Modelo Estrutural para Compartilhamento de Arquivos Peer-to-Peer

Evandro da Silva Rezende

Modelo Estrutural para Compartilhamento de Arquivos Peer-to-Peer

Evandro da Silva Rezende

Orientadora: ***Prof[a]. Dr[a]. Roberta Spolon***

Dissertação de Mestrado elaborada junto ao Programa de Pós-Graduação em Ciência da Computação - Área de Concentração em Sistemas de Computação, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação

São José do Rio Preto, São Paulo
Julho de 2009

Agradecimentos

Em primeiro lugar, gostaria de agradecer a Professora Doutora Roberta Spolon, pela confiança em mim depositada, pela orientação, dedicação e acompanhamento que recebi durante a realização deste trabalho.

Gostaria de agradecer a todos os professores com os quais tive contato durante o Mestrado, por tudo que acrescentaram na minha formação: Hilda Carvalho de Oliveira, Marcos Antônio Cavenaghi, Ivan Rizzo Guilherme e Aparecido Nilceu Marana.

Gostaria de agrader a MStech e todos os seus colaboradores, que conviveram comigo durante este trabalho. Agradeço ao Lucas Gomes e Alex Branti, que me auxiliaram muito quando iniciei meu trabalho com a plataforma Mozilla.

Também, gostaria de agradecer ao LTIA e ao LCAD, por todo o suporte oferecido ao meu trabalho, estando sempre de portas abertas quando precisei me dedicar com maior intensidade ao mestrado.

Além disso, gostaria de agradecer a Juliana, que esteve presente em todos os momentos desta fase, sempre disposta a fazer o necessário para me auxiliar, como acompanhar, com muita paciência, palestras sobre sistemas distribuídos, grids e todo o conteúdo presente neste documento.

E, é claro, gostaria de agradecer a uma peça fundamental do trabalho aqui apresentado, meu amigo Evgueni Dodonov, que me ajudou desde o ínicio desta jornada, inclusive nos momentos mais difíceis.

Agradeço a minha familia, fundamental em tudo que fiz até hoje, e aos meus amigos, que conviveram comigo durante esta fase.

Resumo

O processo de compartilhamento de arquivos passou por grande evolução desde a criação dos computadores, até os dias atuais. Esta evolução teve início com o compartilhamento de dados em computadores de grande porte, passando pela criação das mídias removíveis e, posteriormente, beneficiando-se dos avanços nas tecnologias de redes de computadores. Nesta última, é possível verificar duas fases distintas, a primeira, em que a arquitetura predominante foi cliente/servidor, e a segunda, na qual busca-se voltar às origens da Internet, e utilizar largamente o conceito de computação peer-to-peer. Apesar de todo avanço ocorrido com o compartilhamento de arquivos peer-to-peer, desde seu surgimento no final da década de noventa até os dias atuais, verifica-se que todo o avanço ocorrido ainda não foi suficiente para garantir a plena excelência no que esta tecnologia foi proposta, facilitar a troca de arquivos entre usuários e sistemas. Assim, esta dissertação apresenta um modelo de indexação e acesso a arquivos compartilhados, de maneira totalmente descentralizada e autônoma. Além disso, busca-se criar mecanismos de acesso ao conteúdo compartilhado através de padrões da indústria, oferecendo acesso a estes arquivos para diversas aplicações e sistemas, por exemplo, navegadores web, o qual foi foco de um estudo de caso neste trabalho, onde foi possível verificar todos os benefícios e viabilidade de tal modelo.

Abstract

The file sharing process has evolved significantly since the creation of the first personal computers to the present days. Such evolution begun with the possibility of data sharing among the users of a same supercomputer, passed through the creation of removable storage devices and, after that, took advantage in the technology advances in computer networks. In this last phase, it is possible to distinguish two different approaches one based on a client-server architecture, and another aiming to use the peer-to-peer concepts for data access. The growth in the adoption of peer-to-peer approaches, however, still have not been enough to guarantee the primary goal of this approach, which is to simplify access to shared data, due to the existence of multiple incompatible peer-to-peer networks. In this context, this dissertation presents a completely decentralized and autonomic approach to data indexing and access. In this sense, this approach intends to provide mechanisms to access data stored on independent peer-to-peer networks using industry standards, and offering access to such data to user and system applications, such as the web browsers, which was the focus of study case in this work, and made possible to verify all it's benefits and feasibility.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

AFS	Andrew File System.
API	Application Programming Interface.
CDN	Content Delivery Network.
CODA	Constant Data Availability.
CPU	Central Processing Unit.
CSL	Centralized Service Locator.
DHT	Distributed Hash Table.
DSM	Distributed Shared Memory.
EP	Elemento Processador.
FSL	Flooding-based Service Locator.
FTP	File Transfer Protocol.
HTTP	Hypertext Transfer Protocol.
IP	Internet Protocol.
LAN	Local Area Network.
MPI	Message Passing Interface.
NASA	National Aeronautics and Space Administration.
NFS	Network File System.
NFSP	Netware File Sharing Protocol.
NPFS	Network Parallel File System.
pNFS	Parallel Network File System.
PVFS	Parallel Virtual File System.
PVM	Parallel Virtual Machine.
RDF	Resource Description Framework.
RMI	Remote Method Invocation.
RPC	Remote Procedure Call.
SAD	Sistema de Arquivos Distribuído.
SAP	Sistema de Arquivos Paralelo.
SOAP	Simple Object Access Protocol.
SSI	Single System Image.
TCP	Transmission Control Protocol.
UDDI	Universal Description, Discovery and Integration.
VO	Virtual Organization.
WSDL	Web Service Definition Language.
XML	Extensible Markup Language.
XPCOM	Cross Platform Component Object Model.
XUL	XML User Interface Language.

CAPÍTULO

1

Introdução

1.1 Considerações Preliminares

Desde a criação dos computadores eletrônicos por Von Neuman nos anos quarenta, até os dias atuais das redes de longa distância e alta velocidade, os processos de desenvolvimento de *software* vêm passando por grandes evoluções, em parte devido à necessidade de adequação à evolução dos meios computacionais e de outro lado, pela necessidade de atender a expansão na utilização de computadores no mundo moderno.

Quando se trata, específicamente, das tecnologias para o compartilhamento de arquivos, podem ser constatadas muitas mudanças nas técnicas utilizadas de acordo com os recursos tecnológicos existentes, passando pelos computadores de grande porte, criação das mídias removíveis, desenvolvimento dos microprocessadores e avanços nas tecnologias de rede.

Até o final da década de setenta, a forma mais comum de compartilhamento de arquivos digitais entre *hosts* era realizado por meio de de mídias removíveis (como por exemplo, disquetes). Desta forma, para que uma pessoa pudesse ter acesso a arquivos que não estivessem armazenados em sua máquina, ela deveria armazenar uma cópia do arquivo desejado em uma dessas mídias e, então, transportá-la até o computador destino.

A partir da segunda metade dos anos oitenta, o desenvolvimento de microprocessadores e avanços nas tecnologias de redes de computadores mudaram a computação e possibilitaram o surgimento de um novo paradigma computacional, os sistemas distribuídos (Internet e Intranet) (Tanenbaum & van Steen, 2006). Desta forma, a criação de novos mecanismos baseados em redes de computadores mostrou-se necessária.

O surgimento do *FTP*, em meados da década de oitenta, promoveu grandes mudanças no que se refere ao compartilhamento de arquivos, possibilitando a troca e manipulação de arquivos

em redes de computadores baseadas no protocolo TCP (Barakat, 2001). Através da arquitetura cliente/servidor, aplicações clientes *FTP* deveriam se conectar a servidores *FTP* para manipular os arquivos neles hospedados.

A tecnologia *IRC* (*Internet Relay Chat*) possibilitou o compartilhamento de arquivos para os usuários de salas de *chat*. Posteriormente, tal tecnologia foi aprimorada com a introdução do protocolo *XDCC* , o qual eliminava a necessidade de interação dos usuários do *IRC* durante o compartilhamento de arquivos.

A contínua ascensão da Internet e o aumento no número de computadores ao redor do mundo fizeram com que um dos principais conceitos por trás da Internet se tornasse cada vez mais popular a partir do final da década de noventa, a computação *peer-to-peer (P2P)*. Esta volta às origens da Internet teve como principais responsáveis as aplicações destinadas ao compartilhamento de arquivos, com início em 1999 a partir da criação do Napster (Oram, 2001) por Shawn Fanning.

Após a criação do Napster, o crescimento do número de protocolos *P2P* aumentou de maneira exponencial, promovendo o surgimento de redes *P2P* com diferentes arquiteturas e topologias. Porém, apesar de existir grandes diferenças na forma como cada protocolo trabalha, verifica-se que o objetivo de cada um acaba sendo muito semelhante aos demais pelo fato de promover a criação de redes fechadas e incompatíveis.

Assim, os dados disponibilizados em redes *P2P* são, freqüentemente, redundantes e com acesso restrito à determinada aplicação cliente. Tal cenário tem motivado estudos para promover a interoperabilidade entre tais protocolos e também estudos que buscam novos meios de acesso a arquivos compartilhados, como é o foco deste trabalho (Rezende *et al.*, 2009), e também de outros trabalhos que começaram a surgir em meados de 2009, como o Opera Unite [1].

1.2 Motivação

O aumento no número de computadores em uso ao redor do mundo, somado à oferta de redes de interconexão com largura de banda cada vez maior e mais popular, têm estimulado a construção de sistemas que visam beneficiar-se da interconexão dos recursos presentes nestes computadores, independentemente de sua localização (Pietro, 2001), o que resulta em um grande número de inovadoras aplicações, as quais são utilizadas para o compartilhamento dos mais diversos recursos, entre eles, arquivos digitais como músicas e vídeos.

Apesar de todo o avanço obtido desde a criação do Napster, as características dos sistemas para compartilhamento de arquivos *P2P* atualmente existentes evidenciam diversos problemas. As diversas topologias de redes de sobreposição existentes (Androutsellis-Theotokis & Spinellis, 2004), somadas ao uso de diferentes protocolos, tornam necessária a utilização de diferentes

[1] unite.opera.com/

aplicações para permitir o uso dos arquivos compartilhados nestas redes.

Além disso, verifica-se que as redes de compartilhamento de arquivos atualmente existentes deixam a desejar no quesito qualidade de serviço. De um lado, as arquiteturas centralizadas, utilizadas em sistemas como o Napster, apresentam desvantagens intrínsecas ao modelo cliente/servidor, ou seja, menor escalabilidade e disponibilidade em virtude da sobrecarga em número relativamente pequeno de servidores, e a existência de pontos centrais de falhas.

Por outro lado, têm-se os sistemas totalmente descentralizados, baseados nas chamadas redes *P2P* puras. Apesar de estes sistemas não apresentarem os mesmos problemas encontrados na arquitetura cliente/servidor, nota-se que os mecanismos criados pelos mesmos para gerenciar a topologia da rede criada aumentaram a complexidade em seu desenvolvimento, além de contribuir com a perda de desempenho em operações de busca por arquivos.

Atualmente, também se observa um grande número de redes *P2P* baseadas em arquiteturas hibridas, ou seja, utilizam a arquitetura cliente/servidor através do uso de super-nós, responsáveis pela realização de tarefas críticas, como a indexação de conteúdo e a autenticação de usuários. Apesar de esta arquitetura eliminar vários problemas da abordagem totalmente descentralizada, ela apresenta como desvantagens a perda de escalabilidade e menor disponibilidade em decorrência da sua centralização parcial.

Apesar da diversidade de protocolos que surgiram, percebe-se que o compartilhamento de arquivos entre dois indivíduos ainda precisa se beneficiar de maiores facilidades para que o principal objetivo destas aplicações seja alcançado.

Em virtude de todas as limitações apresentadas pelos sistemas atualmente utilizados, percebe-se a necessidade da criação de novos modelos de compartilhamento de arquivos *P2P*, assim como mecanismos que possibilitem aos usuários aproveitar a grande quantidade de objetos já compartilhados na Internet e em outras redes *P2P*, independentemente do protocolo utilizado.

1.3 Objetivos

Este trabalho apresenta um modelo de indexação e acesso a arquivos disponíveis em repositórios da Internet, como servidores *FTP*, redes *P2P* já existentes, assim como novos repositórios que venham a ser criados especificamente para agregar ainda mais recursos a rede criada.

O modelo apresentado é baseado em um repositório global de índices, criado a partir da organização de computadores em uma rede *P2P* estruturada. A partir da disponibilização de tais informações, através de padrões da indústria, busca-se possibilitar que diversos tipos de aplicações tenham acesso ao compartilhamento de arquivos *P2P*.

Também, é objetivo deste trabalho criar uma *API* para acesso ao repositório de índices, assim como *download* dos arquivos descritos no mesmo. Neste estudo, tal *API* foi implementada na linguagem *Python* e oferece suporte a um número limitado de protocolos, podendo ser estendida em trabalhos futuros.

1.4 Estrutura da Monografia

Além do capítulo inicial, esta monografia é organizada em mais quatro capítulos, descritos a seguir:

- ***Capítulo 2:*** aborda os sistemas distribuídos e apresenta o contexto de seu desenvolvimento, principais características e exemplos de sistemas desta classe atualmente utilizados. Também são apresentadas algumas tecnologias que visam oferecer o compartilhamento de dados nestes sistemas.
- ***Capítulo 3:*** apresenta os sistemas *peer-to-peer*, exibindo motivações para seu desenvolvimento, principais características, classificações dos mesmos, tipos de aplicações que atualmente utilizam esta tecnologia, desafios inerentes e o estado atual.
- ***Capítulo 4:*** apresenta o trabalho desenvolvido.
- ***Capítulo 5:*** apresenta as conclusões, contribuições e indicações de trabalhos que podem ser realizados futuramente, com base no trabalho desenvolvido.

CAPÍTULO

2

Sistemas Distribuídos

2.1 Considerações Preliminares

Desde a criação dos computadores eletrônicos por Von Neuman, nos anos quarenta, até os dias atuais, os sistemas computacionais vêm sofrendo constante evolução. Até o início dos anos oitenta, os computadores eram grandes e caros. Segundo a lei de Grosch (Ein-Dor, 1985), o poder computacional de uma *CPU* era proporcional ao quadrado de seu preço. Desta forma, as empresas necessitavam gastar grandes quantias de dinheiro com máquinas caras, para que seus requisitos computacionais fossem atendidos.

A partir da segunda metade dos anos oitenta, o desenvolvimento de microprocessadores, somado aos avanços nas tecnologias de redes de computadores, mudou esta situação e possibilitou o surgimento de um novo paradigma computacional, os sistemas distribuídos (Tanenbaum & van Steen, 2006). Como exemplos de sistemas distribuídos, podem ser citados a Internet, a Intranet, a computação móvel e a computação ubíqua.

Além do agrupamento de elementos processadores, a possibilidade de compartilhamento dos mais diversos tipos de recursos também passou a ser grande atrativo para o desenvolvimento de tais sistemas (Dollimore *et al.*, 2005). Alguns recursos que podem ser compartilhado nestes sistemas são espaço de armazenamento, aplicações, presença humana e *hardware*.

De acordo com a literatura existente, diversas definições podem ser dadas para os sistemas distribuídos. Segundo Silberschatz *et al.* (2004), um sistema distribuído é uma coleção de processadores fracamente acoplados e interconectados por uma rede de comunicação. Do ponto de vista de um processador específico, os outros processadores e seus respectivos recursos são remotos enquanto o mesmo possui recursos próprios e locais.

Para Dollimore *et al.* (2005), um sistema distribuído é aquele no qual componentes

de computadores conectados em rede se comunicam apenas através de mensagens. Algumas características destes sistemas são concorrência de componentes e falhas de componentes independentes, ou seja, a falha de um componente não resulta na parada de funcionamento do sistema como um todo.

Em (Tanenbaum & van Steen, 2006), a definição de um sistema distribuído é dada como uma coleção de computadores que aparece para o usuário do sistema como um único computador. Além disso, os *hardwares* destes sistemas devem apresentar comportamento autônomo.

Apesar de diversas definições, pode-se dizer que o principal objetivo dos sistemas distribuídos é o compartilhamento de recursos lógicos e/ou físicos. O trabalho neste tipo de sistema recebe o nome de computação distribuída (Reis, 2005).

2.2 Características

As principais características dos sistemas distribuídos são (Dollimore *et al.*, 2005; Tanenbaum & van Steen, 2006; Oliveira, 2006):

- ***Abertura:*** um sistema aberto é aquele que pode ser estendido e reimplementado de diversas maneiras. Tal característica é obtida por meio da definição de claras interfaces de *software* e mecanismos de comunicação uniforme. Desta forma, é possível criar sistemas distribuídos a partir de *softwares* e *hardwares* heterogêneos;
- ***Heterogeneidade:*** apesar de não ser uma regra, geralmente os sistemas distribuídos são compostos por elementos heterogêneos (rede, *hardware*, sistema operacional, linguagem de programação). Não obstante, os sistemas distribuídos devem apresentar mecanismos que possibilitem o funcionamento e o desenvolvimento de tarefas sem restrições;
- ***Transparência:*** a transparência é alcançada a partir do momento que o sistema oculta a separação dos componentes distribuídos para o usuário final e programadores de aplicativos. Segundo Tanenbaum & van Steen (2006), diferentes tipos de transparência podem ser obtidos:
  - ***Transparência de localização:*** os usuários não identificam onde os recursos estão localizados;
  - ***Transparência de migração:*** recursos podem ser movimentados sem a alteração em seus nomes;
  - ***Transparência de replicação:*** impossibilidade dos usuários dizerem o número de cópias existentes de determinado recurso;
  - ***Transparência de concorrência:*** múltiplos usuários podem compartilhar recursos automaticamente;

  - ***Transparência de paralelismo:*** atividades podem ocorrer de forma paralela sem o conhecimento do usuário.

- ***Concorrência:*** ocorre quando determinado recurso do sistema é requisitado por mais de um usuário em determinado instante. Os sistemas distribuídos devem conter mecanismos que tratem tal situação;

- ***Escalabilidade:*** um sistema distribuído deve trabalhar de forma eficiente mesmo com o aumento do número de recursos e usuários. A Internet é um exemplo de sistema distribuído em que o número de participantes e serviços aumentaram de forma substancial nos últimos anos, como pode ser visto na tabela 2.1.

Tabela 2.1: Crescimento no número de computadores e usuários na Internet (Dollimore *et al.*, 2005).

Data	Computadores	Serviços web
Dez. 1979	188	0
Jul. 1989	130.000	0
Jul. 1999	56.218.000	5.560.956
Jul. 2003	171.638.297	35.424.956

- ***Tolerância a falhas:*** diferentemente dos sistemas centralizados, falhas nos sistemas distribuídos são parciais, isto é, quando ocorre falha em um dos componentes, os demais devem continuar operando normalmente. Desta forma, são necessários mecanismos complexos para o tratamento de falhas. Diversas técnicas para tratamento de falhas são apresentadas em (Dollimore *et al.*, 2005).

2.3 Estado Atual

A seguir, são descritas algumas tecnologias utilizadas atualmente em sistemas distribuídos: *clusters* (Yeo *et al.*, 2006), computação em grade (Dias, 2006; Foster *et al.*, 2001), sistemas *P2P* (Francesquini, 2007; Hu, 2005; Kubiatowicz, 2003; Androutsellis-Theotokis & Spinellis, 2004) e agentes móveis (Pietro, 2001).

2.3.1 Cluster

A possibilidade de distribuir o processamento e acesso aos dados em uma rede local levou a criação de ambientes distribuídos conhecidos como *clusters* computacionais (Yeo *et al.*, 2006).

Segundo Yeo *et al.* (2006), um *cluster* computacional pode ser definido como um conjunto fracamente acoplado de computadores independentes, agrupados para a realização de uma tarefa computacional, visando oferecer uma imagem única do sistema (*SSI - Single System Image*), conforme mostra a figura 2.1.

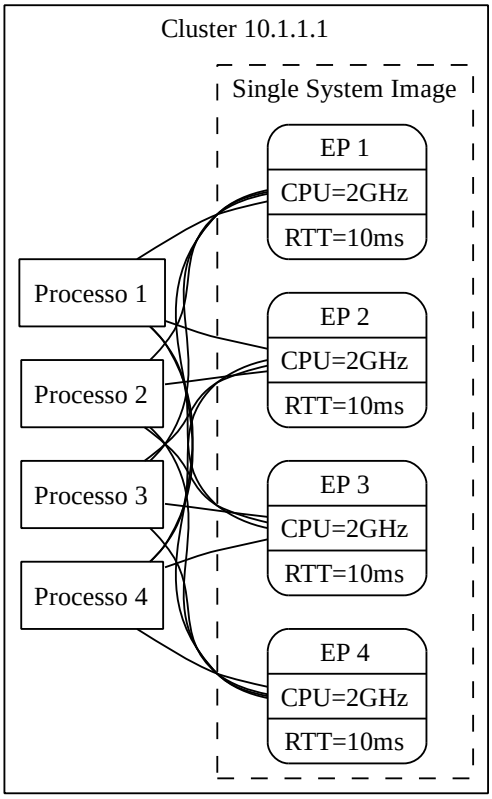


Figura 2.1: Arquitetura de um cluster computacional (Dodonov, 2009).

Entre as principais características desta tecnologia, pode ser destacada a atuação dentro de um único domínio administrativo e o uso de recursos (*hardware* e *software*) homogêneos e dedicados.

Segundo suas funcionalidades, os *clusters* computacionais podem ser classificados em (Yeo *et al.*, 2006):

- ***Clusters de alta disponibilidade (High Availability - HA):*** visam aumentar a disponibilidade dos serviços oferecidos. Para tanto, são utilizados conceitos de substituição transparente dos *EPs* que apresentam falhas;
- ***Clusters de balanceamento de carga:*** visam balancear a carga de processamento do sistema entre seus *EPs*, minimizando ociosidade dos mesmos;
- ***Clusters de armazenamento:*** visam oferecer maior capacidade de acesso aos dados, provendo menor latência e maior vazão;
- ***Clusters de processamento:*** visam reduzir o tempo de execução de um problema computacional por meio da subdivisão da aplicação em tarefas menores e divisão das mesmas entre os *EPs*.

2.3.2 Computação em Grade

O aumento contínuo da complexidade das tarefas a serem solucionadas e o alto custo de recursos especializados, tais como máquinas paralelas e *clusters*, tornaram evidente a necessidade de uma tecnologia capaz de oferecer acesso a recursos distribuídos em larga escala (Foster *et al.*, 2001).

Em meados dos anos noventa, foi proposta uma infra-estrutura de *hardware* e *software* com suporte à execução de aplicações paralelas e acopladas a recursos distribuídos e heterogêneos. Essa infra-estrutura foi denominada computação em grade ou grades computacionais (*grid computing*) (Foster *et al.*, 2001). A arquitetura típica de uma grade é composta por nós e gerenciadores, onde os nós são responsáveis pelo compartilhamento de recursos e os gerenciadores são os responsáveis pela administração, coleta de informação e escalonamento de tarefas (Figura 2.2).

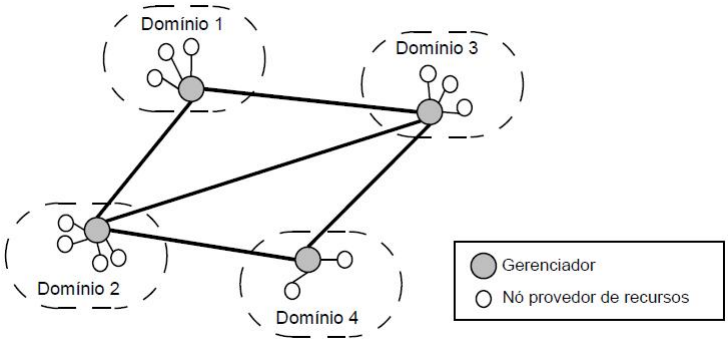


Figura 2.2: Arquitetura típica de uma grade (Rocha *et al.*, 2004).

A denominação computação em grade foi utilizada como metáfora às malhas de interligação dos sistemas de energia elétrica, os chamados *power-grids*, que possibilitam o uso da rede elétrica sem a preocupação de como e onde a energia consumida é gerada (Dias, 2006). Da mesma forma, a computação em grade oferece acesso consistente e pervasivo a recursos distribuídos, independente da sua localização, *hardware*, sistema operacional e demais tecnologias utilizadas.

Originalmente, o desenvolvimento das grades computacionais foi motivado por requisitos de comunidades de usuários profissionais, que necessitavam de acesso a recursos como grandes conjuntos de computadores que eram utilizados em simulações e análise de dados em larga escala.

Atualmente, as tecnologias de grade vêm se espalhando amplamente e sendo adotadas por outras comunidades, além da comunidade científica (Mungioli, 2005). A disseminação desta tecnologia possibilitou o surgimento de projetos com diferentes propósitos, variando no tamanho, escopo, duração, estrutura, comunidade e sociologia (Foster *et al.*, 2001).

Apesar das diferenças existentes entre grades computacionais, é possível identificar duas

entidades que estão em constante interação: produtores e consumidores. Os produtores são os donos dos recursos compartilhados, enquanto os consumidores são os usuários de tais recursos (Santos, 2005).

A computação em grade supõe que muitos destes recursos estão ociosos na maior parte do tempo e podem ser compartilhados através de redes de interconexão e uma camada de *software* (*middleware*) que oferece serviços básicos para os participantes. Entre os recursos compartilhados em uma grade computacional podem ser citados poder computacional, espaço para armazenamento, sensores, aplicações e dados.

O conjunto de indivíduos e/ou instituições com tais perfis somados a regras de compartilhamento e acesso bem definidas é chamado de organização virtual (*virtual organizations - VO*) (Foster *et al.*, 2001). O propósito de uma organização virtual é altamente variável, além de possuir como característica marcante a resolução de problemas baseada na colaboração, computação e ambientes ricos em dados.

O desenvolvimento das grades computacionais é realizado atualmente sob forte influência de duas organizações, *Global Grid Forum* e *The Globus Alliance*. A primeira está ligada a comunidades acadêmicas e comerciais, tendo como objetivo a criação de padrões e documentação de boas práticas para o desenvolvimento das grades. Já a segunda, exerce sua influência através do *Globus Toolkit* (Foster & Kesselman, 1997), o qual é utilizado no desenvolvimento de diversos *middlewares* para computação em grade.

2.3.3 Peer-to-Peer

Os sistemas *P2P* representam um paradigma para a construção de sistemas e de aplicativos distribuídos onde dados e recursos computacionais são provenientes da colaboração uniforme de diversos *hosts* através da Internet.

Tais sistemas são desenvolvidos para permitir o compartilhamento de recursos em larga escala e sem a necessidade de servidores e infra-estruturas associadas para gerenciamento.

Seu sucesso é conseqüência do rápido crescimento da Internet e aumento no número de usuários exigindo acesso a recursos compartilhados. A popularidade destes sistemas ocorreu, principalmente, devido às aplicações utilizadas para compartilhamento de arquivos. Maiores detalhes sobre esta tecnologia são apresentados na seção 3 deste trabalho.

2.3.4 Agentes Móveis

Um agente móvel é um programa que se move entre computadores em um ambiente de rede. O objetivo destes programas é realizar uma tarefa específica, podendo fazer uso de recursos da localidade visitada, e retornar com os resultados para a sua origem.

O uso de agentes móveis reduz custo e tempo de comunicação, quando comparado com um cliente estático que solicita acesso a recursos através de requisições remotas, já que tais requisições podem originar a transferência de grandes volumes de dados.

Um agente móvel pode ser utilizado, por exemplo, para comparar preços de produtos de diversos fornecedores, através da visita em cada site e execução de diversas operações de consulta.

O uso de agentes móveis requer grande atenção em aspectos relacionados à segurança dos recursos existentes nos computadores visitados e conclusão das tarefas programadas. Maiores detalhes podem ser encontrados em (Lu, 2004).

2.4 Acesso a Dados em Sistemas Distribuídos

Diversas são as tecnologias utilizadas atualmente para o compartilhamento de dados em sistemas distribuídos, como os sistemas de arquivos distribuídos *(NPFS, PVFS, GRID, FTP)*, bibliotecas para troca de mensagens *(MPI, PVM)* e acesso transparente à memória compartilhada *(DSM)*. A seguir, são apresentadas algumas destas tecnologias.

2.4.1 Serviços Web

Segundo Liu & Deters (2007), os serviços *web* representam a mais popular tecnologia utilizada em sistemas cuja arquitetura é orientada a serviços. Atualmente, o desenvolvimento desta tecnologia é feito sob responsabilidade da *W3C* e *OASIS* , contando ainda com o apoio de empresas como *IBM* e *Microsoft* . Além da padronização, a alta disponibilidade de ferramentas e infra-estruturas para a criação de serviços *web* são fatores determinantes para sua popularidade.

A arquitetura de comunicação dos serviços *web* é composta por três componentes:

- ***Consumidor:*** denota a entidade que deseja utilizar os serviços *web*;
- ***Transporte:*** define o meio de comunicação que o consumidor utiliza enquanto interage com o serviço requisitado;
- ***Fornecedor:*** entidade que fornece os serviços *web*.

Toda a comunicação entre os consumidores e fornecedores é feita através dos padrões *XML* (Vaughan-Nichols, 2003) e *SOAP* (Yeo *et al.*, 2006), ou seja, os dados são transferidos no formato *XML* e encapsulados pelo protocolo *SOAP* (Figura 2.3). O protocolo de transporte utilizado não é determinado pela tecnologia, podendo ser utilizado, por exemplo, o *HTTP* .

Uma vez que os serviços *web* estão criados e está definida a forma de comunicação, é necessário definir mecanismos para a publicação e busca dos serviços. Para tanto é utilizada uma

linguagem de descrição de serviços *web* (Tsai *et al.*, 2002) e o *UDDI* (Curbera *et al.*, 2002), como meio de publicação e busca de descrições dos serviços. Maiores detalhes sobre esta tecnologia podem ser encontrados em (Fan & Kambhampati, 2005).

```
<SOAP-ENV: Envelope xmlns:
SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV: encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <SOAP-ENV:Header>
        <t:Transaction-ID xmlns:t="some-URI">
                552511951722
        </t:Transaction-ID>
      </SOAP-ENV:Header>
      <SOAP-ENV:Body>
        <m:RemoteFunction xmlns:m="some-URI">
        <Parameter1>123</Parameter1>
        </m: RemoteFunction>
      </SOAP-ENV:Body>
</SOAP-Envelope>
```

Figura 2.3: Anatomia de um envelope SOAP (Platzer, 2004).

2.5 Sistemas de Arquivos Distribuídos

O objetivo dos Sistemas de Arquivos Distribuídos (SAD) é possibilitar o compartilhamento de diretórios de forma transparente através da rede, permitindo a execução de aplicações e sistemas operacionais sem a necessidade de unidade de armazenamento local.

Segundo Aguiar *et al.* (2008), a maior vantagem deste modelo é permitir que os dados sejam requisitados somente quando necessários, podendo ser compartilhados por diversos computadores simultaneamente. Além disso, pode ser obtida redução de custos uma vez que os computadores que acessam as informações pela rede não necessitam de disco rígido e leitores de mídia.

Alguns dos sistemas de arquivos mais pesquisados são *NFS*, *AFS* e *CODA* (Aguiar *et al.*, 2008).

2.6 Sistemas de Arquivos Paralelos

A adoção dos Sistemas de Arquivos Paralelos (SAP) (Aguiar *et al.*, 2008; Dodonov, 2004) está associada principalmente ao fato das tecnologias dos discos de armazenamento não terem evoluído na mesma velocidade que os processadores e tecnologias de rede evoluíram. Assim, os acessos ao disco rígido passaram a ser gargalos nos sistemas, impedindo que aplicações atingissem uso pleno dos processadores.

Com os *SAPs*, é possível fazer acesso a dados na velocidade da rede local, superando o gargalo anteriormente citado. Dentre os *SAPs* mais utilizados, podem ser destacados *PVFS*, *NFSP* e *pNFS* (Aguiar *et al.*, 2008; Dodonov, 2004).

2.7 Memória Distribuída Compartilhada

Memória Distribuída Compartilhada (*Distributed Shared Memory - DSM*) refere-se a uma classe de implementação de *software* e *hardware* utilizada para permitir que um computador tenha acesso a um espaço de memória superior ao seu espaço local, de forma transparente. Assim, elementos inseridos neste ambiente podem fazer acesso a endereços de memória sem conhecer a localização física da mesma.

Para isso, diversas técnicas podem ser utilizadas, sejam baseadas em *hardware* (Amza *et al.*, 1996), *software* (*Munin* (Bennett *et al.*, 1990), *Controller* (Ramachandran *et al.*, 1989), *Shasta* (Scales *et al.*, 1996)), e abordagens híbridas).

2.8 Considerações Finais

Neste capítulo foram apresentadas as motivações para o desenvolvimento dos sistemas distribuídos, assim como suas principais características e principais modelos existentes.

Abordagens utilizadas pelos diversos sistemas distribuídos foram apresentadas, abrangendo a área de *clusters*, grades computacionais, serviços web, agentes móveis e sistemas *P2P*. Além disso, foram apresentadas algumas técnicas de acesso a dados em sistemas distribuídos.

Com a evolução de ambientes distribuídos, a necessidade de um meio de compartilhamento de dados entre diversos elementos de processamento independentes tornou-se evidente, o que motivou diversas pesquisas. Essas pesquisas resultaram na criação de protocolos de comunicação *P2P* específicos, caracterizados por diferentes arquiteturas, os quais são apresentados no próximo capítulo.

CAPÍTULO

3

Sistemas Peer-to-Peer

3.1 Considerações Preliminares

Atualmente, não existe um consenso na literatura sobre a definição ou caracterização dos sistemas *P2P*. Na sua forma mais simples, o termo refere-se à interação entre duas ou mais entidades. Desta forma, o termo pode ser observado em sistemas que vão além das redes de comunicação, como por exemplo, uma conversa entre duas pessoas (Hu, 2005).

A definição de um sistema *P2P* puro refere-se a um sistema totalmente distribuído, onde todos os participantes possuem o mesmo papel, atuando como clientes quando acessam informações; quando disponibilizam informações, são servidores; e quando estão repassando informações, são roteadores. Porém, percebe-se que esta definição não engloba sistemas que são largamente aceitos como *P2P*, no entando, fazem uso de servidores centrais na execução de determinadas tarefas, como por exemplo, indexar o conteúdo compartilhado (Francesquini, 2007; Kubiatowicz, 2003; Androutsellis-Theotokis & Spinellis, 2004).

Outras definições referem-se à *P2P* como uma classe de aplicações que aproveitam de recursos (ciclos, armazenamento, conteúdos, presença humana) disponíveis na borda da Internet (Tsai *et al.*, 2002). Esta definição é mais abrangente e engloba também aqueles sistemas que, de alguma forma, apoiam-se em servidores centrais para a realização de tarefas específicas, além das grades computacionais, como o Seti@home.

Apesar das diversas definições baseadas no modelo de interconexão utilizado, percebe-se que muitos sistemas são rotulados como *P2P* não necessariamente pela sua arquitetura ou operações internas, mas ao invés disso, pela percepção externa que se tem do mesmo, isto é, pela impressão que ele fornece de estar conectando computadores diretamente. Desta forma, diversas definições de *P2P* são aplicadas para acomodar os diferentes sistemas e aplicações *P2P*

existentes (Androutsellis-Theotokis & Spinellis, 2004).

Em (Androutsellis-Theotokis & Spinellis, 2004) são apresentadas duas características que definem os sistemas *P2P*:

- Compartilhamento de recursos computacionais de forma direta entre as partes envolvidas, sem intermédio de servidores centrais, no entanto, aceita-se a existência de servidores centrais para a execução de tarefas específicas, como a localização de *peers* e autenticação de usuários;
- Habilidade para tratar instabilidade e variação de conectividade, adaptando-se de forma automática a falhas nas conexões de rede e computadores, bem como comportamento transitório dos *peers*.

Com base nas características apresentadas, chega-se a seguinte definição (Androutsellis-Theotokis & Spinellis, 2004):

Sistemas *P2P* são sistemas distribuídos, constituídos por *peers* interconectados, habilitados para a auto-organização em uma topologia de rede, com o propósito de compartilhar recursos como conteúdo, ciclos de *CPU*, armazenamento e largura de banda, com capacidade de se adaptar a falhas e acomodar-se à variação no número de *peers* participantes, mantendo aceitável conectividade e desempenho, sem a necessidade de intermediação ou suporte de um servidor global centralizado.

3.1.1 Características

A arquitetura *P2P* incorpora um dos conceitos chave da Internet, o conceito de *Host Software*, cuja descrição está na primeira *RFC* (*Request for Comments*), de 1969. Atualmente, características de tal conceito podem ser encontradas principalmente na criação de protocolos da camada de transporte, como o *TCP/IP*, onde não há distinção entre as partes envolvidas, nem ponto central para controle da comunicação.

Na camada de aplicação, grande parte dos protocolos utilizados, atualmente, é baseada na arquitetura cliente/servidor (por exemplo, *Web* e *e-mail*). Assim, pode-se dizer que o aumento na popularidade de aplicações *P2P* é uma volta aos princípios que nortearam o desenvolvimento da Internet (Francesquini, 2007).

A seguir, são apresentadas as principais características das redes *P2P*. Além de possibilitar a classificação das mesmas, tais características são utilizadas também em estudos comparativos (Hu, 2005).

Descentralização

O modelo de armazenamento e processamento de dados somente em servidores centralizados e o acesso a conteúdos feito a partir de protocolos de requisição/resposta são questionados pelo modelo *P2P*. Apesar de o modelo centralizado ter a seu favor a diminuição no custo do *hardware* e aumento na capacidade do mesmo, tal modelo traz desvantagens como o custo de manutenção e o surgimento de gargalos no sistema.

Uma das mais poderosas idéias relacionadas à descentralização é a posse de recursos por parte dos usuários. Em sistemas totalmente descentralizados, cada nó é um participante igual aos outros. Porém, tal característica pode trazer desvantagens, como a complexidade na implementação de tais sistemas.

Devido ao aumento na complexidade de implementação e gerenciamento dos sistemas totalmente descentralizados, alguns sistemas *P2P* acabam utilizando uma solução híbrida, em que existe um servidor com informações globais, porém, a troca de informações é realizada diretamente entre os nós participantes.

Escalabilidade

Na arquitetura cliente/servidor, os servidores são os únicos responsáveis por toda a carga do sistema, o que lhes permite ficar sobrecarregados em horários de pico e o sistema como um todo acaba oferecendo serviço com baixa qualidade.

Nos sistemas *P2P*, conforme aumenta o número de clientes aumenta também o número de servidores, uma vez que todos os participantes podem atuar como clientes e servidores simultaneamente. Desta forma, os recursos compartilhados aumentam na mesma proporção do aumento da demanda pelos mesmos, o que possibilita ao sistema crescer sem sem comprometer sua capacidade de atender a qualidade de serviço oferecida.

Auto-organização

Auto-organização está relacionada com a capacidade de um sistema espontaneamente se manter organizado sem a necessidade de controle e/ou interação de um sistema externo. Sua importância nos sistemas *P2P* é conseqüência de características como a escalabilidade, necessidade de recuperação de falhas e conexões intermitentes.

O aumento no número de computadores, usuários e o aumento de carga em um sistema *P2P* podem ocorrer de maneira imprevisível. Assim, um sistema centralizado com tal característica necessitaria de reconfigurações freqüentes.

As conexões intermitentes dos sistemas *P2P* fazem com que qualquer configuração pré-definida não permaneça intacta por um longo período de tempo. Adaptações no sistema

são necessárias sempre que novos participantes se conectam ou se desconectam do sistema. Por fim, devido ao custo de se manter um equipamento ou pessoa dedicada ou gerenciamento do sistema, tal tarefa é distribuída entre os próprios *peers* do sistema.

3.2 Redes de Sobreposição

Segundo Androutsellis-Theotokis & Spinellis (2004), a operação dos sistemas *P2P* é baseada nas redes de sobreposição (*overlays networks*). Estas, são definidas como redes lógicas criadas sobre uma rede física, ou seja, uma rede com uma topologia virtual (Figura 3.1).

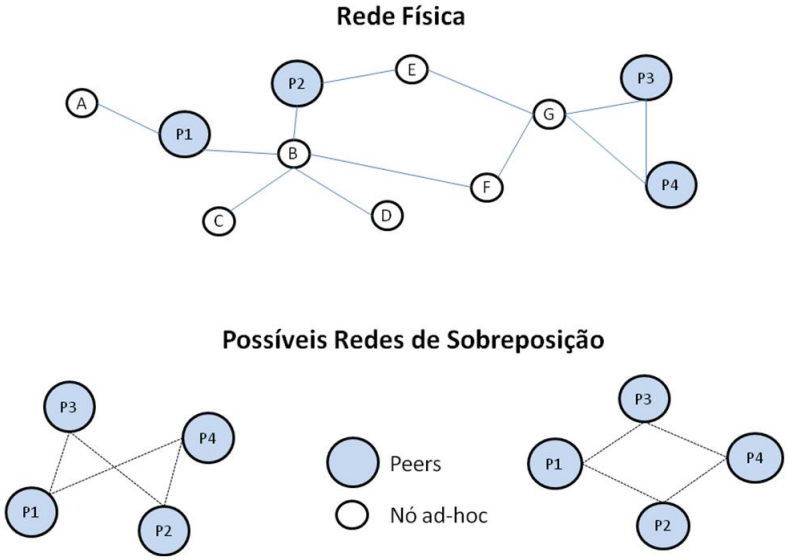


Figura 3.1: Redes de sobreposição.

Tais redes podem ser classificados de acordo com a estrutura e topologia da rede de sobreposição utilizada, conforme segue:

Sistemas Não-estruturados

O armazenamento do conteúdo é feito de forma não relacionada à topologia da rede de sobreposição. Os mecanismos de busca utilizados variam de métodos que utilizam a força bruta (por exemplo, a propagação de consultas pela rede) a métodos mais sofisticados que utilizam estratégias como caminhos randômicos e índices de rota.

Sistemas Estruturados

As redes de sobreposição estruturadas foram criadas com o foco principal em escalabilidade. Tais sistemas possuem topologia fortemente controlada e os recursos (ou ponteiros para os mesmos) são armazenados em localizações precisamente especificadas. Um mapeamento entre conteúdo e localização é oferecido pelos sistemas estruturados para que as consultas possam ser

feitas, sem nós intermediários, aos *peers* que possuem os mesmos. Nos dias atuais, esta classe de sistemas é baseada principalmente no uso de tabelas *hash* distribuídas (*Distributed Hash Tables - DHT*).

3.3 Tabelas *Hash* Distribuídas

DHTs constituem uma classe de sistemas distribuídos que oferece serviço de busca similar as tabelas *hash*, ou seja, utilizando pares $[x, v]$. Porém, diferentemente das tabelas *hash* tradicionais, nas *DHTs*, a responsabilidade pela manutenção das informações e mapeamento de nomes para valores é distribuída entre os nós que as constituem.

3.3.1 Características

As principais características das *DHTs* são:

- ***Descentralização:*** não existe nenhum tipo de coordenação central nas *DHTs*. Assim, a responsabilidade pela manutenção da mesma, assim como o armazenamento das informações disponibilizadas é dividida entre todos os nós;
- ***Escalabilidade:*** devido a capacidade de auto-organização, o funcionamento do sistema não é prejudicado pelo aumento no número de participantes;
- ***Tolerância a falhas:*** o funcionamento do sistema não é comprometido pela constante entrada/saída de nós ou falhas nos mesmos, ou seja, a falha em um dos participantes não compromete o funcionamento do sistema como um todo.

3.3.2 Implementações

Dentre as diversas implementações de *DHT*, podem ser citadas:

Chord

A rede *Chord* (Stoica *et al.*, 2001) oferece alto desempenho na busca de itens. Dentre os objetivos do *Chord*, estão escalabilidade, disponibilidade, balanceamento de carga e flexibilidade.

Nela, cada item possui um identificador de *m-bits* que é criado a partir de uma função *hash* do tipo *SHA*. Esta função é aplicada no endereço *IP* de um nó participante ou no identificador (*nome*, *URL*) do item compartilhado. Posteriormente, os nós são organizados em seqüência, utilizando um modelo de anel conhecido como *Chord Ring* (Figura 3.2).

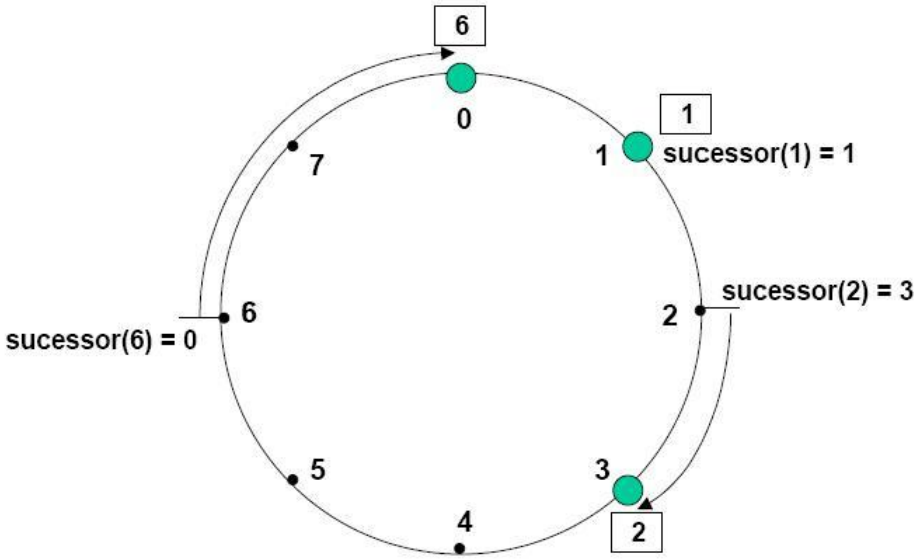


Figura 3.2: Uma rede Chord constituída por três nós (0, 1 e 2). Nesta figura, a chave 1 é localizada no nó 1, chave 2 no nó 3 e a chave 6 no nó 0 (Stoica *et al.*, 2001).

A operação básica do *Chord* é da seguinte forma: dado uma chave, mapeá-la para um nó da rede de acordo com o seu valor. Todos os nós do *Chord Ring* são responsáveis pelos itens cujo valor da chave é menor que o valor de sua chave e maior do que o valor da chave de seu predecessor.

No processo de localização de uma chave, o nó solicitante necessita conhecer apenas seu sucessor. Assim, numa pesquisa, ele faz o pedido a seu sucessor e caso este não possa atendê-la, encaminha novamente o pedido para o próximo sucessor.

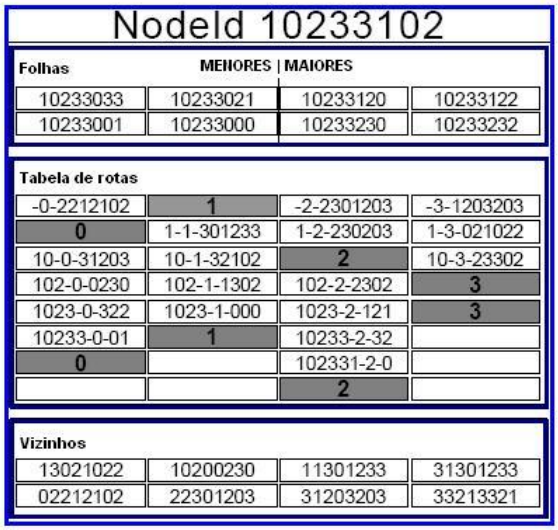
Pastry

No *Patry* (Rowstron & Druschel, 2001), cada nó da rede recebe um identificador de 128 *bits* que é gerado a partir da aplicação de uma função *hash* ao seu *IP* ou a sua chave pública. Este identificador é utilizado para indicar a posição do nó em um espaço de chave circular com faixa de 0 a 2^{128-1}.

Quando um recurso é inserido na rede, o mesmo é mapeado para o nó cujo identificador está, numéricamente, mais próximo de sua identificação numérica. Desta forma, as mensagens no *Pastry* são roteadas para o nó cujo identificador está mais próximo da identificação do recurso procurado. A cada etapa de roteamento, um nó encaminha as mensagens para outro nó cujo identificador compartilhe com a chave pelo menos um dígito a mais do que é compartilhado com o nó atual. Caso nenhum nó seja conhecido, a mensagem é encaminhada para um nó cujo identificador compartilhe um prefixo com a chave procurada e está, numericamente, mais próximo da chave do que o nó atual.

Para suportar tais procedimentos, cada nó mantém uma tabela de roteamento, infor-

mações sobre um conjunto de vizinhanças e um conjunto de folhas. O conjunto de vizinhança contém os nós que possuem identificadores numéricos mais próximos do nó atual. As folhas representam os nós sucessores e predecessores do nó atual. A figura 3.3 apresenta o estado de um nó *Patry* hipotético em dado momento.



Nodeld 10233102

Folhas	MENORES | MAIORES		
10233033	10233021	10233120	10233122
10233001	10233000	10233230	10233232

Tabela de rotas

-0-2212102	1	-2-2301203	-3-1203203
0	1-1-301233	1-2-230203	1-3-021022
10-0-31203	10-1-32102	2	10-3-23302
102-0-0230	102-1-1302	102-2-2302	3
1023-0-322	1023-1-000	1023-2-121	3
10233-0-01	1	10233-2-32	
0		102331-2-0	
		2	

Vizinhos

13021022	10200230	11301233	31301233
02212102	22301203	31203203	33213321

Figura 3.3: Estado de um nó Pastry hipotético, cujo ID é 10233102 (Rowstron & Druschel, 2001).

Além disso, o *Pastry* oferece procedimentos para lidar com a entrada e saída de nós na rede, além de aplicar uma heurística que garante que as entradas na tabela de roteamento são escolhidas de forma que o processo de localização seja otimizado.

Tapestry

O *Tapestry* (Zhao *et al.*, 2004) possui muitas similaridades em relação ao *Pastry*. Entre as similaridades estão o uso de prefixo e sufixo para roteamento, o algoritmo de entrada e saída dos nós e o custo de armazenamento do conteúdo.

Entre as diferenças, está a inexistência dos conjuntos de folhas e vizinhos. Desta forma, quando a tabela de roteamento de um nó não possui uma entrada com sufixo comum a chave, a mensagem é encaminhada para o nó cujo identificador numérico é mais próximo da chave do que do nó atual.

OpenDHT

OpenDHT (também conhecida como OpenHash) (Rhea *et al.*, 2005) é um projeto desenvolvido com o objetivo de criar uma *DHT* de acesso público, em forma de serviço. Desta forma, pode-se facilitar o desenvolvimento de aplicações distribuídas sem a necessidade de se preocupar com detalhes de implementação e manutenção de tabelas *hash* distribuídas.

Através de uma interface simples, esta rede oferece operações como inserção, recuperação e remoção de entradas do tipo $[x, v]$ e todos os demais recursos oferecidos por uma *DHT*.

A Figura 3.4 exibe a arquitetura da *OpenDHT*. Nela são encontrados dois tipos de nós: Os nós de infra-estrutura executam o código servidor, ou seja, o código de implementação da *DHT*. Os nós clientes são aqueles que executam aplicações que invocam os serviços da *DHT* através de *RPC*. Além de colaborar com a tarefa de roteamento e espaço de armazenamento, cada nó da *OpenDHT* também atua como um *gateway*, aceitando *RPCs* dos nós clientes.

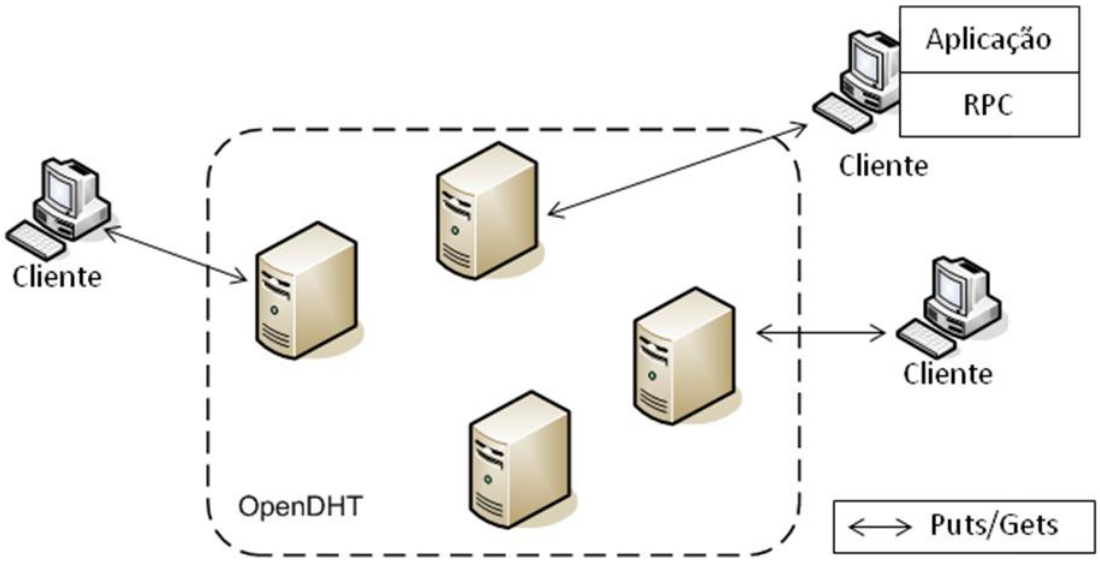


Figura 3.4: Arquitetura da OpenDHT.

Sua implementação é atualmente baseada na *DHT Bamboo* e na rede *PlanetLab*.

3.4 Classificação dos Sistemas P2P

Os sistemas P2P podem ser classificados de diversas formas. Num primeiro modelo, leva-se em consideração a forma como os índices dos objetos compartilhados são armazenados. Assim, tem-se:

- ***Modelo Centralizado:*** mantém um índice global com as informações referentes aos objetos que são compartilhados. Desta forma, ele é acessado diretamente pelos clientes que desejam determinado objeto. Um exemplo de aplicação deste tipo é o *Napster*;
- ***Modelo Descentralizado:*** não mantém nenhum tipo de índice global, por exemplo, *Gnutella*;
- ***Modelo Hierárquico:*** faz uso de nós com características especiais, os super nós, para a execução de tarefas como a manutenção de índices, como ocorre no *Kazaa*. Este modelo pode ser visto como um intermediário entre os dois modelos anteriores.

Também, é possível classificar estes sistemas de acordo com os mecanismos utilizados para efetuar buscas por objetos. Nesta classificação, são observados os seguintes modelos:

- ***Serviço de Localização Centralizada (Centralized Service Location - CSL):*** caracterizado pelas buscas em servidores centrais de índices, como ocorre no *Napster*;
- ***Serviço de Localização por Inundação (Flooding-based Service Location - FSL):*** baseado no reenvio de requisições entre nós que possuem comportamentos idênticos, por exemplo, *Gnutella*. O termo inundação faz referência ao número de mensagens que são enviadas a rede até que o nó desejado seja encontrado;
- ***Serviço de Localização baseado em* DHT *(Distributed Hash Table-based Service Location - DHT):*** utiliza um mecanismo de busca semelhante às tabelas *hash* tradicionais, porém, mantém partes desta tabela em diversos nós que compõem a rede.

Outra classificação divide estes sistemas em:

- ***Modelo Centralizado:*** idêntico ao modelo centralizado apresentado na primeira classificação e possui mecanismo de busca igual ao modelo *CSL* da segunda classificação;
- ***Modelo Descentralizado e Estruturado:*** modelo classificado como descentralizado, segundo a primeira classificação apresentada, e possui mecanismos de buscas baseados em *DHT*;
- ***Modelo Descentralizado e Não Estruturado:*** modelo classificado como descentralizado ou hierárquico segundo a primeira classificação apresentada, fazendo uso de mecanismos de buscas *FSL*, conforme a segunda classificação apresentada.

Em (Laffranchi, 2004), o modelo de comunicação *P2P* é classificado da seguinte forma:

- ***Modelo Descentralizado:*** também chamado de *P2P* puro, trabalha sem nenhum servidor ou repositório central. Neste modelo, todos os *peers* são auto-suficientes, atuando como cliente e servidor.

  Para seu funcionamento, os *peers* fazem solicitações aos *peers* vizinhos que estão ao seu alcance, no exemplo exibido na Figura 3.5, "A"se conecta nas máquinas com as quais tem uma conexão. Por sua vez, estas se conectam a outros *peers* com os quais mantém conexão e repetem a solicitação inicial. Tal procedimento é realizado repetidamente até que o recurso procurado seja encontrado, no caso, em "B". Quando o recurso é encontrado, uma conexão é criada entre o nó onde o recurso foi encontrado e o nó que fez a requisição, executando-se assim a transferência do recurso solicitado sem a existência de nenhum intermediário (Laffranchi, 2004).

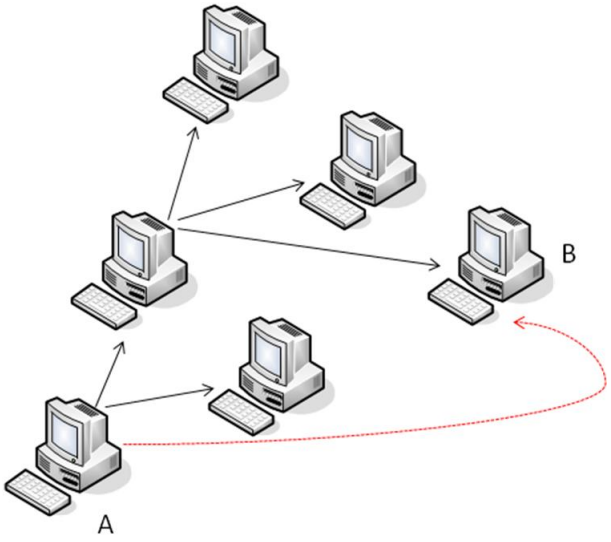


Figura 3.5: Arquitetura de um sistema P2P totalmente descentralizado.

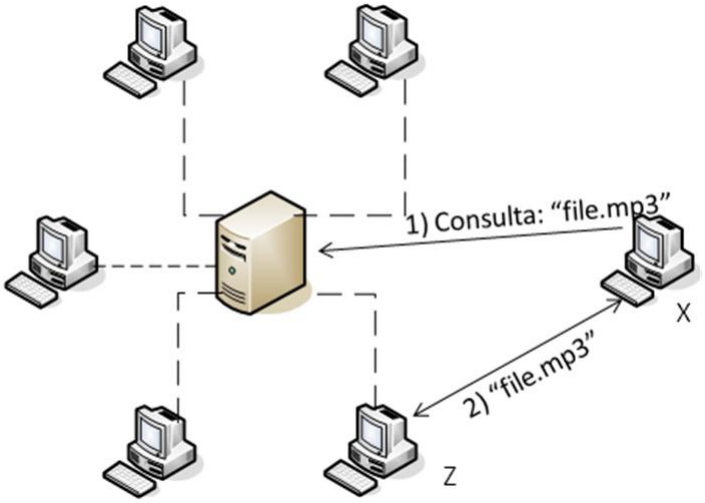


Figura 3.6: Arquitetura P2P híbrida.

- ***Modelo Híbrido:*** o modelo descentralizado é raramente utilizado devido a sua complexidade. Desta forma, são utilizados servidores para a execução de tarefas específicas, como autenticação de usuários, serviço de diretório e mapeamento de recursos disponíveis, obtendo-se assim o modelo híbrido.

  Conforme mostra a Figura 3.6, "X"contata um servidor para saber a localização de determinado recurso. A seguir, a informação obtida a partir do servidor é utilizada para que "X"se conecte diretamente ao nó que oferece tal recurso, no caso, "Z". Assim, a transação é efetuada entre "X"e "Z".

3.5 Tipos de Aplicações P2P

Os três grandes tipos de aplicações P2P são:

- *Computação Distribuída;*
- *Compartilhamento de Arquivos;*
- *Colaboração peer-to-peer.*

A seguir, os tipos de aplicações apresentados são descritos (Laffranchi, 2004; Androutsellis-Theotokis & Spinellis, 2004).

3.5.1 Computação Distribuída

Nesta categoria, estão aplicações que buscam aproveitar o poder de processamento disponível em outros *peers*.

Inicialmente, um *software* servidor divide as tarefas em pequenas partes, que, a seguir, são divididas entre computadores que estão conectados ao sistema através da Internet e utilizam determinado *software* cliente. Depois que uma tarefa é executada, o resultado da mesma é enviado para o servidor e uma nova tarefa é enviada ao cliente.

Uma das aplicações mais conhecidas desta categoria é o projeto *SETI@home* (Anderson *et al.*, 2002). Este projeto é realizado em conjunto entre a *Universidade de Berkley* e a *NASA* , tendo seu início em 1997. Ele é constituído por uma coleção de projetos de pesquisas científicas voltadas para a descoberta de civilizações extraterrestres. Nele, *PCs* clientes utilizam um protetor de tela que carrega os dados do servidor do *SETI@home* e, então, processam e enviam os resultados de volta ao servidor quando possuem capacidade de processamento ociosa.

Tal categoria apresenta como uma de suas maiores limitações a necessidade de os trabalhos processados poderem ser divididos em pequenas partes independentes.

3.5.2 Compartilhamento de Arquivos

Atualmente, a maioria dos sistemas *P2P* é enquadrada nesta categoria, a qual é a grande responsável pelo aumento da popularidade desta tecnologia, a partir do lançamento do *Napster* em 1999.

A idéia por trás desta categoria de aplicações é clara: criar pontos de expansão para armazenamento remoto, para substituir o disco rígido local e permitir a interação entre vários pares da rede para obtenção e disponibilização de arquivos.

Atualmente, existem três principais modelos de aplicações *P2P* desta categoria:

- ***Modelo Centralizado:*** Aplicações que utilizam servidores centrais para a execução de tarefas como autenticação de usuário e localização de arquivos. Um exemplo de aplicação deste modelo é o *Napster*;

- ***Modelo de busca por inundação:*** Aplicações totalmente descentralizadas que fazem uso de reenvio de solicitações entre *peers* no processo de busca por arquivos. Tal modelo pode ser verificado em sistemas como o *Gnutella*;
- ***Modelo de roteamento de documentos:*** Este modelo de aplicações faz uso principalmente das *DHTs* para o roteamento de mensagens de solicitação de arquivos. Como exemplo de aplicação deste modelo pode ser citado o *FreeNet* (Clarke *et al.*, 2001)

Diversas questões são levantadas quando se trata do compartilhamento de arquivos *P2P*, dentre elas, largura de banda, segurança e eficiência dos processos de busca. Exemplos destes sistemas são apresentados no decorrer deste trabalho.

3.6 Colaboração P2P

Esta categoria inclui sistemas que oferecem infra-estrutura necessária para a colaboração e comunicação direta entre computadores e usuários. Essas aplicações incluem *chats*, mensagens instantâneas, jogos e aplicações compartilhadas.

As aplicações colaborativas permitem que usuários criem grupos e iniciem uma determinada tarefa, podendo ser os grupos formados por apenas um par de usuários ou um número maior, com participantes localizados ao redor do planeta. Uma das principais características destas aplicações é sua orientação a eventos. Desta forma, quando uma mudança ocorre em uma *peer*, um evento é gerado e enviado ao resto do grupo. Na camada de aplicação, cada interface de um *peer* é atualizada de acordo com cada aplicação.

Dentre as aplicações desta categoria, pode ser citado como exemplo o *Groove*. Tal aplicação habilita espaços compartilhados para o trabalho colaborativo, incluindo ferramentas para interação de grupos, compartilhamento de conteúdo, comunicação e atividades em comum em tempo real.

3.7 Estado Atual

A seguir, são apresentadas as principais características de algumas aplicações de compartilhamento de arquivos *P2P* hoje utilizadas.

Napster

O Napster foi lançado em 1999 e foi o aplicativo pioneiro em compartilhamento de arquivos *peer-to-peer*. Foi através do compartilhamento de musicas com o Napster que o compartilhamento de dados utilizando sistemas *peer-to-peer* alcançou atenção de pesquisadores e usuários.

Como principal característica do Napster, é possível citar a utilização de servidores centrais dedicados, os quais são utilizados pelo mecanismo de autenticação de usuários e para localizar arquivos.

Inicialmente, o cliente acessa um servidor onde são armazenadas informações referentes a localização dos arquivos compartilhados. A seguir, é criada uma conexão direta entre cliente e o nó que possui o arquivo procurado, e, assim, é realizada a transferência do arquivo.

Gnutella

Gnutella representa a primeira família de aplicações *peer-to-peer* totalmente descentralizadas e é caracterizada pelo acesso anônimo aos arquivos compartilhados e uso de mecanismos de busca por inundação, ou seja, quando uma requisição é realizada, todos os nós são consultados e somente o nó detentor da informação retorna a consulta. Uma rede Gnutella é formada por nós conectados através do protocolo *TCP/IP*, sendo que cada nó atua como cliente (quando faz requisições por arquivos), servidor (quando fornece arquivos), e como roteador (quando reenvia consultas e respostas).

A figura 3.7 apresenta o mecanismo de busca utilizado pelo *Guntella*. Este mecanismo é chamado de busca por inundação e tem seu funcionamento baseado na propagação de mensagens pelos nós participantes da rede. Conforme mostra a figura 3.7, em (1), um dos nós envia uma mensagem de consulta para os nós com os quais mantêm uma conexão e esta mensagem é enviada, sucessivamente, para os demais nós da rede, até que o recurso procurado seja encontrado. A seguir (2), uma mensagem com a localização deste recurso é propagada novamente pela rede, mas, desta vez em sentido contrário, buscando alcançar o nó que procura o recurso. Finalmente (3), uma conexão é criada diretamente entre o nó que fez a requisição e aquele que possui o recurso procurado, dando origem a uma conexão direta, pela qual é realizado o *download*.

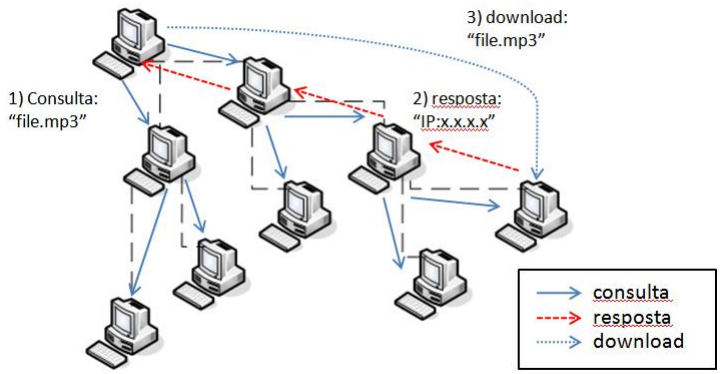


Figura 3.7: Busca por inundação.

Kazaa

Dentre as redes de compartilhamento de arquivos *P2P*, *Kazaa* (Good & Krekelberg, 2003) é sem dúvida a mais popular, tanto quando se fala em número de usuários (mais de 4 milhões até a primeira metade da presente década), assim como quando se fala em volume de arquivos trocados entre os seus participantes.

Estudos mostram que, nos Estados Unidos, 76% do tráfego oriundos do compartilhamento de arquivos *P2P* sejam de responsabilidade do *Kazaa*, enquanto apenas 8% seja pertencente a rede Gnutella (Ripeanu, 2001).

Assim como *Gnutella*, a rede *Kazaa* não faz uso de servidores dedicados à localização do conteúdo compartilhado, porém, diferentemente da *Gnutella*, onde originalmente todos os nós eram iguais, no *Kazaa* é utilizada uma arquitetura hierarquica, composta por dois tipos de nós, os nós comuns (ON) e super nós (SN) (Figura 3.8).

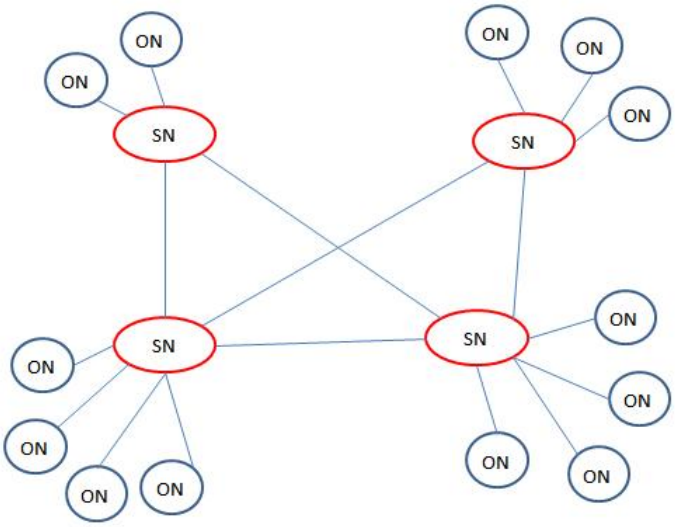


Figura 3.8: Nós comuns e super nós na rede Kazaa.

Desta forma, tem-se uma rede composta por diversos SNs que atuam como "mini servidores", utilizados para armazenar informações sobre os arquivos compartilhados pelos NCs com os quais mantém conexão. Além disso, os SNs mantêm conexões com outros SNs, criando uma rede de sobreposição que permite maior exploração da rede criada.

Quando um usuário decide participar da rede *Kazaa*, é necessário que ele se conecte à um dos SNs da rede e informe ao mesmo os metadados dos arquivos que pretende compartilhar. Para realizar uma busca, é necessário que o cliente envie uma *query* ao seu SN, que poderá retornar a localidade do arquivo compartilhado ou repassar a *query* para outro SN com o qual mantenha uma conexão.

Aparentemente, apesar de oferecer acesso a grande parte do conteúdo da rede, estudos

(Liang *et al.*, 2004) mostram que a abordagem utilizada por Kazaa permite que, em dado momento, cada SN esteja conectado apenas à 0,1% do número total de SN da rede, e esses SNs com que mantém conexão são alterados frequentemente, garantindo que a rede esteja atualizada em relação aos nós disponíveis.

BitTorrent

BitTorrent (Yeo *et al.*, 2006) é um protocolo de transferência de arquivos *peer-to-peer* largamente utilizado nos dias atuais. Sua primeira versão foi lançada em Julho de 2001, sendo desenvolvida por Bram Cohen. Atualmente, tal protocolo é mantido pela companhia BitTorrent, Inc.

Um cliente *BitTorrent* é qualquer aplicação que implementa o protocolo BitTorrent. Assim, cada cliente é capaz de preparar, requisitar e transmitir qualquer tipo de arquivo pela rede, utilizando este protocolo.

Cada arquivo compartilhado pelo protocolo *BitTorrent* é dividido em partes iguais (tipicamente entre 64KB e 4MB) e possui um *checksum* associado com cada uma destas partes. Para compartilhar um arquivo ou grupo de arquivos, um nó cria um pequeno arquivo chamado "torrent", por exemplo, "minhaMusica.torrent". Este arquivo é composto por metadados sobre o arquivo compartilhado e sobre o computador que coordena a distribuição do arquivo (*tracker*). Quando um nó quer fazer o *download* de um arquivo, primeiro é necessário obter um arquivo torrent e então se conectar a um *tracker* específico, o qual irá fornecer as informações necessárias para baixar cada parte do arquivo.

Freenet

Freenet é um repositório de dados distribuído desenvolvido por Ian Clarke. Seu objetivo é oferecer compartilhamento de informações *peer-to-peer*, de forma anônima, com forte proteção da privacidade de seus usuários.

Seu funcionamento é baseado no agrupamento de largura de banda e espaço de armazenamento dos nós, permitindo que usuários anônimos possam acessar ou oferecer diversos tipos de informações. O espaço de armazenamento do Freenet é distribuído entre todos os computadores participantes.

O mecanismo de busca utilizado pelo *Freenet* é baseado na criação de uma rede *peer-to-peer* estruturada, que fornece os mecanismos necessários para o roteamento de mensagens baseados em pares $[chave, valor]$, de forma semelhante as tabelas *hash* distribuídas. Desta forma, quando um arquivo é armazenado no *Freenet*, uma chave representando seu identificador é gerada e utilizada para identificar tal arquivo no espaço de armazenamento criado.

3.8 Desafios

Segundo Laffranchi (2004), além do desafio inerente para se ter aplicações comunicando-se de igual para igual, o modelo *P2P* enfrenta outros desafios, tais como segurança, interoperabilidade, controle de acesso, integridade dos dados, sincronização de uma aplicação entre uma ou mais máquinas, assim como controle dos dados distribuídos pela rede. A seguir, é apresentada uma breve descrição destes desafios.

- ***Segurança:*** atualmente, existe uma vasta gama de ameaças aos sistemas de rede. Desta forma, mecanismos de segurança são extremamente importantes. Além disso, os sistemas *P2P* estão se tornando interessantes para o uso comercial, necessitando de técnicas e métodos para autenticação, autorização, disponibilidade, integridade dos dados e confiança;

- ***Interoperabilidade:*** a existência de aplicações que utilizam protocolos e interfaces específicas dificulta a interoperabilidade entre diferentes aplicações ou redes *P2P* existentes. Surge, desta forma, o desafio de encurtar o tempo de desenvolvimento e habilitar as aplicações a serem implementadas facilmente nos sistemas existentes;

- ***Controle de acesso:*** o controle sobre quais usuários acessam o sistema e as informações presentes no mesmo também é um dos desafios de desenvolvimento dos sistemas *P2P*;

- ***Integridade dos dados:*** a natureza distribuída dos sistemas *P2P* possibilita a dependência entre dados distribuídos entre diversos nós da rede. Assim, mostram-se necessários mecanismos que garantam a integridade dos mesmos;

- ***Sincronização:*** dados sendo acessados por diferentes usuários em determinado intervalo de tempo faz das técnicas de sincronização de dados um dos grandes desafios dos sistemas *P2P*;

- ***Distribuição dos dados:*** a distribuição de dados pode ser feita de várias formas, entre elas, a replicação (um alteração de estado é passado para as réplicas) e partição (a aplicação é dividida em várias partes e cada máquina fica encarregada de uma).

3.9 Considerações Finais

Neste capítulo foram apresentados os sistemas *P2P*, mostrando suas principais características e implementações. A utilização dos sistemas *P2P* é prevista neste trabalho para a criação de uma base de índices totalmente distribuída, a qual, será utilizada para a obtenção de informações sobre o conteúdo compartilhado entre diferentes aplicações e sistemas.

Foram apresentadas as tecnologias de redes de sobreposição, tabelas *hash* distribuídas, assim como exemplos de implementações destas tabelas. Também foram abordadas maneiras de classificações destes sistemas, assim como exemplos de aplicações utilizadas para o compartilhamento de arquivos *P2P*.

No contexto deste trabalho, este capítulo é relevante uma vez que apresenta as vantagens e desvantagens da utilização de cada arquitetura *P2P*, assim como os mecanismos de buscas utilizados em cada uma delas, essenciais para o desenvolvimento da base de índices proposta.

CAPÍTULO

4

Modelo Estrutural para Compartilhamento de Arquivos Peer-to-Peer

4.1 Considerações Preliminares

Desde o lançamento do *Napster* em 1999, a popularidade das aplicações para compartilhamento *P2P* vem tendo grande crescimento. De acordo com estudos recentes (Ferguson, 2006), tais aplicações são responsáveis por mais de 60% do tráfego total na Internet (Figura 4.1).

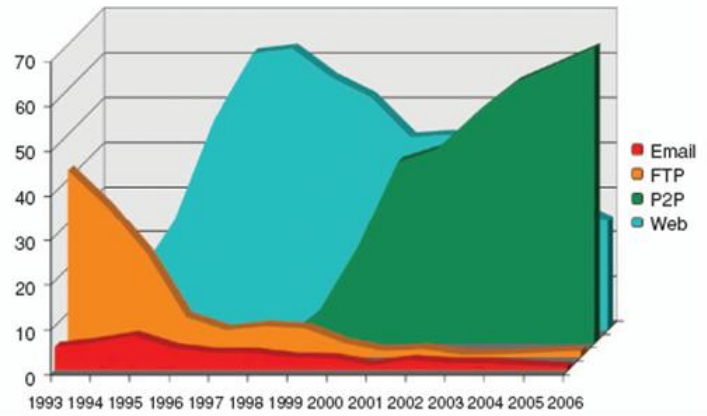


Figura 4.1: Tráfego de dados na Internet (Ferguson, 2006).

Apesar do grande número de aplicações e protocolos existentes, verifica-se que o principal benefício destas aplicações, ou seja, a facilidade de compartilhamento de conteúdo, ainda deixa a desejar, devido ao fato de existir um grande número de repositórios com acesso limitado a aplicações específicas. Desta forma, tem-se uma grande quantidade de objetos compartilhados, porém, estes são encontrados em diferentes redes e estão inacessíveis para diferentes sistemas e aplicações.

A abundante disponibilidade de diferentes protocolos, somada a falta de interoperabilidade entre eles, já motivou trabalhos como apresentado em (Lui & Kwok, 2002), em que foi proposto um modelo de *peer-to-peer gateway*. O principal objetivo de (Lui & Kwok, 2002) foi possibilitar a troca de mensagens entre diferentes protocolos de forma transparente. Inicialmente, foram suportados os protocolos: Gnutella, Freenet e Napster.

Apesar da interoperabilidade alcançada, tal proposta apresentou diversas limitações, como a necessidade de constantes conexões com as redes suportadas, ausência de metadados para identificar similaridades entre os conteúdos compartilhados, estrutura centralizada, falta de mecanismos de buscas e pouca flexibilidade para expansões. Além disso, tal proposta não trata de questões relativas ao compartilhamento dos recursos com outros sistemas, limitando-se aos protocolos previamente definidos.

Em Olmedilla & Palmer (2005) é apresentada uma proposta de interoperabilidade para o sistema Edutella (Nejdl, 2002). Assim como Lui & Kwok (2002), tal proposta também é apoiada em um modelo de *peer-to-peer gateway* e tem por objetivo possibilitar o compartilhamento de arquivos, especificamente, objetos educacionais, entre diferentes redes. Além disso, é apresentado também um modelo de mapeamento semântico, o qual fornece um mecanismo de transformação entre diferentes modelos de dados. Desta forma, é garantida além da troca de mensagens entre diferentes sistemas, a interoperabilidade entre dados que foram descritos seguindo padrões distintos.

Da mesma forma que Lui & Kwok (2002), mais uma vez foi alcançada a interoperabilidade entre diferentes redes, porém, com o acréscimo de um mecanismo de interoperabilidade semântica. Apesar dos avanços obtidos, as mesmas limitações de Lui & Kwok (2002) são encontradas. Além disso, a interoperabilidade semântica apresenta-se restrita a um único domínio de aplicações, aquelas destinadas ao compartilhamento de objetos educacionais.

Desta forma, considerando as características e limitações dos sistemas para compartilhamento de arquivos *P2P* existentes, assim como esforços que já foram dedicados para facilitar o acesso às informações compartilhadas entre diferentes redes (Lui & Kwok, 2002; Olmedilla & Palmer, 2005), o presente trabalho apresenta um modelo arquitetural para o compartilhamento de arquivos *P2P* (Rezende *et al.*, 2009), com o objetivo de facilitar o acesso a arquivos compartilhados na Internet. Também, busca-se oferecer mecanismos que possibilitem que aplicações largamente utilizadas, como navegadores *web*, possam se beneficiar da rede criada.

Diferente dos sistemas mais utilizados atualmente, assim como as propostas de interoperabilidade citadas neste capítulo, este trabalho apresenta uma arquitetura focada na organização de metadados em uma base de índices global e totalmente distribuída, assim como na recuperação dessas informações e acesso aos arquivos através do uso de padrões da indústria. De maneira semelhante ao que foi feito com a *web*, que teve suas páginas indexadas por ferramentas de buscas, este trabalho visa indexar arquivos disponíveis na Internet e em redes *P2P*, provendo

acesso a eles para diversas aplicações e usuários.

4.2 Arquitetura e Topologia

Buscando explorar a heterogeneidade dos computadores que venham fazer parte da rede formada a partir do presente trabalho, foi criada uma arquitetura composta por módulos totalmente independentes, podendo ser agrupados conforme perfil de cada participante. Além disso, propõe-se uma rede com topologia mista, contendo características de redes hierárquicas e ao mesmo tempo características de redes estruturadas (seção 3.4). A seguir, são dados maiores detalhes sobre a arquitetura e topologia da proposta deste trabalho.

O modelo proposto é apoiado em três módulos: *Index Space*, *Storage Space* e *Client API*. A figura 4.2 mostra cada um desses módulos e como eles podem ser combinados de diferentes formas. Conforme observado, eles são totalmente independentes e podem ser implementados de acordo com o perfil e objetivo do nó que deseja participar da rede criada.

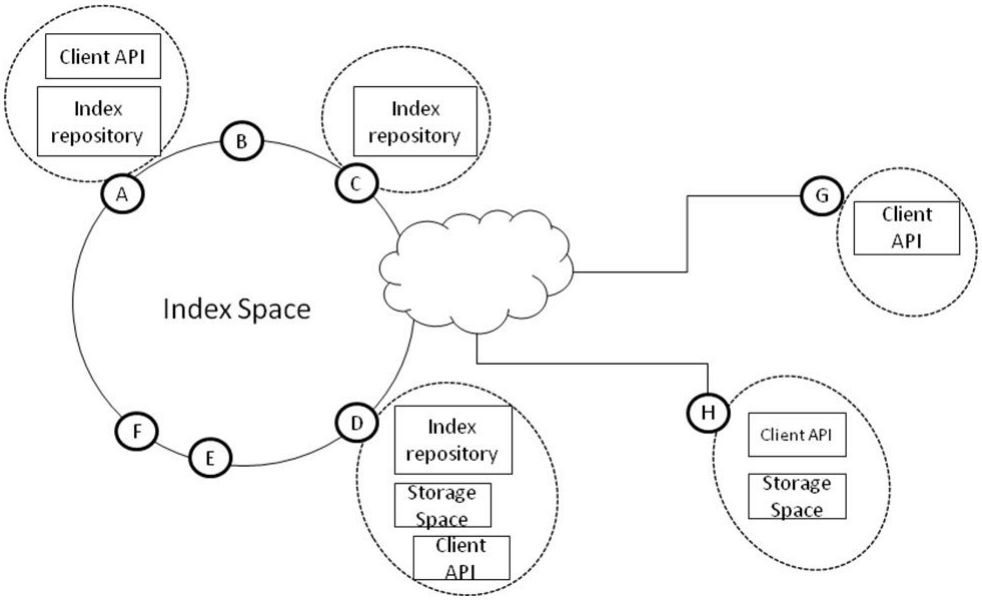


Figura 4.2: Arquitetura proposta para indexação e acesso a arquivos compartilhados.

Por fazer parte do *Index Space*, o nó A deve conter o módulo rotulado como *Index Repository* (pelo mesmo motivo, tal módulo é implementado também nos nós C e D). Além disso, o nó A acessa os índices armazenados no *Index Space*, por isso, ele possui também uma implementação da *Cliente API*.

Além de fazer parte do *Index Space*, o nó D acessa a base de índices através da *Client API* (como o nó A) e disponibiliza espaço de armazenamento local para os arquivos compartilhados (representado na figura 4.2 pelo módulo *Storage Space*).

A seguir, tem-se os nós G e H, que não compõe o *Index Space*, apenas consomem as informações lá armazenadas através da *Client API*. Além disso, o nó H também oferece espaço

de armazenamento local para o armazenamento de arquivos compartilhados.

O núcleo do modelo proposto está no módulo *Index Space*, que utiliza atualmente a *DHT Bamboo* para promover a organização estruturada do espaço de armazenamento de índices. Neste módulo, ficam armazenados os metadados referentes aos arquivos compartilhados, na forma de pares $[x, v]$, onde, x é uma das *tags* utilizadas para se fazer buscas ao arquivo e v é seu descritor, composto de informações como: título do documento, tamanho, autores, formato e URLs na qual o mesmo pode ser encontrado.

Além de definir como os índices são organizados e mantidos, o *Index Space* possibilita acesso aos mesmos através de uma interface simples, construída sobre as seguintes funcões "primárias":

- $put(x, v, p, t)$: adiciona uma entrada no *Index Space* composta pela chave x, valor v, e senha p (utilizada em operações de remoção) e período de validade t (em segundos);
- $get(x)$: retorna todas as entradas e_i = $[x_i, v_i]$, onde x_i = x;
- $remove(x, p)$: remove a entrada $[x_i, v_i]$, protegida pela senha p, tal que x_i = x.

As funções descritas anteriormente são acessadas através do uso de *RPC*, facilitando sua utilização em qualquer aplicação cliente que respeite as especificações impostas. Atualmente, este trabalho oferece suporte para as tecnologias *XML RPC* (Engelen *et al.*, 2000) e *Sun RPC*.

A seguir, tem-se possíveis implementações das funções *put, get* e *remove*, utilizando *XML RPC* e a linguagem de programação *Python*:

```
def put(self, gw, ttl, secret, key, value):
        pxy = ServerProxy(gw)
        res = {0:"Success", 1:"Capacity", 2:"Again"}
        m = hashlib.sha1(key)
        k = Binary(m.digest())
        v = Binary(value)
        t = int(ttl)
        m = hashlib.sha1(secret)
        shash = Binary(m.digest())
        if (secret == ""):
            return res[pxy.put(k,v,t,"put.py")]
        else
            return res[pxy.put_removable(k,v,"SHA",shash,t,"put.py")]
```

Onde: *gw* é o endereço de um dos nós do *Index Space*; *ttl* é o tempo que os valores inserido deverão ficar armazenados no *Index Space*; *key* é a chave que será inserida e *value* é o valor inserido. O valor de retorno é uma *string* contendo o *status* da operação.

```
def get(self, gw, maxValues, key):
        pxy = ServerProxy(gw)
        maxvals = int(maxValues)
        pm = Binary("");
        m = hashlib.sha1(key)
        k = Binary(m.digest())
        ret = []
        while 1:
            vals, pm = pxy.get(k,maxvals,pm,"get.py")
            for v in vals:
                if(pm.data != ""):
                    ret.append(pm.data)
                else:
                    break
        return ret
```

Onde: *gw* é o endereço de um dos nós do *Index Space*; *maxValues* é o número máximo de itens que deverão ser retornados; *key* é a chave procurada. O valor de retorno é uma lista de *strings* contendo os valores encontrados, dado a chave procurada.

```
def remove(self, gw, key, value, secret):
        pxy = ServerProxy(gw)
        res = {0:"Sucess", 1:"Capacity", 2:"Again"}
        m = hashlib.sha1(key)
        k = Binary(m.digest())
        m = hashlib.sha1(key)
        vh = Binary(m.digest())
        m = hashlib.sha1(secret)
        sec = Binary(m.digest())
        return res[pxy.rm(k,vh, "SHA", sec, 36000, "rm.py")]
```

Onde: *gw* é o endereço de um dos nós do *Index Space*; *key* é a chave que será removida; *value* é o valor da chave e *secret* é a senha utilizada para proteger a chave de remoções indevidas. O valor de retorno é uma *string* contendo o *status* da operação.

O módulo *Storage Space* é responsável pelo armazenamento dos arquivos compartilhados, podendo oferecê-los através de diversos protocolos, como *HTTP*, *FTP* e *torrent*. Este módulo é implementado de duas maneiras:

1. um servidor de arquivos pode ser disponibilizado junto com a aplicação cliente;
2. podem ser utilizadas fontes de arquivos já disponíveis na Internet, necessitando apenas que seus arquivos compartilhados sejam indexados na base *Index Space*

O primeiro caso é destinado aos programadores que desejam criar aplicações especificas para compartilhar arquivos nesta rede, implementando os três módulos na mesma. Para estes

casos, este trabalho disponibiliza na *Client API* um conjunto de classes que possibilitam a criação de servidores *HTTP* e *FTP*. O segundo caso pode ser aplicado por exemplo na indexação de *CDNs* (*Content Delivery Network*) já existentes, ou seja, tais repositórios podem ser indexados e ter suas informações armazenadas no *Index Space* para que seu acesso seja oferecido a outras aplicações que por sua vez farão acesso ao mesmo através da *Client API*. Desta forma, os arquivos compartilhados por estes servidores passam a estar disponíveis para qualquer aplicação que tenha acesso ao *Index Space*.

A *Client API* oferece uma biblioteca de *software* com as funções básicas para acesso ao repositório de índices criado, assim como construções de *software* que abstraem o processo de *download* dos arquivos compartilhados. Desta forma, é possível utilizar o mecanismos de herança e polimorfismo da programação orientada à objeto para integrar funcionalidades específicas àquelas já disponibilizadas pela *Client API*. A figura 4.3 apresenta as funcionalidades da *Client API*.

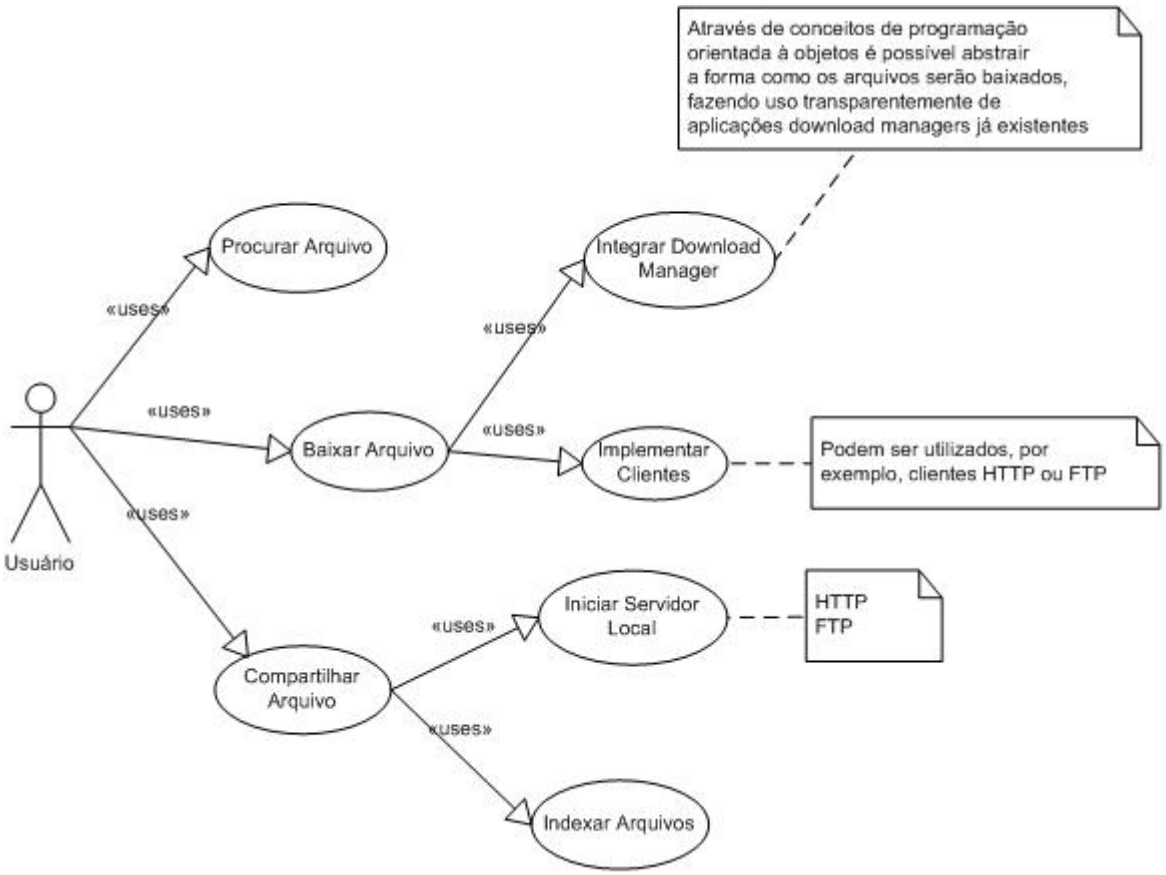


Figura 4.3: Funcionalidades da *Client API*.

Conforme observado na figura 4.3, é possível notar as seguintes funcionalidades na *Client API*:

- Busca por arquivos: processo realizado através de chamadas *RPC* ao repositório de índices. Tais chamadas utilizam como parâmetro uma ou mais *tags* que são utilizadas

para identificar arquivos;

- *Download* de arquivos: realizado através da utilização de aplicações *Download Manager* ou através da implementação de determinado protocolo (por exemplo, *FTP*) na própria *API*;

- Compartilhamento de arquivos: processo que pode contar com a utilização de um servidor de arquivos local (diponibilizado pela própria *API*) ou somente com chamadas *RPC* que inserem itens no repositórios de índices.

Até o presente momento, a *Client API* é implementada utilizando a linguagem de programação *Python*, porém, ela pode ser facilmente implementada em outras liguagens de programação.

A integração destes três módulos gera uma rede com a topologia mostrada na Figura 4.4. Conforme pode ser observado, esta topologia apresenta caracteríscas das redes hierárquicas e redes estruturadas. Do ponto de vista da conexão entre "clientes"e os nós que compõem o *Index Space*, esta rede apresenta comportamento semelhante às redes hierárquicas, onde os nós do *Index Space*, ao serem responsáveis pela manutenção dos índices, desempenham papel semelhante aos super nós de uma rede como a *Kazaa*. Porém, diferentemente de uma rede hierárquica, a conexão entre os nós que compõem o *Index Space* (super nós) é realizada seguindo regras bem definidas, através do uso de uma *DHT*.

Assim, esta topologia mantém os benefícios das redes hierárquicas, como escalabilidade e robustez, ao mesmo tempo em que torna possível buscas com maior alcance na rede gerada. Também, constata-se que o número de passos para se encontrar um arquivo pode ser significantemente menor do que nas redes estruturadas, visto que o algoritmo de roteamento da *DHT* permite buscas por índices de forma otimizada (Zhang, 2004).

4.3 Modelo de Indexação

Diversas abordagens podem ser utilizadas para indexar os conteúdos no *Index Space*, uma vez que este trabalho apresenta uma arquitetura independente de implementação. Buscando validar o modelo apresentado, este trabalho faz uso de um mecanismo de *tags*, ou seja, associa-se uma ou mais palavras chave ao arquivo compartilhado.

Sistemas construídos sobre *DTHs* utilizam mapeamento $[k, v]$ (onde k é a chave do objeto armazenado e v é a localização do mesmo) para mapear objetos e os nós onde os mesmos estão armazenados. Devido a esta característica, tais sistemas possuem como uma das principais limitações a necessidade de conhecimento prévio e exato dos identificadores dos objetos que são procurados.

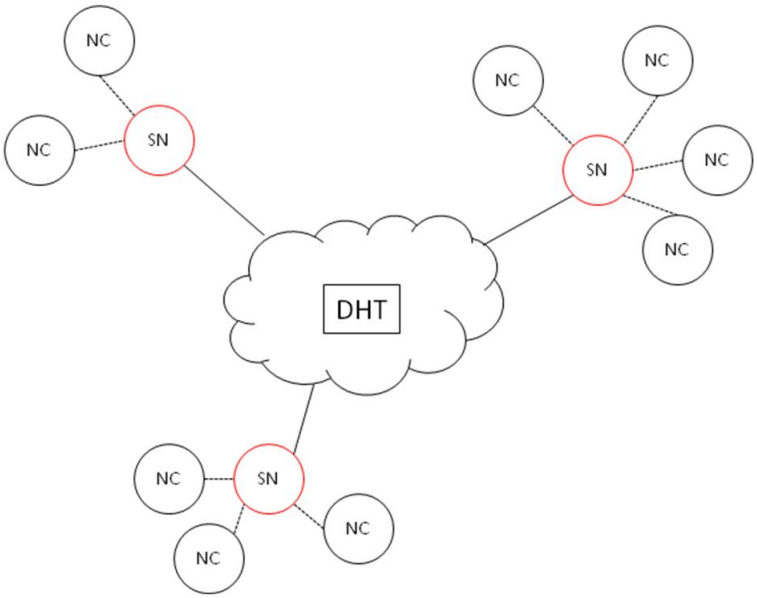


Figura 4.4: Topologia da rede criada segundo a proposta deste trabalho.

Buscando amenizar tal limitação, este trabalho faz uso de um modelo de indexação baseado em *tags*. Assim, além das buscas poderem ser realizadas pelo identificador do arquivo, as mesmas passam a ter um espaço de identificadores maior, contendo palavras relacionadas ao mesmo (a Figura 4.5 exibe uma ilustração onde n *tags* são associadas à determinado arquivo). Assim, este modelo pode servir de base para mecanismos de *web semântica*, utilizando as *tags* já existentes como base para consultas mais avançadas.

Uma das implicações deste modelo é o maior número de entradas na base de índices criada *Index Space*, uma vez que para cada objeto compartilhado, mais de uma entrada na *DHT* será realizada (num primeiro momento, este trabalho não limita o número de *tags* utilizadas).

Além de possibilitar buscas por palavras chave, tal solução possibilita também o refinamento dos resultados encontrados, podendo aplicar novas consultas com base nos metadados retornados por uma pesquisa anterior.

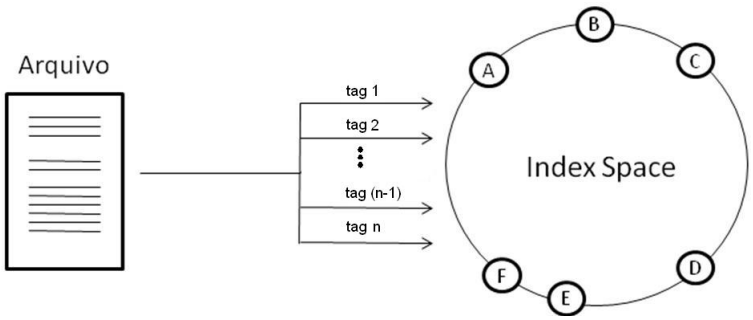


Figura 4.5: Múltiplas *tags* são utilizadas para identificar um arquivo na *Index Space*.

Apesar de oferecer grandes benefícios quando comparado com o modelo de indexação das *DHTs*, o sistema de *tags* apresentado ainda exibe muitas limitações, uma vez que ainda necessita de conhecimento prévio e exato de ao menos uma das *tags* utilizadas. Porém, tal

mecanismo pode facilmente ser substituído por mecanismos mais avançados, incluindo conceitos de busca semântica.

4.4 Download de Arquivos

O processo de *download* de arquivos é composto por duas etapas. Inicialmente, o usuário realiza uma busca pelo arquivo procurado no *Index Space*, a seguir, é feita a requisição pelo arquivo procurado diretamente a um ou mais nós que contém o mesmo.

Conforme exibido na figura 4.6, na primeira etapa é realizada a busca no *Index Space*. Esta busca utiliza como parâmetro uma ou mais *tags* contendo palavras chave de interesse. A seguir, com base no retorno da primeira etapa, uma conexão é criada com um ou mais nós que contenha o arquivo procurado.

A partir deste momento, o processo de *download* pode ser realizado conforme implementação utilizada, ou seja, o mecanismo utilizado para fazer acesso ao arquivo pode ser implementado conforme necessidade e recursos dos clientes.

Através da utilização de aplicações gerenciadoras de *downloads* (*download managers*) já existentes e largamente utilizadas (como Aria2 [1] , FlashGet [2] , e GetRight [3]), é possível realizar o processo de *download* a partir de mais de uma fonte em paralelo. A figura 4.7 exibe um processo de *download* assistido por uma dessas aplicações.

Pesquisas realizadas mostram que diferentes domínios podem apresentar grandes variações na latência e parâmetros da rede (Figura 4.8) (Dodonov, 2009). Desta forma, a proposta de uma arquitetura aberta e facilmente integrada com aplicações do tipo *download manager*, proporciona grandes benefícios a este trabalho, uma vez que possibilita acesso a diversas fontes de arquivos em paralelo, além da capacidade de detectar automáticamente qual a melhor fonte, ou combinação de fontes, para se obter determinado arquivo.

Todo o processo apresentado na figura 4.7 é apoiado pela *Client API*. Desta forma, os detalhes de como os arquivos serão acessados podem ser facilmente implementados de forma integrada ao processo de buscas no *Index Space*.

4.5 Upload de Arquivos

O processo de *upload* de arquivos consiste na inserção de um novo registro no *Index Space*, assim como na constante atualização do mesmo. Desta forma, busca-se evitar problemas de indisponibilidade causados por índices desatualizados.

[1] http://aria2.sourceforge.net/
[2] www.flashget.com/en/download.htm
[3] getright.com/

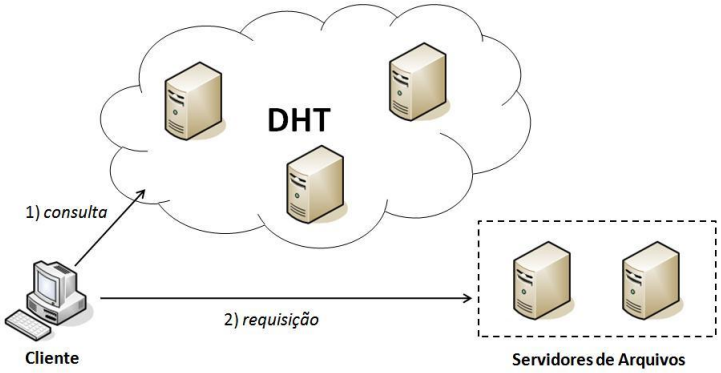


Figura 4.6: Processo de download de arquivos.

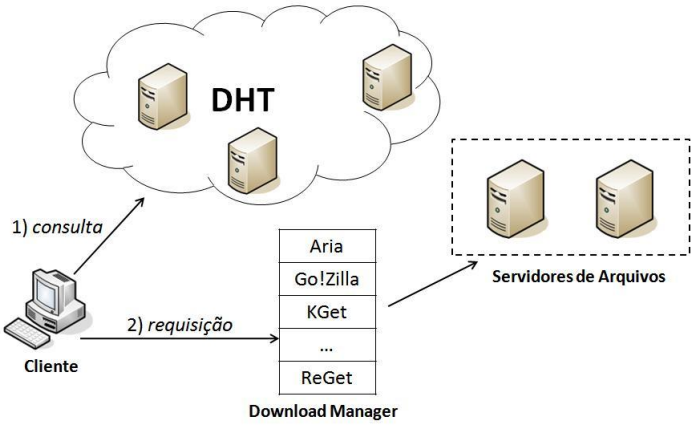


Figura 4.7: Processo de *download* utilizando aplicações gerenciadoras de *download* já existentes.

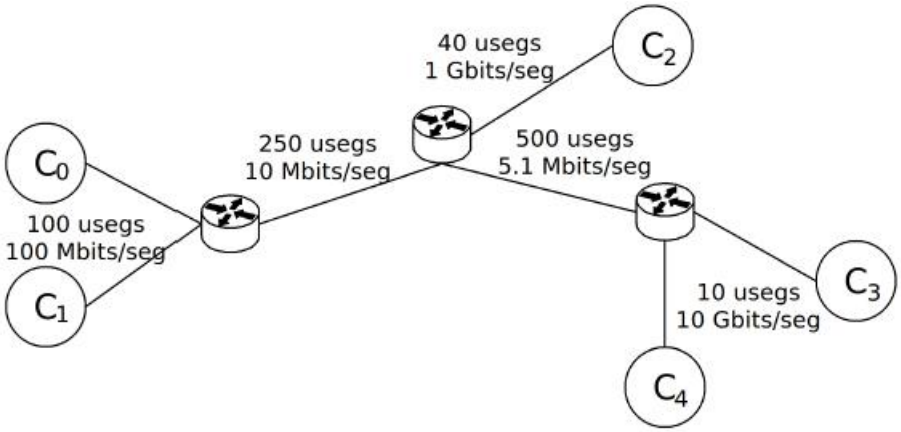


Figura 4.8: Variação dos parâmetros de rede em diferentes domínios (Dodonov, 2009).

No registro adicionado, estão presentes metadados do arquivo compartilhado. A localização do arquivo compartilhado pode variar de um repositório local criado junto a aplicação que faz acesso a base de índices do sistema, a uma localidade na Internet, ou seja, a indexação de conteúdos disponibilizados por repositórios já existentes.

4.6 Considerações Finais

Neste capítulo, foi apresentado um novo modelo de compartilhamento de arquivos *P2P*, baseado na indexação descentralizada e acesso independente de protocolo. Outrossim, foram exibidas motivações que orientaram o desenvolvimento deste trabalho e trabalhos relacionados ao mesmo.

O processo de indexação proposto, é baseado em uma infra-estrutura totalmente descentralizada e organizada através dos príncipios das *DHTs*. Com isso, foi obtido uma base de índices altamente escalável e com propriedades que possibilitam rápido acesso ao seu conteúdo, sem a necessidade de grande tráfego de rede, como ocorre nas buscas por inundação.

Também, foi proposto um modelo de indexação baseado em *tags*, o que ameniza as limitações das *DHTs*, através do aumento no espaço de busca, e ao mesmo tempo, possibilita que arquivos com conteúdos semelhantes (baseado nas *tags* utilizadas) possam ser também retornados ao usuário. Além disso, a independência do mecanismo de indexação utilizado possibilita que este modelo possa ser futuramente integrado com conceitos semânticos mais avançados.

O processo de acesso aos arquivos foi abstraído em uma *API*, a qual possibilita desde o uso de clientes *FTP* e *HTTP* pré configurados, até o uso de mecanismos customizados, podendo também ser feita a integração com aplicações *download managers* já existentes. Com a segunda abordagem, é possível estender o número de protocolos suportados nas operações de *download*, incluindo aqueles utilizados por outras redes *P2P*. Além disso, esta abordagem possibilita que avanços obtidos com os *download managers* sejam facilmente integrados neste trabalho.

CAPÍTULO

5

Estudo de Caso: AriaZilla

5.1 Considerações Preliminares

Atualmente, uma nova classe de aplicações vem recebendo cada dia mais atenção da comunidade de desenvolvedores de *software*, as extensões para navegadores *web*. Tais aplicações são destinadas ao aumento das funcionalidades dos navegadores através da instalação de *plugins*.

De acordo com as funcionalidades de tais *plugins*, os mesmos podem ser enquadrados em diversas categorias, como ferramentas de alertas, configuradores de aparência do navegador, gestores de páginas favoritas, ferramentas de idiomas, gerenciamento de fotos, músicas e vídeos, ferramentas de segurança, desenvolvimento *web* e colaboração.

Quando se trata especificamente de ferramentas de colaboração, é possível notar o desenvolvimento de aplicações cada vez mais inovadoras e abrangentes, como é o caso do projeto *Opera Unite* e o projeto *Veiled* (apresentado oficialmente em Julho de 2009, na BlackHat[1]). O primeiro é caracterizado pela disponibilização de ferramentas como compartilhamento de arquivos, compartilhamento de fotos, servidor *web*, *chat* e *media player* dentro do navegador Opera [2]. O segundo, é desenvolvido com o objetivo de criar uma rede privada, onde usuários possam compartilhar arquivos e idéias livremente.

Assim, levando em consideração a tendência de desenvolvimento de ferramentas para colaboração baseadas em navegadores *web*, portabilidade e facilidade de distribuição das mesmas, este capítulo apresenta um estudo de caso baseado em uma extensão para o navegador *web Mozilla Firefox*. Com isso, busca-se validar a proposta deste trabalho em um cenário real de uso.

[1] http://www.blackhat.com/html/bh-usa-09/bh-usa-09-schedule.html
[2] http://www.opera.com

5.2 Download de Arquivo através do Mozilla Firefox

Atualmente, tem-se vários *plugins/add-ons* que integram funcionalidades de *downloads managers* e troca de arquivos ao *Firefox*. A seguir, dois deles são apresentados, para que se possa verificar as funcionalidades dos mesmos, assim como as diferenças entre o que tais aplicações normalmente oferecem e o que pode ser implementado utilizando o modelo deste trabalho, conforme exemplificado no estudo de caso deste capítulo.

DownThemAll [3] representa um desses *add-ons* (Figura 5.1). Com ele, é possível fazer *download* de todo o conteúdo de uma página *web* através de dois *clicks* com o *mouse*. Além disso, seus criadores dizem que tal *add-on* pode oferecer ganhos de performance de até 400%, além de recursos como *download pause*, *download resume*, filtros e renomeação avançada de arquivos.

O *add-on FileZilla* [4] possibilita que servidor e cliente *FTP*, *FTPS* e *SFTP* sejam integrados ao *Firefox*. Também, é fornecido suporte a IPv6, filtros de arquivos, favoritos e *logs*. A figura 5.2 apresenta a janela principal do *FileZilla*.

Diferentemente de *DownThemAll* e *FileZilla*, *FlashGot* [5] apresenta um ferramenta que possibilita a integração de diversos *download managers* ao *Firefox*, variando com a disponibilidade de aplicações para cada sistema operacional.

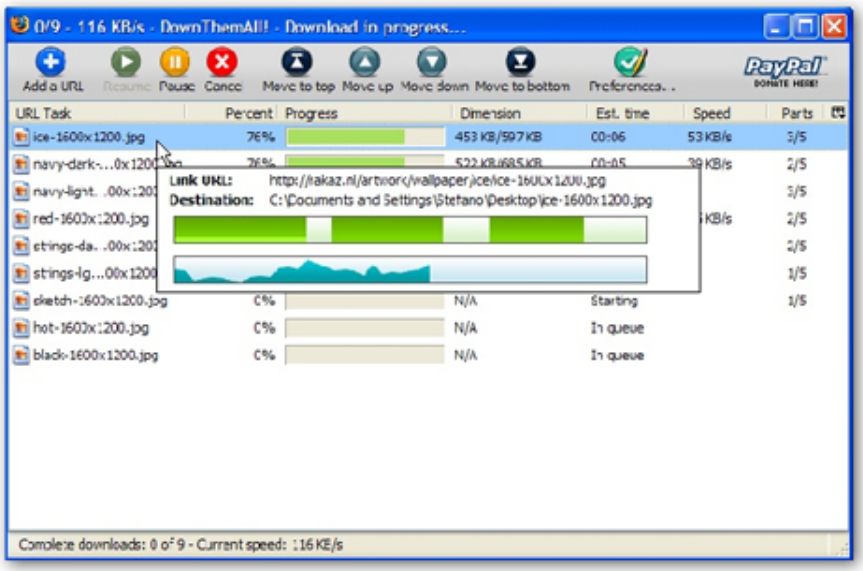


Figura 5.1: DownThemAll: *downloads* em andamento.

Apesar das diversas opções de *download managers* que atualmente são integrado aos navegadores *web*, verifica-se que grande parte deles são implementados para possibilitar acesso a arquivos disponíveis em páginas *web*, sem a possibilidade de tornar disponível um arquivo

[3] https://addons.mozilla.org/en-US/firefox/addon/201
[4] http://filezilla-project.org/
[5] http://flashgot.net/

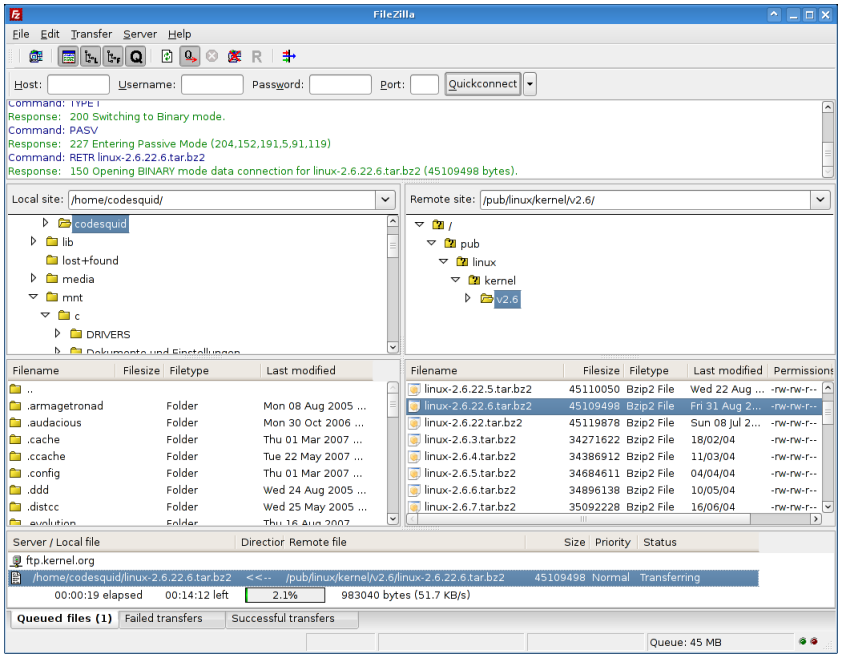


Figura 5.2: FileZilla: Janela principal.

local ou acesso a arquivos presentes em máquinas de outros usuários da Internet.

5.3 Ambiente de Teste

Para avaliar o modelo apresentado neste trabalho, foi criado um ambiente onde buscou-se replicar cada um dos módulos apresentadados neste modelo de compartilhamento, assim como a interação entre os mesmos no ambiente real de sua utilização, a Internet.

Para tanto, foi criada uma infra-estrutura de *Index Space* e uma aplicação cliente que através da *Client API*, é capaz de indexar no *Index Space* arquivos locais e URLs da Internet, assim como realizar *download* de conteúdo já indexado.

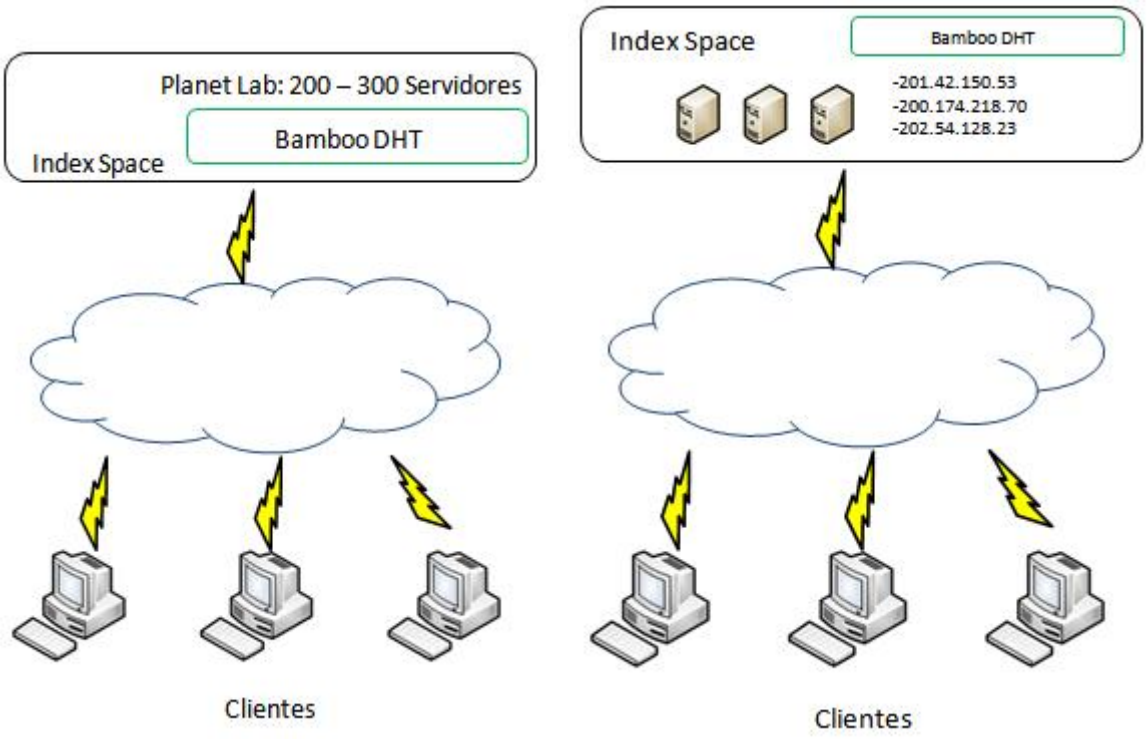
5.3.1 Repositório de Índices

Para reproduzir o *Index Space*, duas abordagens foram utilizadas:

1. Uso da infra-estrutura da *OpenDHT*;
2. Criação de uma base de índices própria, baseada na *DHT Bamboo*.

OpenDHT é um projeto que tem por objetivo disponibilizar uma *DHT* de acesso público (contendo entre 200 e 300 nós), a qual possibilita que desenvolvedores de aplicações *P2P* possam trabalhar em seus projetos sem se preocupar com detalhes de implementação de tal estrutura. Para tanto, a mesma oferece uma interface de acesso simplificado, possibilitando que operações

de inclusão, acesso e remoção aos itens armazenados possam ser realizado via *RPC*. Assim, além de simplificar o desenvolvimento do estudo de caso, a primeira abordagem possibilitou que buscas por arquivos fossem realizadas numa grande *DHT*(Figura 5.3a).



(a) *Planet Lab* (b) Infra-estrutura própria.

Figura 5.3: Duas abordagens utilizadas na criação do *Index Space*.

Porém, apesar dos resultados satisfatórios com as buscas que foram realizadas, em determinados momentos a mesma não possibilitava a inserção de novos itens na *DHT*, devido limitações do espaço de armazenamento. Desta forma, foi iniciada a criação de uma infra-estrutura própria para armazenar os índices dos arquivos. Para tanto, foram utilizados recursos de *hardware* e uma camada de *software*.A plataforma computacional utilizada foi constituída por 3 computadores:

- Intel Celeron M 1.73 GHz, 512 MB RAM, Ubuntu 9 (virtualizado), conexão com link de 2 Mb;
- AMD Sempron 1.8 GHz, 1.5 GB RAM, Debian Squeeze, conexão com link de 2 Mb.
- Intel Core 2 Duo, 512 MB RAM, Ubuntu 9, conexão com link de 2 Mb.

Em cada uma destas máquinas foi criada uma instância de nós da *DHT Bamboo*, a qual fornece serviços de roteamento de mensagens, manutenção de índices e interface de acesso contendo as funções básicas para inserção, busca e remoção de itens (Figura 5.3b).

Em seguida, foi executado um *script* Python para oferecer uma carga inicial de dados ao repositório criado. Tais dados foram baseados em repositórios *web* de instaladores de diversas distribuições do sistema operacional Linux , sendo encontrados, nesta base, arquivos disponibilizados para *download* via *HTTP*, *FTP* e *torrent*.

Apesar de não sobrecarregar a base de índices, esta abordagem permitiu que fosse criada uma base de índices para atender os requisitos do estudo de caso desenvolvido.

5.3.2 AriaZilla

Para simular a utilização da *Client API* foi desenvolvida um extensão para o navegador *Mozilla Firefox*, chamada *AriaZilla*. Esta extensão integra as funcionalidades necessárias para o acesso ao *Index Space*, um servidor *FTP* local (*Storage Space*), para disponibilizar os arquivos compartilhados pelo usuário, assim como a integração com o *download manager Aria2*. Para seu desenvolvimento, foram utilizadas as seguintes tecnologias (Feldt, 2007).

XUL (XML User Interface Language): é a linguagem aberta e baseada em *XML* criada pela Mozilla. Seu objetivo é possibilitar a criação de aplicações multi-plataformas, que podem ser executadas independentemente de conexão com a Internet. Entre os recursos oferecidos, estão janelas, botões e uma grande coleção composta por objetos de interface de usuário.

JavaScript: linguagem utilizada para interagir com os componentes de interface gráfica da aplicação criada.

XPCOM (Cross Platform Component Object Model): um dos principais fatores que transformam o ambiente Mozilla em uma autêntica plataforma, sendo responsável pelor funcionalidades que tornam o desenvolvimento de *software* para Mozilla em algo multi-plataforma. Dentre suas funcionalidades, podem ser citadas: gerênciamento de componentes, abstração de arquivos, troca de mensagem entre objetos, gerênciamento de memória. Além disso, ele é responsável pela comunicação entre o JavaScript e a lógica de negócio criada em uma das linguagens suportadas, por exemplo, Python.

Além dessas tecnologias, foi necessário instalar a extensão *PythonExt* [6] no *Firefox*. Desta forma, o navegador pode contar com os benefícios da linguagem *Python* para o desenvolvimento de suas extensões.

Assim, tem-se a arquitetura da figura 5.4:

Com isso, somadas as portabilidades da plataforma *Mozilla*, com a portabilidade da linguagem *Python*, obteve-se uma solução altamente interoperável, com suporte a vários sistemas operacionais e diferentes plataformas.

[6] http://pyxpcomext.mozdev.org/

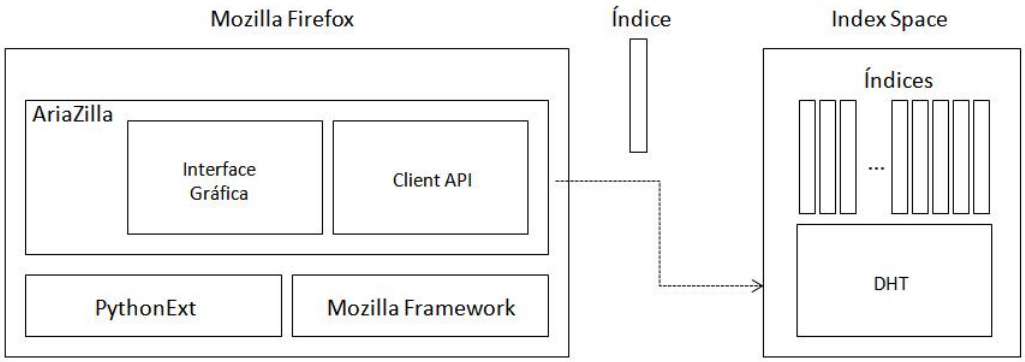


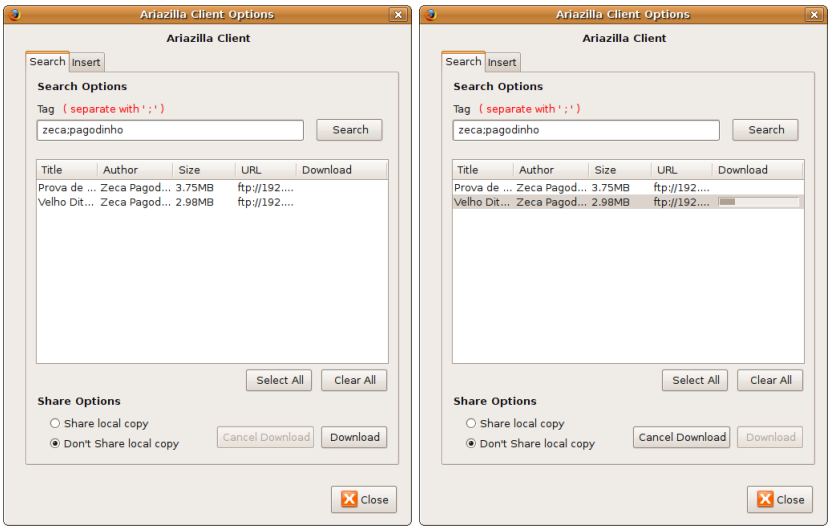
Figura 5.4: AriaZilla: Arquitetura.

AriaZilla: Download e Upload

As figuras 5.5 e 5.6 apresentam a interface utilizada para realizar buscas por arquivos e a interface utilizada para realizar o compartilhamento de arquivos locais, respectivamente.

Para avaliar a flexibilidade de *download* com a *AriaZilla*, dois modelos de *downloads* foram implementados. O primeiro baseado em um cliente *FTP* disponibilizado pela própria *Client API*, implementado dentro da *AriaZilla*, e o segundo, fazendo uso do *download manager Aria2*.

Com a abordagem baseada na aplicação *Aria2*, foi possível dispor da funcionalidade de *downloads* paralelos, além do suporte a diferentes protocolos, como *HTTP*, *HTTPS*, *FTP* e *torrent*. Conforme estudos recentes (Bhuvaneswaran *et al.*, 2006; Mello *et al.*, 2007), a possibilidade de *download* a partir de mais de uma fonte simultaneamente traz grandes benefícios quando comparado com métodos convencionais.



(a) AriaZilla: Busca por um arquivo (b) AriaZilla: *Download de um arquivo*

Figura 5.5: Busca e *download* de arquivos com AriaZilla.

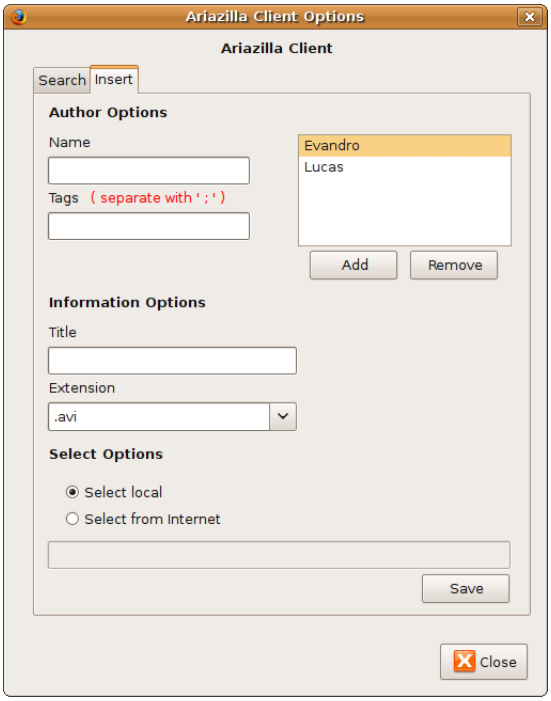


Figura 5.6: AriaZilla: Interface para realizar a indexação de um arquivo.

5.4 Resultados

Com o estudo de caso *AriaZilla*, foi possível observar a flexibilidade e benefícios do modelo proposto. Dentre os pontos de destaque, podem ser citados:

1. A integração com o *Index Space* possibilitou grande desempenho nas buscas, mesmo quando estas foram realizadas em um ambiente largamente utilizado como o *Planet Lab*;

2. As buscas baseadas em *tags* proporcionaram boa experiência de uso, possibilitando a descoberta de vários itens relacionados a determinado termo;

3. Diferentemente das extensões que fazem *downloads* a partir de *links* das páginas *web*, a extensão AriaZilla possibilitou realizar *download* de diversas fontes, inclusive repositórios utilizados por outras redes *P2P*;

4. Além de acessar arquivos compartilhados por outras redes, é possível também compartilhar arquivos locais, a partir da inicialização de um servidor *FTP* disponibilizado pela *Client API*;

5. Também, é possível indexar *URLs* na própria *AriaZilla*, desta forma, arquivos encontrados na Internet, através de ferramentas de buscas, podem ser indexados no *Index Space* pelos próprios usuários da aplicação;

6. Apesar de ter utilizado o *download manager Aria2*, a proposta mostrou-se flexível, possibilitando a utilização de outras aplicações deste tipo. Assim, pode-se atualizar a

extensão conforme tais aplicações evoluem, ou conforme disponibilidade para um sistema operacional específico;

A tabela 5.1 resume as diferenças e vantagens entre AriaZilla e as extensões apresentadas neste capítulo.

Tabela 5.1: Comparativo entre funcionalidades do AriaZilla, DownThemAll e FileZilla.

Extensão	Mecanismos de Busca	Fontes de arquivos	Servidor local	Download paralelo
AriaZilla	Sim	Máquinas de usuários, Redes P2P, Internet	Sim	Sim
DownThemAll	Não	Links em páginas web	Não	Não
FileZilla	Não	Servidores FTP	Sim	Não

Desta forma, fica claro as vantagens de se utilizar o modelo apresentado neste trabalho par aa criação de redes de compartilhamento de arquivos, podendo o conteúdo das mesmas ser acessado a partir das mais diversas aplicações, por exemplo, uma extensão de navegador *web*.

5.5 Considerações Finais

Para avaliar o modelo proposto neste trabalho, foi desenvolvida uma aplicação protótipo, a qual foi testada em um estudo de caso (*AriaZilla*). Tal aplicação, foi baseada em uma extensão para o navegador *Mozilla Firefox*, sendo desenvolvida através das tecnologias *XUL*, *XPCOM* e *Python*.

A aplicação utilizada no estudo de caso, além de atender os requisitos esperados para avaliar este trabalho, mostrou-se também altamente interoperável, podendo ser utilizada em diversos sistemas operacionais e plataformas, dado as tecnologias utilizadas em seu desenvolvimento.

Além de ser uma prova de conceito do modelo aqui apresentado, esta aplicação mostrou estar alinhada com novidades que estão sendo realizadas na área de colaboração entre usuários da Internet, como é o caso, por exemplo, dos projetos *Opera Unite* e *Veiled*. Assim, fica claro que este trabalho pode ser aplicado nos mais diversos cenários, além do compartilhamento de arquivos.

CAPÍTULO

6

Conclusões

6.1 Conclusões Gerais

No contexto abordado, diversas atividades foram realizadas para criar e validar a proposta deste trabalho. Inicialmente, foi realizado um estudo com o objetivo de avaliar tecnologias de sistemas distribuídos, abordando as motivações para desenvolvimento dos mesmos, suas principais características, estado atual e mecanismos utilizados para o compartilhamento de dados em tais sistemas.

A seguir, foi abordada com maiores detalhes uma categoria especifica de sistema distribuído, os sistemas *P2P*. Foram realizados estudos com o objetivo de caracterizar os mesmos, bem como tecnologias utilizadas pelas aplicações desta categoria de sistemas. Também, foi feito um levantamento sobre aplicações *P2P* utilizadas atualmente para o compartilhamento de arquivos, verificando suas principais características e limitações. Com base nesses estudos, foram definidas as características e requisitos da proposta aqui apresentada. Também, foram estudados sistemas cujo objetivo é similar ao apresentado neste trabalho, ou seja, superar as deficiências constatadas nos sistemas de compartilhamento de arquivos *P2P* e propor novos mecanismos de compartilhamento de arquivos, assim como possibilitar acesso a grande quantia de arquivos já disponíveis na Internet.

Então, foram apresentadas as motivações para este trabalho e uma nova proposta arquitetural para compartilhamento de arquivos *P2P*, caracterizada pelo foco na criação de uma base índices organizada em uma rede *P2P* estruturada, uso de padrões da indústria para acesso aos índices e independência de protocolo.

Finalmente, a proposta deste trabalho foi avaliada em um estudo de caso, o qual foi baseado em uma extensão para o navegador *web Mozilla Firefox*. Com o resultado dos testes

realizados, foi possível fazer um breve comparativo entre algumas extensões do *Firefox*, cujo objetivo é facilitar a troca de arquivos entre seus usuários, e o que pode ser obtido através do uso da proposta aqui apresentada.

Assim, foi possível validar a proposta apresentada em um cenário real de uso e verificar todos os benefícios de tal modelo, como o bom desempenho nas buscas em ambientes formados por centenas de computadores, *downloads* paralelos a partir de diversas fontes (como repositórios da Internet e outras redes *P2P*), compartilhamento de arquivos locais, indexação de arquivos já compartilhados na Internet e flexibilidade na integração com aplicações do tipo *download manager*. Além disso, durante o desenvolvimento deste trabalho constatou-se total alinhamento do mesmo com novas tecnologias desenvolvidas na área de compartilhamento de arquivos e colaboração *P2P*.

6.2 Trabalhos Futuros

Trabalhos futuros podem ser realizados para melhorias e complementos deste trabalho, como:

- Estudo comparativo entre diferentes *DHTs*, para que seja determinada aquela que pode oferecer melhor desempenho ao modelo proposto, levando em consideração os mecanismos utilizado para sua manutenção e sistema de roteamento de mensagens;
- Disponibilização de uma *API* que possibilite a criação de novos nós para compor a base de índices utilizada;
- Estudo de outros tipos de aplicações que podem se beneficiar com o modelo aqui apresentado. Algumas aplicações que podem ser estudadas são: redes sociais e sistemas de *e-mail P2P*;
- Levantamento das implicações relacionadas a segurança que surgem com esta proposta de trabalho;
- Proposta de um modelo de incentivo para que usuários disponibilizem espaço para a base de índices e arquivos armazenados em seus computadores;
- Estudos sobre técnicas avançadas de buscas em *DHTs*, visando eliminar as limitações existentes com os mecanismos tradicionais. Assim, dado uma *tag*, a rede deve ser capaz de sugerir itens relacionados a mesma.

Referências

Aguiar, C. S.; Cruz, D. I.; Ulson, R. S.; Cavenaghi, M. A. (2008). The application of distributed virtual machines for enterprise computer management: A two-tier network file system for image provisioning and management. *COMPSAC '08: Proceedings of the 2008 32nd Annual IEEE International Computer Software and Applications Conference*, p. 428–431, Washington, DC, USA. IEEE Computer Society.

Amza, C.; Cox, A. L.; Dwarkadas, S.; Keleher, P.; Lu, H.; Rajamony, R.; Yu, W.; Zwaenepoel, W. (1996). Treadmarks: Shared memory computing on networks of workstations. *IEEE Computer*, v.29, n.2, p.18–28.

Anderson, D. P.; Cobb, J.; Korpela, E.; Lebofsky, M.; Werthimer, D. (2002). Seti@home: an experiment in public-resource computing. *Commun. ACM*, v.45, n.11, p.56–61.

Androutsellis-Theotokis, S.; Spinellis, D. (2004). A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys (CSUR)*, v.36, n.4, p.335–371.

Barakat, C. (2001). Tcp/IP modeling and validation. *IEEE Network*, v.15, n.3, p.38–47.

Bennett, J. K.; Carter, J. B.; Zwaenepoel, W. (1990). Munin: Distributed shared memory based on type-specific memory coherence. p. 168–176.

Bhuvaneswaran, R. S.; Katayama, Y.; Takahashi, N. (2006). Redundant parallel data transfer schemes for the grid environment. *ACSW Frontiers '06: Proceedings of the 2006 Australasian workshops on Grid computing and e-research*, p. 71–78, Darlinghurst, Australia, Australia. Australian Computer Society, Inc.

Clarke, I.; Sandberg, O.; Wiley, B.; Hong, T. W. (2001). Freenet: A distributed anonymous information storage and retrieval system. *Lecture Notes in Computer Science*, v.2009, p.46–??

Curbera, F.; Duftler, M.; Khalaf, R.; Nagy, W.; Mukhi, N.; Weerawarana, S. (2002). Unraveling the web services web: an introduction to soap, wsdl, and uddi. *Internet Computing, IEEE*, v.6, n.2, p.86–93.

Dias, R. A. (2006). Computação em grade para bioinformática. Dissertação (Mestrado), IME/USP.

Dodonov, E. (2004). Um mecanismo integrado de cache e prefetching para sistemas de entrada e saída de alto desempenho. Dissertação (Mestrado), DC/UFSCar.

Dodonov, E. (2009). *Uma abordagem de predição da dinâmica comportamental de processos para prover autonomia a ambientes distribuídos*. Tese (Doutorado), ICMC/USP.

Dollimore, J.; Kindberg, T.; Coulouris, G. (2005). *Distributed Systems: Concepts and Design (4th Edition) (International Computer Science Series)*. Addison Wesley.

Ein-Dor, P. (1985). Grosch's law re-revisited: Cpu power and the cost of computation. *Commun. ACM*, v.28, n.2, p.142–151.

Engelen, R. V.; Gallivan, K.; Gupta, G.; Cybenco, G. (2000). Xml-rpc agents for distributed scientific computing. *IMACS*.

Fan, J.; Kambhampati, S. (2005). A snapshot of public web services. *SIGMOD Rec.*, v.34, n.1, p.24–32.

Feldt, K. (2007). *Programming Firefox: Building Rich Internet Applications with XUL*. O'Reilly Media, Inc.

Ferguson, D. (2006). Trends and statistics in peer-to-peer. *Workshop on Technical and Legal Aspects of Peer-to-Peer Television*.

Foster, I.; Kesselman, C. (1997). Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, v.11, p.115–128.

Foster, I.; Kesselman, C.; Tuecke, S. (2001). The anatomy of the grid: Enabling scalable virtual organizations. *Lecture Notes in Computer Science*, v.2150.

Francesquini, E. C. (2007). Hermes: um arcabouço para a programação de aplicações p2p. Dissertação (Mestrado), IME/USP.

Good, N. S.; Krekelberg, A. (2003). Usability and privacy: a study of kazaa p2p file-sharing. *CHI '03: Proceedings of the conference on Human factors in computing systems*, p. 137–144. ACM Press.

Hu, T. H. (2005). *Restructuring Peer-to-Peer Networks*. Tese (Doutorado), New South Wales University.

Kubiatowicz, J. (2003). Extracting guarantees from chaos. *Commun. ACM*, v.46, n.2, p.33–38.

Laffranchi, M. M. (2004). *Uma solução peer-to-peer para a implantação de jogos multiusuário baseada no padrão emergente MPEG-4 MU*. Tese (Doutorado), DC/UFSCar.

Liang, J.; Kumar, R.; Ross, K. (2004). Understanding kazaa.

Liu, X.; Deters, R. (2007). An efficient dual caching strategy for web service-enabled pdas. *SAC '07: Proceedings of the 2007 ACM symposium on Applied computing*, p. 788–794, New York, NY, USA. ACM.

Lu, J. (2004). Some research on componentware frameworks based on mobile agent technology. *SIGSOFT Softw. Eng. Notes*, v.29, n.2, p.8–8.

Lui, S. M.; Kwok, S. H. (2002). Interoperability of peer-to-peer file sharing protocols. *SIGecom Exch.*, v.3, n.3, p.25–33.

Mello, R. F.; Andrade, A. J.; Dodonov, E.; Ishii, R. P.; Yang, L. T. (2007). *Parallel and Distributed Processing and Applications*. Springer Berlin / Heidelberg.

Mungioli, A. S. R. (2005). *Uma proposta de tecnologia para videoconferência integrando tecnologias grid*. Tese (Doutorado), Escola Politécnica/USP.

Nejdl, W. (2002). Semantic web and peer-to-peer technologies for distributed learning repositories. *Proceedings of the IFIP 17th World Computer Congress - TC12 Stream on Intelligent Information Processing*, p. 33–50, Deventer, The Netherlands. ACM.

Oliveira, J. A. (2006). Um estudo comparativo de cargas de trabalho e políticas de escalonamento para aplicações paralelas em clusters e grids computacionais. Dissertação (Mestrado), ICMC/USP.

Olmedilla, D.; Palmer, M. (2005). Interoperability for peer-to-peer network: Opening p2p to the rest of the world. *WWW2005*.

Oram, A. (2001). *Peer-to-Peer : Harnessing the Power of Disruptive Technologies*. O'Reilly.

Pietro (2001). Mobile agents: an introduction. *Microprocessors and Microsystems*, v.25, n.2, p.65–74.

Platzer, C. (2004). Trust-based security in web-services. Dissertação (Mestrado), Information System Institute/Technical University of Viena, Austria.

Ramachandran, U.; Yousef, M.; Khalid, A. (1989). An implementation of distributed shared memory. *Symposium on Experiences with Distributed and Multiprocessor Systems*, p. 21–38.

Reis, V. Q. (2005). Escalonamento em grids computacionais: estudo de caso. Dissertação (Mestrado), ICMC/USP, São Carlos, Brasil.

Rezende, E. S.; Dodonov, E.; Ulson, R. S.; Cavenaghi, M. A.; Lobato, R. S. (2009). Towards interoperability in p2p world: An indexing middleware for multi-protocol peer-to-peer data sharing. *Internet and Web Applications and Services, International Conference on*, v.0, p.102–105.

Rhea, S.; Godfrey, B.; Karp, B.; Kubiatowicz, J.; Ratnasamy, S.; Shenker, S.; Stoica, I.; Yu, H. (2005). Opendht: A public dht service and its uses.

Ripeanu, M. (2001). Peer-to-peer architecture case study: Gnutella network. p. 99–100.

Rocha, J.; Domingues, M.; Callado, A.; Souto, E.; Silvestre, G.; Kamienski, C.; Sadok, D. (2004). Peer-to-peer: Computação colaborativa na internet.

Rowstron, A.; Druschel, P. (2001). Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, v.2218, p.329–??

Santos, A. S. (2005). Uma proposta de escalonamento descentralizado de tarefas para computação em grade. Dissertação (Mestrado), Instituto de Informática/UFRGS, Porto Alegre, Brasil.

Scales, D. J.; Gharachorloo, K.; Thekkath, C. A. (1996). Shasta: a low overhead, software-only approach for supporting fine-grain shared memory. *SIGOPS Oper. Syst. Rev.*, v.30, n.5, p.174–185.

Silberschatz, A.; Galvin, P. B.; Gagne, G. (2004). *Operating System Concepts*. Wiley.

Stoica, I.; Morris, R.; Karger, D.; Kaashoek, F. M.; Balakrishnan, H. (2001). Chord: A scalable peer-to-peer lookup service for internet applications. *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, v. 31, p. 149–160, New York, NY, USA. ACM Press.

Tanenbaum, A. S.; van Steen, M. (2006). *Distributed Systems: Principles and Paradigms (2nd Edition)*. Prentice Hall.

Tsai, W. T.; Paul, R.; Wang, Y.; Fan, C.; Wang, D. (2002). Extending wsdl to facilitate web services testing. p. 171–172.

Vaughan-Nichols, S. J. (2003). Xml raises concerns as it gains prominence. *Computer*, v.36, n.5, p.14–16.

Yeo, C.; Buyya, R.; Pourreza, H.; Eskicioglu, R.; Graham, P.; Sommers, F. (2006). Cluster computing: High-performance, high-availability, and high-throughput processing on a network of computers. p. 521–551.

Zhang, Z. (2004). The power of dht as a logical space. *Distributed Computing Systems, 2004. FTDCS 2004. Proceedings. 10th IEEE International Workshop on Future Trends of*, p. 325–331.

Zhao, B. Y.; Huang, L.; Stribling, J.; Rhea, S. C.; Joseph, A. D.; Kubiatowicz, J. D. (2004). Tapestry: A resilient global-scale overlay for service deployment. *Selected Areas in Communications, IEEE Journal on*, v.22, n.1, p.41–53.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo