto Ideal, porém nesta última, sendo sempre menor o crescimento. Este comportamento de crescimento é justificado pelo fato de alguns repositórios serem sempre os mais requisitados. Na estratégia Gulosa os repositórios mais requisitados serão obviamente os com maior confiabilidade. Os repositórios com menor confiabilidade serão os mais requisitados na estratégia do Subconjunto Ideal, pois repositórios com esta característica são sempre utilizados para evitar o "desperdício" de confiabilidade.

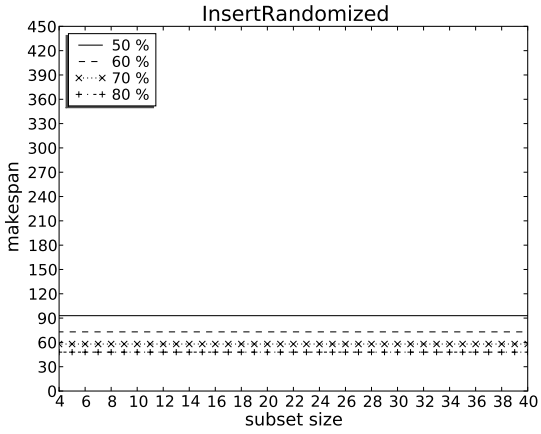

Figura 4.6: Makespan em função do tamanho do subconjunto.

Em um conjunto de repositórios com capacidades de armazenamento limitadas, alguns repositórios podem não ter espaço suficiente para acomodar réplicas na inserção de um

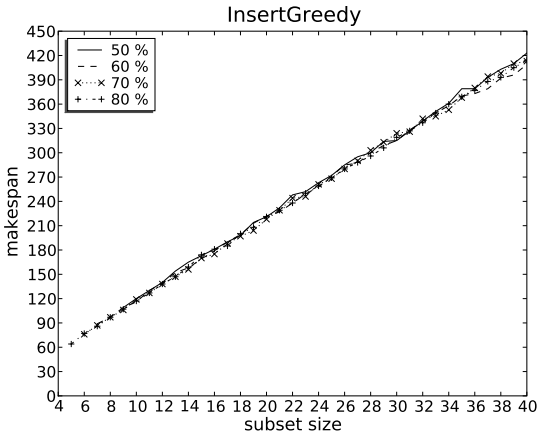

Figura 4.7: Makespan em função do tamanho do subconjunto.

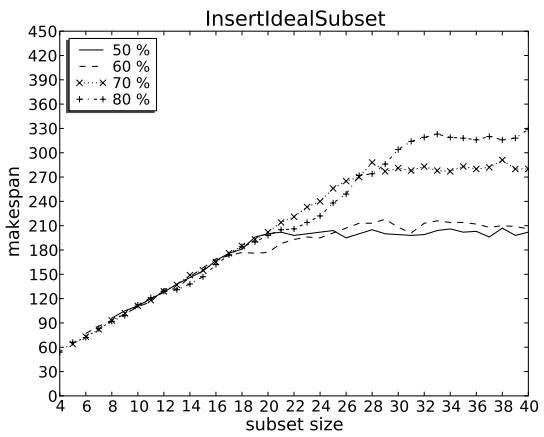

Figura 4.8: Makespan em função do tamanho do subconjunto.

item. O próximo experimento discute repositórios com tais características.

4.4.4 Taxa de Sucesso na Inserção de Itens

Para conjuntos de repositórios com capacidades de armazenamento homogêneas, o *makespan* é um importante índice pois avalia a taxa de sucesso na inserção de itens. A estratégia que apresentar o maior *makespan* implicará na menor taxa de sucesso para a inserção de itens. Entretanto, em ambientes reais é comum encontrarmos redes nas quais a capacidade de armazenamento dos repositórios diferem em várias ordens de magnitude. Neste tipo de rede, o *makespan* não necessariamente vai indicar o mesmo comportamento que nos repositórios com capacidade igual de armazenamento.

O principal objetivo deste experimento é medir a taxa de sucesso na inserção de itens em repositórios heterogêneos em termos capacidade. Inserimos itens até que algum deles falhe na criação das réplicas por falta de capacidade de armazenamento em um repositório. Os itens têm tamanho de 35 MB e confiabilidade desejada de 99%. A capacidade de armazenamento dos 100 repositórios varia entre 100 MB e 100.000 MB, e confiabilidade média deles é de 67% (desvio padrão de 17%).

Dependendo da estratégia utilizada, alterar o tamanho do subconjunto de repositórios selecionados causa diferentes taxas de inserção: quanto maior for o subconjunto selecionado de repositórios, maior o número possível de soluções e conseqüentemente maior a

taxa de inserção de itens. A Tabela 4.3 mostra a inserção de itens em função do tamanho do subconjunto de repositórios para as três estratégias de criação de réplicas. A estratégia do Subconjunto Ideal tem os melhores resultados.

	10	20	40	100
Aleatorizada	18569	30228	32464	34138
Gulosa	17544	28182	30891	33899
Subconjunto Ideal	20535	31094	35399	39107

Tabela 4.3: Inserção de itens variando o tamanho do subconjunto.

4.4.5 Discussão dos Resultados Obtidos

A estratégia Aleatorizada distribui uniformemente as réplicas nos repositórios do subconjunto selecionado e o ponto fraco dela é ao "desperdiçar" confiabilidade: no momento em que falta somente uma réplica para atingir a confiabilidade desejada, a seleção desta réplica vai acontecer de maneira aleatória, podendo selecionar um repositório com confiabilidade que ultrapassa demais o valor necessário.

A estratégia Gulosa sempre vai "encher" os repositórios com maior confiabilidade antes. O pior caso para esta estratégia é quando existir repositórios com grandes confiabilidades que tenham o espaço de armazenamento pequeno, onde o algoritmo sempre vai tentar colocar réplicas nestes espaços.

A estratégia do Subconjunto Ideal é capaz de sempre selecionar o subconjunto mais justo, desperdiçando pouca confiabilidade. Independente do grau de heterogeneidade do subconjunto de repositórios, esta estratégia deverá sempre seguir com os melhores resultados.

Dentre as três estratégias, a Gulosa sempre vai gerar o menor número de réplicas. Por outro lado, ela sempre apresenta o *makespan* mais alto. A estratégia Aleatorizada é a que apresenta o melhor balanceamento das réplicas, porém apresenta o pior número de réplicas geradas. O meio termo entre diminuir o número de réplicas e espalhá-las é obtido com a estratégia do Subconjunto Ideal. Além disso, em um conjunto de repositórios de capacidades de armazenamento heterogêneas, a estratégia do Subconjunto Ideal demons-

trou preencher melhor os espaços dos repositórios ao inserir uma quantidade maior de itens.

Capítulo 5

Um Sistema Peer-to-Peer para Arquivamento Digital

"The future is here. It's just not widely distributed yet." –
William Gibson, romancista de ficção científica *(1948 –)*

O modelo proposto no capítulo 4 está definido de maneira genérica para ser aplicado sobre qualquer mecanismo distribuído de organizar repositórios de armazenamento. Em particular, as redes P2P estruturadas apareceram como canditadas naturais para organizar esses repositórios, visto que são altamente escaláveis para a distribuição e recuperação de dados. Entretanto, é indispensável a utilização de estratégias de replicação que garantam a confiabilidade da informação da forma que o ambiente de arquivamento digital exige. Além disso, existe uma dificuldade inerente das redes P2P estruturadas que é a seleção exata dos nodos (repositórios) que armazenarão as réplicas. Selecionar um subconjunto específico de nodos não é trivial utilizando a maneira determinística de rotear informações das DHTs.

Neste capítulo apresentamos um sistema de arquivamento digital implementado sobre a arquitetura das redes P2P estruturadas. O sistema é baseado no modelo e nos algoritmos de replicação propostos no capítulo 4. A troca de mensagens entre os repositórios do sistema é realizada através das DHTs. Além disso, o sistema se apóia na estratégia de múltiplas funções *hash* para permitir a criação de réplicas e a seleção dos repositórios que

as armazenarão. Tal estratégia torna o sistema independente do protocolo DHT utilizado.

O sistema proposto está essencialmente preocupado com o armazenamento organizado e confiável da informação. O acesso e a auditoria do conteúdo previamente inserido no sistema também é tratado, porém com menos ênfase. Técnicas mais robustas para isso são encorajadas a serem aplicadas, como por exemplo o *Open Archives Initiative Protocol for Metadata Harvesting* [53] para o acesso e o LOCKSS para a auditoria. É importante salientar também que está fora do objetivo deste trabalho tratar de questões relacionadas a obsolescência de *software* e formatos. Porém, para prover um sistema de preservação completo, certamente estratégias que tratem estas ameaças devem ser exploradas.

O restante desse capítulo está organizado da seguinte maneira. A seção 5.1 apresenta a descrição do sistema de arquivamento sobre P2P, detalhando a arquitetura e a razão das estratégias utilizadas para sua concepção. Na seção 5.2 são apresentados os algoritmos de operação do sistema. Cabe a seção 5.3 mostrar a avaliação do sistema, bem como os experimentos executados em um ambiente real.

5.1 Arquitetura

A seguir apresentaremos e defenderemos a arquitetura utilizada para a implementação do sistema P2P de arquivamento digital. Em especial, mostraremos como funcionam as partes deste sistema e seu funcionamento.

O coordenamento geral de todas as partes do sistema tem como objetivo garantir a confiabilidade das coleções digitais por um longo prazo e para isso realiza as seguintes tarefas: (1) recebe requisitos de clientes para a inserção ou recuperação de objetos digitais no sistema. Na inserção de cada objeto, o cliente especifica uma confiabilidade que deseja que o objeto seja preservado; (2) dependendo da estratégia de replicação empregada e da confiabilidade de cada sítio, calcula o número de réplicas necessárias do objeto, e; (3) faz a auditoria de cada objeto digital comparando com as várias réplicas da rede, corrigindo-a se necessário e garantindo a preservação a longo prazo.

5.1.1 P2P Estruturada e DHT

O sistema faz o roteamento das mensagens na rede através das P2P estruturadas, utilizando as DHTs. Acreditamos que a arquitetura P2P não-estruturada não satisfaça bem os requisitos de um sistema para arquivamento digital por algumas razões. Primeiramente, no ambiente de arquivamento a longo prazo a frequencia de busca por arquivos é normalmente uniforme, ou seja, todos os objetos são igualmente populares [40]. Redes P2P não-estruturadas normalmente usam algoritmos de força bruta, através de inundação, para realizar a busca e portanto são mais apropriados para conteúdos populares. Além disso, algoritmos de busca das redes não-estruturadas geralmente não conseguem localizar itens raros, o que é inadmissível para o contexto de preservação e arquivamento digital. E por fim, a escalabilidade das não-estruturadas deixam a desejar quando comparada as estruturadas.

A principal crítica em relação a P2P estruturadas é com populações extremamente transientes. Manter a estrutura das tabelas de roteamento é relativamente custoso quando acontece *churn*. Entretanto, a transiência das máquinas em organizações que têm a intenção de preservar documentos digitais não acontece tão frequentemente quando comparada com máquinas utilizadas em aplicações tradicionais na arquitetura não-estruturada [13]. Portanto, os ajustes necessários na topologia da redes estruturadas não devem sobrecarregar o sistema de arquivamento no caso de *churn*.

5.1.2 Replicação

No ambiente de arquivamento digital é importante salientar que não estamos interessados em atualizações das réplicas, uma vez que estamos falando de objetos *somente de leitura* e *estáticos* [46]. Feita uma réplica, ela se torna imutável no sistema. Além disso, para armazenamento a longo prazo a *integridade* e a *autenticidade* das réplicas são requisitos básicos que devem ser tratados. Utilizaremos a estratégia de auditoria (seção 5.2) para garantir estes requisitos.

Foi escolhida a estratégia de múltiplas funções *hash* para replicar os dados no sistema P2P. Utilizamos uma maneira muito simples para criar as *hashs*. O número da

réplica a ser criada i é anexado no final da chave do objeto e então passado como argumento para uma função h hipotética. Suponha que o nome do objeto é *foo*, então $h(foo1), h(foo2), \ldots, h(foor)$ nos daria r *hashs* para este objeto. Esta maneira simula muito bem a criação de múltiplas funções *hash* e recuperação das mesmas. A seguir, enumeramos os principais pontos que nos motivaram a empregar a estratégia de múltiplas funções *hash*:

- **a independência por um protocolo DHT**. A estratégia de múltiplas *hash* permite que possamos utilizar qualquer protocolo DHT para rotear as mensagens na rede. Outras estrátégias de replicação, como replicação por vizinhança ou por caminho, não se beneficiam de tal característica.

- **número independente de réplicas para cada objeto**. Como foi dito no capítulo 4, uma das principais características do nosso modelo de replicação é a independência pelo número de réplicas que cada objeto digital possui. A estratégia de replicação P2P simétrica não tem esta flexibilidade e não poderia ser utilizada como mecanismo de replicação para o nosso modelo. Podemos citar o BRICKS como um exemplo de um sistema de armazenamento confiável de dados ao qual define o grau de replicação fixo dos itens.

- **recuperação fácil do objeto**. A facilidade de recuperar uma determinada réplica sem a dependência de recuperar outra previamente é uma grande motivação da estratégia com múltiplas *hash*. A independência na hora de recuperar uma réplica facilita o algoritmo de busca, como veremos na seção 5.2. Na estratégia do *hash* correlacionado, todas as chaves de um determinado objeto são correlacionadas a primeira chave, não possibilitando tal característica.

5.1.3 Seleção dos Repositórios

Nas redes P2P estruturadas a topologia da rede é bem definida e o conteúdo é colocado em locais determinados – a maneira que a DHT indexa uma chave é sempre determinística. Assim, escolher exatamente o nodo que um conteúdo deve ser guardado não é uma

característica deste tipo de rede. Entretanto se estamos interessados em salvar o conteúdo em repositórios com confiabilidades determinadas, então devemos prover um mecanismo que simule este processo de seleção.

Ao aplicar a chave de um objeto em r funções *hash*, temos um subconjunto com r parâmetros de confiabilidade dentro dele. Desta maneira, obtemos um subconjunto de confiabilidades ao qual podemos selecionar as que melhores se adequarão nas estratégias de criação das réplicas. É importante notar que selecionar *todos* os nodos da rede não é uma boa abordagem pois, em eventuais algoritmos executados no sistema, uma instância teria o tamanho da rede e neste caso a complexidade computacional seria inviável. Por outro lado, executar estratégias a partir de um subconjunto de tamanho constante é viável mesmo em face da escalabilidade da rede, como veremos nos Algoritmos 4, 5 e 6.

Esta maneira de selecionar os repositórios implica em não colocar nenhuma informação centralizada sobre o local em que as réplicas estão armazenadas. Uma alternativa seria existir um super-nodo que tivesse um "diretório" ao qual pudesse ser feitas consultas sobre a informação exata do lugar das réplicas de determinado objeto. Porém, este super-nodo seria um ponto de contenção e uma reponsabilidade muito grande seria perdida caso ele falhasse. O nosso sistema evita super-nodos, adotando uma abordagem totalmente distribuída onde nenhuma informação é centralizada. Assim, para fazer a recuperação de uma réplica por exemplo, todos os repositórios do subconjunto determinado pelas múltiplas funções *hash* são considerados.

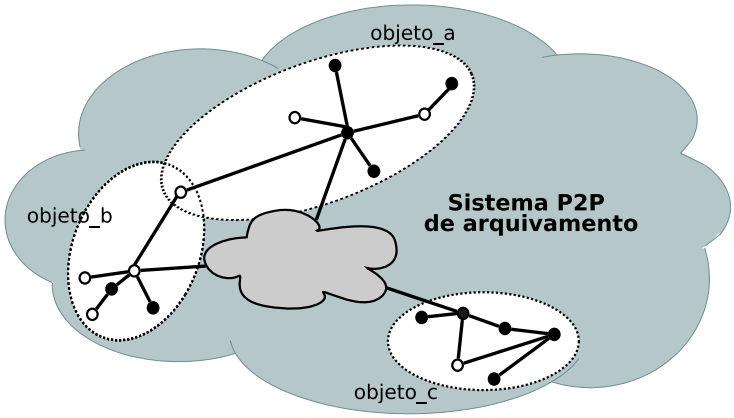

Figura 5.1: Subconjunto de repositórios determinado pelos objetos digitais.

A Figura 5.1 ilustra a seleção de repositórios realizada por três objetos diferentes. Na

figura, os subconjuntos de repositórios determinados pelas chaves `objeto_a`, `objeto_b` e `objeto_c` são representados pelos traços pontilhado. Note que não necessariamente estes subconjuntos precisam ser disjuntos. Os círculos pretos são repositórios que contêm réplicas enquanto os brancos não as contêm.

5.2 Algoritmos de Operação do Sistema

Uma grande característica do sistema proposto é que nenhuma modificação no protocolo DHT é necessária para permitir sua utilização. Simplesmente é realizado um "invólucro" nas operações básicas da DHT, `put` e `get`, chamando-as respectivamente de algoritmos `insert` e `retrieve`. É importante notar que ambos algoritmos são processados localmente em cada nodo. Por exemplo, um usuário que desejar fazer a inserção ou recuperacão de um objeto contacta qualquer nodo da rede, que através da DHT inicia todo o processo de roteamento da mensagem.

Para inserir um objeto, o usuário do sistema de arquivamento executa a rotina `insert(name, value, reliability)`, como visto no Algoritmo 4. Ao inserir um item na rede, o usuário previamente escolhe a confiabilidade desejada deste item (`reliability`). Dependendo da estratégia de inserção utilizada, é feita a busca das confiabilidades de cada repositório utilizado (que poderia ser armazenada na própria DHT por exemplo). Várias funções *hash* irão mapear o item em diferentes nodos da rede utilizando `put(key, value)`, satisfazendo a confiabilidade desejada. O algoritmo de inserção cuida para não atribuir duas ou mais réplicas de um mesmo objeto em único nodo (Observação 1, seção 4.2).

```
input: name, value, desired reliability r_i
begin
    S = get subset of repositories by the insert strategy; use r_i
    foreach s in S do
        j ← get hash function number of s
        put(h_j(name), value)
end
```

Algoritmo 4: insert (name, value, reliability)

Para recuperar o objeto, o usuário executa a função `retrieve(name)`, onde `name` é o nome do objeto requerido, visto no Algoritmo 5. A idéia do algoritmo é escolher um número aleatório de maneira uniforme entre 1 e r, onde r é o tamanho do subconjunto de repositórios selecionados. Se o número sorteado corresponde a uma função *hash* existente (réplica), o algoritmo termina com sucesso e o conteúdo do objeto é retornado para o usuário. Outras dois fatores podem acontecer para que a DHT não retorne o objeto: quando o nodo está ausente ou quando o valor sorteado é uma função que aponta para um nodo que não contém a réplica do objeto. Para ambos os casos, o algoritmo sorteia outro número. Um contador (`counter`) existe no algoritmo para garantir o término caso nenhuma réplica exista após percorrer todos os possíveis valores de r.

```
input: name
begin
    counter ← r
    while counter ≠ 0 do
        Let i chosen uniformly at random in {1, ..., r}
        value ← get(h_i(name))
        if value is not null then
            return value
        counter ← counter − 1
    return −1 /* not found                                    */
end
```

Algoritmo 5: retrieve (name)

Eventuais problemas podem acontecer nos objetos digitais devido a ataques maliciosos e modificações inadvertidas comprometendo a confiabilidade dos mesmos. Usuários de um sistema de arquivamento digital sob nenhuma circunstância devem ler objetos digitais sujeitos a modificações não autorizadas. Para solucionar isto, estratégias de auditoria devem ser concebidas. O Algoritmo 6 mostra um esboço de um processo de auditoria.

Periodicamente `audit (name)` é chamada para algum objeto `name`, sorteado aleatoriamente. Na primeira fase do algoritmo, uma quantia de réplicas do objeto (*num_rep*) é selecionada e recuperada através das múltiplas funções *hash* do objeto. Tal quantia deve ser razoável o suficiente para que um objeto não autêntico não interfira no processo de comparação com réplicas autênticas, na fase 2. Ainda na fase 1, uma *hash* segura c – digamos o CRC (*Cyclic Redundancy Check*) – é extraída de cada réplica desse grupo e

```
input: name
begin
    /* PHASE 1                                                    */
    R = {∅}
    Let num_rep < m
    while num_rep ≠ 0 do
        Let i chosen uniformly at random in {1, ..., m}
        value ← get(h_i(name))
        if value is not null then
            c ← checksum(value)
            R ← R ∪ {(i, c)}
        num_rep ← num_rep − 1
    /* PHASE 2                                                    */
    C = {c | (i, c) ∈ R}
    Let c_max the majority element in C
    I = {i | (i, c_max) ∈ R}
    Let j chosen uniformly at random in I
    value_ok ← get(h_j(name))
    foreach (i, c) ∈ R do
        if c ≠ c_max then
            put(h_i(name), value_ok)
end
```

Algoritmo 6: audit (name)

guardada em R, junto com o identificador i da réplica. Na segunda fase, o CRC de cada réplica do conjunto selecionado é comparado com todos os outros e se diferir da maioria o objeto digital de tal réplica é reparado, através da inserção de um objeto autêntico e íntegro.

A cooperação entre os nodos fornece a confiança necessária para garantir que o conteúdo disponível para os leitores estará íntegro e autêntico quando for requisitado. Além disso, torna-se evidente que quanto mais organizações preservarem o mesmo objeto, maior será a garantia de integridade e disponibilidade do conteúdo digital.

5.3 Resultados Experimentais

O sistema P2P de arquivamento foi implementado e avaliado através de experimentos executados em um ambiente real. A implementação foi feita utilizado o ambiente de construção de redes *Overlay Weaver* [54]. O *Overlay Weaver* proporciona grande flexibilidade

na escolha dos protocolos DHT e outros tipos de serviço alto-nível implementados como redes de sobreposição. Em particular, este ambiente de construção suporta o Chord, Pastry, Tapesty e Kademlia. Em nossos experimentos utilizamos o Chord. É importante notar que o *Overlay Weaver* já apresenta um mecanismo próprio de replicação ao qual foi desligado para a avaliação dos nossos experimentos.

Os experimentos foram conduzidos em uma rede de 12 nodos, quantidade suficiente para validar a capacidade de arquivamento a longo prazo das informações e também o mecanismo de criação das réplicas. Para a submissão dos comandos até os nodos e coleta dos resultados foi utilizado *scripts* executados através de *SSH*.

5.3.1 Avaliação

Consideramos a confiabilidade de cada nodo como a probabilidade dele não perder nenhuma informação no período de 1 ano. Assim, podemos simular qual será o estado da rede em relação a preservação das informações com o passar dos anos. Por exemplo, se a confiabilidade de um nodo é 70%, então ele tem 70% de chances de iniciar o segundo ano com todas as suas informações intactas. Da mesma maneira, este mesmo nodo tem 70 % de probabilidade de iniciar o terceiro ano, e também os outros próximos, sem nenhuma perda de informações. Note que o passar dos anos não implica em uma probabilidade maior de perda, pois tais probabilidades são independentes. Nos experimentos não foi avaliado a mudança de confiabilidade dos nodos ao passar dos anos.

O experimento é iniciado com uma determinada rede de nodos na qual são inseridos diversos objetos. Objetos são inseridos com diferentes confiabilidades. Para a replicação dos objetos é utilizado a estratégia do Subconjunto Ideal – estratégia que teve o melhor resultado nas simulações do capítulo 4.

No primeiro ano todos os nodos estão disponíveis (*online*) e nos próximos anos forçamos a desconexão de alguns nodos. Uma vez que o nodo é desconectado da rede (*offline*), ele perde todas as informações que tinha. Naturalmente, nodos que têm confiabilidade baixa, são os mais prováveis a serem desconectados. Ao voltar à rede, o nodo é considerado como um novo. Entretanto, nos experimentos não tratamos de novos nodos entrando na

rede. No final, espera-se que objetos com maior confiabilidade desejada sejam preservados por mais tempo.

5.3.2 Experimentos

Do total dos 12 nodos, 2 deles continham 30% de confiabilidade de não falhar, 2 continham 50%, 2 continham 70%, 3 continham 80% e os 3 restantes continham 90% de confiabilidade. Foi utilizada a estratégia do Subconjunto Ideal com subconjuntos de tamanho máximo 6. A Tabela 5.1 sumariza os resultado do experimento.

	1	5	10	15	20	25	30
mumm:39998 (30%)		x					
mumm:39997 (30%)	x						
talisker:39998 (50%)					x		
talisker:39997 (50%)	x						
cohiba:39998 (70%)		x					
cohiba:39997 (70%)			x				
dalmore:39998 (80%)			x				
dalmore:39997 (80%)				x			
dalmore:39996 (80%)						x	
priorat:39998 (90%)	x						
priorat:39997 (90%)			x				
priorat:39996 (90%)							x

Tabela 5.1: Nodos falhos em função dos anos no sistema de arquivamento.

Inicialmente, no primeiro ano, foram inseridos um total de 25 objetos, separados em 5 grupos, onde cada grupo continha objetos com confiabilidade desejada respectivamente de 30%, 50%, 70%, 90% e 99%.

Após o primeiro ano de existência do sistema, 1 nodo de confiabilidade igual a 30%, 1 nodo de 50% e outro de 90% falharam. Mesmo com estas falhas, todos os 25 objetos foram capazes de serem recuperados.

No final do quinto ano do sistema, mais dois nodos (30% e 70% de confiabilidade) se desconectaram da rede. Neste quinto ano não foi possível recuperar na rede 3 objetos de 30% de confiabilidade desejada, 4 objetos de 50%, 1 de 70% e 1 de 90%.

Ao passar o décimo de existência do sistema mais três nodos ficaram *offline*. Foram

nodos de 70%, 80% e 90%, respectivamente. Não foi possível localizar um total de 3 objetos de 30% de confiabilidade desejada, 4 objetos de 50%, 4 de 70% e 2 de 90%. No final do décimo quinto ano mais um nodo de 80% de confiabilidade se desconectou e os mesmos objetos do décimo ano foram possíveis de se localizar.

No vigésimo ano, 1 nodo de 50% se desconectou, sobrando apenas um nodo de 80% e outro de 90%. Nesta idade do sistema era possível localizar todos os objetos de 99% de confiabilidade desejada e 1 de 90%. Os outros todos já tinham sido perdidos.

No vigésimo quinto ano o nodo de 80% falhou e ainda era possível recuperar 4 objetos de 99% e 1 de 90%. No trigésimo ano o último nodo se desconectou da rede e nenhuma réplica mais existia.

5.3.3 Comentários

Encaramos a confiabilidade de cada nodo como a probabilidade dele não perder nenhuma informação no período de 1 ano. Ao simular o passar dos anos, os experimentos comprovaram a capacidade do sistema em preservar informações por um longo período de tempo. A importância pela preservação de cada objeto digital – medido no modelo pela confiabilidade desejada – impacta em diferentes tempo de vida do mesmo. Objetos que foram inseridos com uma confiabilidade desejada maior tiveram um tempo de vida maior. Assim, através dos experimentos, podemos concluir duas principais características do sistema:

1. **Independência de preservação dos objeto**. Diferentes informações requerem diferentes tempo de armazenamento. Algumas coleções de fotos, revistas e artigos podem precisar de poucos anos de armazenamento enquanto outras informações, como objetos digitais de museus e de bibliotecas, necessitam de centenas de anos. Nosso sistema permite total flexibilidade na escolha do tempo de vida dos objetos a serem preservados.

2. **Otimização dos recursos de armazenamento**. Repositórios de armazenamento podem sofrer várias ameaças em seus conteúdos, portanto cada repositório tem uma diferente confiabilidade. Assim, modelar cada repositório de armazenamento com

um parâmetro capaz de medir sua probabilidade independente de falha é a maneira mais próxima de modelar redes reais. Tal maneira permite a flexibilidade no tempo de preservação de cada objeto e conseqüentemente diferente números de réplicas para eles, impactando em uma melhor utilização dos recursos de armazenamento da rede.

Capítulo 6

Conclusão

Garantir o acesso, a confiabilidade e a integridade dos objetos digitais a longo prazo na sociedade atual é uma prioridade. Constantes ameaças aos objetos existem e sistemas tradicionais de armazenamento não são suficientes para evitá-las. Sistemas baseados em comunidades autônomas de repositórios aparecem como uma alternativa para a tarefa do arquivamento a longo prazo desses objetos.

Esta dissertação apresentou um modelo de replicação confiável de dados para sistemas de arquivamento digital. A principal característica deste modelo é na associação de probabilidades independentes de falha para cada repositório de armazenamento as quais possibilitam que itens sejam guardados conforme suas necessidades pelo tempo de preservação. Além disso, foi proposto três estratégias de replicação que visam a otimização na inserção de itens no sistema. Através de simulações constatamos que dentre as três estratégias, a estratégia do Subconjunto Ideal mostrou-se a mais equilibrada nos quesitos de minimizar as réplicas e o *makespan* e ainda teve a melhor taxa de inserção de objetos. Tal estratégia apresenta uma complexidade computacional que não poderia ser possível de ser executada na prática caso não fosse cuidadosamente formulada.

Devido a alta escalabilidade para a distribuição e recuperação de dados, as redes P2P estruturadas aparecem como candidatas naturais para implementar o modelo do sistema de arquivamento proposto. As P2P estruturadas têm uma dificuldade inerente ao modo de roteamento das informações que dificulta a seleção de nodos (repositórios)

específicos para o armazenamento das réplicas. Entretanto, esta dificuldade foi suprida através da utilização de múltiplas funções *hash*. A implementação deste sistema P2P de arquivamento digital foi avaliada através de experimentos em um ambiente real.

Como trabalho futuro se propõe avaliar a robustez do sistema de arquivamento digital. Em particular, deve ser investigado como os algoritmos se comportariam variando outros parâmetros como o tamanho da rede, a confiabilidade desejada dos itens e etc. Outras estratégias de replicação, tais como o *Erasure code*, também podem ser exploradas pois oferecem custos pequenos de armazenamento comparado a replicação, caso os dados tenham tamanho grande. Entretanto a DHT teria que gerenciar um maior número de itens nas tabelas para um mesmo conjunto de objetos. Avaliar o desempenho neste tipo de ambiente é um possível trabalho.

Também como trabalho futuro está a implementação de um sistema completo de preservação digital. Muita pouca ênfase foi dada para a auditoria das réplicas e recuperação dos itens. Além disso o sistema também deveria se preocupar com outras ameaças tais como obsolescência de *software* e formato, não tratadas neste trabalho. Eventualmente utilizar o nosso modelo de arquivamento digital sobre sistemas como o SAV, BRICKS e LOCKSS e avaliar sua viabilidade também é interessante.

Apêndice A

Solução da Estratégia do Subconjunto Ideal

O problema da mochila (*knapsack problem*) é um problema conhecido na Teoria da Computação. Dado um conjunto de itens, cada qual com um peso e um valor associado, estamos interessados em determinar o número itens a serem incluídos a uma coleção tal que o peso total seja menor que um dado limite e o valor total seja o máximo possível [55]. Formalmente, dado n itens, cada item j tem um valor p_j e um peso w_j. O peso máximo que a mochila suporta é W, então:

$$\begin{aligned} &\max \sum_{j=1}^{n} p_j x_j \\ &\text{s.t.} \sum_{j=1}^{n} w_j x_j \leq W \qquad x_j \in \{0,1\} \end{aligned} \tag{A.1}$$

O problema da mochila é NP-DIFÍCIL e, assumindo P $\neq$ NP, não existe algoritmo polinomial que resolva este problema. Entretanto, para propósitos práticos, o problema da mochila é resolvido em tempo pseudo-polinomial utilizando programação dinâmica. A solução é descrita a seguir.

Inicialmente definimos $B[k, w]$ recursivamente da seguinte maneira:

$B[0, w] = 0 \qquad \forall w \in \{0, \ldots, W\}$; $B[k, 0] = 0 \qquad \forall k \in \{0, \ldots, n\}$

$$B[k,w] = \begin{cases} B[k-1,w] & \text{se } w_k > w \\ max\{B[k-1,w], B[k-1,w-w_k] + p_k\} & \text{c.c.} \end{cases} \tag{A.2}$$

$B[k,w]$ forma uma tabela ao qual é possível encontrar o valor máximo a ser carregado em uma mochila, ou seja, o valor em $B[n,W]$. Entretanto, para saber quais são os items que levam a este valor é necessário um segundo passo, a ser aplicado sob a tabela já preenchida. O Algoritmo 7 descreve o segundo passo.

```
input: Let i = n, k = W
begin
    while i, k > 0 do
        if B[i,k] ≠ B[i-1,k] then
            mark the i^th item as in the knapsack
            k = k - w_i
        i = i - 1
end
```

Algoritmo 7: Knapsack

Notemos a semelhança da definição do problema da mochila (A.1) com a definição do problema de seleção de réplicas utilizando a estratégia do Subconjunto Ideal (A.3). Eles são muito parecidos e o mesmo esquema de construção da tabela do algoritmo de programação dinâmica pode ser utilizado para este último.

$$\begin{aligned} &\min \sum_{\ell_j \in S} a_j x_j - b_i \\ &\text{s.t.} \sum_{\ell_j \in S} a_j x_j \geq b_i \qquad x_j \in \{0,1\} \qquad \forall \ell_j \in S \end{aligned} \tag{A.3}$$

No problema de seleção de réplicas através do Subconjunto Ideal, apenas os valores maiores que b_i são de interesse. Assim, a tabela de programação dinâmica deve ser expandida com mais valores nas colunas de modo que seja possível obter o valor que aproxime do ótimo, $B[n][W]$.

Referências Bibliográficas

[1] C. Webb. Guidelines for the preservation of digital heritage, 2003.

[2] Brian Cooper and Hector Garcia. Creating trading networks of digital archives. In *JCDL '01: Proceedings of the 1st ACM/IEEE-CS joint conference on Digital libraries*, pages 353–362, New York, NY, USA, 2001. ACM.

[3] P. Maniatis, M. Roussopoulos, T. Giuli, D. Rosenthal, and M. Baker. The lockss peer-to-peer digital preservation system, 2005.

[4] Thomas Risse and Predrag Knezevic. A self-organizing data store for large scale distributed infrastructures. In *ICDEW '05: Proceedings of the 21st International Conference on Data Engineering Workshops*, Washington, DC, USA, 2005. IEEE Computer Society.

[5] Michael Mitzenmacher and Eli Upfal. *Probability and Computing : Randomized Algorithms and Probabilistic Analysis*. 2005.

[6] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2001.

[7] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172, New York, NY, USA, 2001. ACM.

[8] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.

[9] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, and John D. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41–53, January 2004.

[10] Kenneth Thibodeau. Overview of technological approaches to digital preservation and challenges in coming years. In *The State of Digital Preservation: An International Perspective*, Washington D.C., 2002.

[11] Vicky Reich and David S.H. Rosenthal. Lockss: A permanent web publishing and access system. Technical report, 2001.

[12] Maohua Lu and Tzi cker Chiueh. Challenges of long-term digital archiving: A survey. *Technical Report ECSL-TR-205*, 2006.

[13] David S. H. Rosenthal, Thomas S. Robertson, Tom Lipkis, Vicky Reich, and Seth Morabito. Requirements for digital preservation systems: A bottom-up approach, 2005.

[14] Mary Baker, Mehul Shah, David S. H. Rosenthal, Mema Roussopoulos, Petros Maniatis, TJ Giuli, and Prashanth Bungale. A fresh look at the reliability of long-term digital storage. In *EuroSys '06: Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*, pages 221–234, New York, NY, USA, 2006. ACM.

[15] Gregory W. Lawrence, William R. Kehoe, Oya Y. Rieger, William H. Walters, and Anne R. Kenney. *Risk Management of Digital Information: A File Format Investigation*. Council on Library and Information Resources, June 2000.

[16] Miguel Ferreira. Introducao a preservacao digital - Conceitos, estrategias e actuais consensos. pages 409–418, Guimaraes, Portugal, 2006. Escola de Engenharia da Universidade do Minho.

[17] Wine HQ Web site. Disponivel em: http://www.winehq.com. Acesso em junho de 2008.

[18] VMWare Workstation Web site. Disponivel em: http://www.vmware.com. Acesso em junho de 2008.

[19] Thomas J. E. Schwarz, Qin Xin, Ethan L. Miller, Darrell D. E. Long, Andy Hospodor, and Spencer Ng. Disk scrubbing in large archival storage systems. In *MASCOTS '04: Proceedings of the The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'04)*, pages 409–418, Washington, DC, USA, 2004. IEEE Computer Society.

[20] Estimating Drive Reliability in Desktop Computers and Consumer Electronics Systems. Disponivel em: www.seagate.com/. Acesso em junho de 2008.

[21] Barbara Liskov, Sanjay Ghemawat, Robert Gruber, Paul Johnson, and Liuba Shrira. Replication in the harp file system. *SIGOPS Oper. Syst. Rev.*, 25(5):226–238, 1991.

[22] Brian Cooper, Arturo Crespo, and Hector Garcia. Implementing a reliable digital object archive. In *ECDL '00: Proceedings of the 4th European Conference on Research and Advanced Technology for Digital Libraries*, London, UK, 2000. Springer-Verlag.

[23] Brian Cooper and Hector Garcia. Peer-to-peer data preservation through storage auctions. *IEEE Trans. Parallel Distrib. Syst.*, 16(3):246–257, 2005.

[24] George Schusse. Client/server: Past, present and future. Disponivel em: http://www.dciexpo.com/geos/dbsejava.htm. Acesso em junho de 2008.

[25] Kai Fischbach, Christian Schmitt, and Detlef Schoder. Core concepts in peer-to-peer networking. In *Peer to Peer Computing: The Evolution of a Disruptive Technology*, chapter 1. Idea Group Inc., 2005.

[26] R. Schollmeier. A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications. In *P2P '01: Proceedings of the First International Conference on Peer-to-Peer Computing (P2P'01)*, page 101, Washington, DC, USA, 2001. IEEE Computer Society.

[27] Clay Shirky. What is p2p... and what isn't? Disponivel em: www.openp2p.com/pub/a/p2p/2000/11/24/shirky1-whatisp2p.html. Acesso em junho de 2008.

[28] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *Int. J. High Perform. Comput. Appl.*, 15(3):200–222, August 2001.

[29] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. A Survey of Peer-to-Peer Content Distribution Technologies. *ACM Comput. Surv.*, 36(4):335–371, December 2004.

[30] Gnutella. Gnutella website. Disponivel em: www.gnutella.com/. Acesso em dezembro de 2008, 2008.

[31] W. T. Sullivan III, D. Werthimer, S. Bowyer, J. Cobb, D. Gedye, and D. Anderson. A new major seti project based on project serendip data and 100,000 personal computers. In *Proceedings of the 5th International Conference on Bioastronomy*, 1997.

[32] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: a distributed anonymous information storage and retrieval system. In *International workshop on Designing privacy enhancing technologies*, New York, USA, 2001. Springer-Verlag Inc.

[33] Jabber – Open Instant Messaging and Presence. Disponivel em: http://www.jabber.org/. Acesso em junho de 2008.

[34] The eMule-Project. Disponivel em: www.emule-project.net/. Acesso em maio de 2008.

[35] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in communications (JSAC)*, 20(8):1489–1499, 2002.

[36] Ramaprabhu Janakiraman, Marcel Waldvogel, and Qi Zhang. Indra: A peer-to-peer approach to network intrusion detection and prevention. In *Proceedings of IEEE WETICE 2003*, June 2003.

[37] P. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini, and I. Zaihrayeu. Data management for peer-to-peer computing: A vision. In Workshop on the Web and Databases, WebDB 2002., 2002.

[38] R. Huebsch, J. M. Hellerstein, N. L. Boon, T. Loo, S. Shenker, and I. Stoica. Querying the internet with pier. In *In Proceedings of 19th International Conference on Very Large Databases (VLDB)*, 2003.

[39] Kazaa website. Disponivel em: www.kazaa.com/. Acesso em junho de 2008.

[40] Qin Lv, Pei Cao, Edith Cohen, Kai Li, and Scott Shenker. Search and replication in unstructured peer-to-peer networks. In *SIGMETRICS '02: Proceedings of the 2002 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 258–259, New York, NY, USA, 2002. ACM.

[41] Kunwadee Sripanidkulchai. The popularity of Gnutella queries and its implications on scalability. Disponivel em: http://www.cs.cmu.edu/ kunwadee/research/p2p/gnutella.html. Acesso em junho de 2008.

[42] Napster website. Disponivel em: www.napster.com/. Acesso em junho de 2008.

[43] Ali Ghodsi. Distributed k-ary system: Algorithms for distributed hash tables. In *PhD dissertation*, KTH-Royal Institute of Technology, October 2006.

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

[45] Y. Saito and M. Shapiro. Optimistic replication, 2005.

[46] Vidal Martins, Esther Pacitti, and Patrick Valduriez. Survey of data replication in P2P systems, 2006.

[47] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric, 2002.

[48] Jinoh Kim and Eric Chan-Tin. Robust Object Replication in a DHT Ring, 2007.

[49] Salma Ktari, Mathieu Zoubert, Artur Hecker, and Houda Labiod. Performance evaluation of replication strategies in DHTs under churn. In *MUM '07: Proceedings of the 6th international conference on Mobile and ubiquitous multimedia*, pages 90–97, New York, NY, USA, 2007. ACM.

[50] Ali Ghodsi, Luc Onana Alima, and Seif Haridi. Symmetric replication for structured peer-to-peer systems. In *DBISP2P*, pages 74–85, 2005.

[51] Ye Xia, Shigang Chen, and Vivekanand Korgaonkar. Load balancing with multiple hash functions in peer-to-peer networks. In *ICPADS '06: Proceedings of the 12th International Conference on Parallel and Distributed Systems*, pages 411–420, Washington, DC, USA, 2006. IEEE Computer Society.

[52] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. 1979.

[53] M. Nelson C. Lagoze, H. Van de Sompel and S. Warner. The Open Archives Initiative Protocol for Metadata Harvesting, version 2.0. Disponivel em: www.openarchives.org/OAI/openarchivesprotocol.html. Acesso em junho 2008.

[54] Kazuyuki Shudo, Yoshio Tanaka, and Satoshi Sekiguchi. Overlay weaver: An overlay construction toolkit. *Comput. Commun.*, 31(2):402–412, 2008.

[55] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition*. McGraw-Hill Science/Engineering/Math, July 2001.