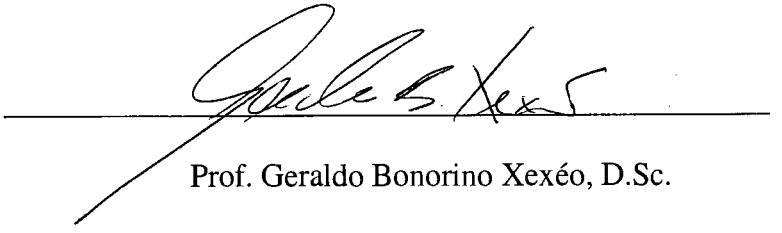
UM ESQUEMA DE VOTAÇÃO DINÂMICA DESCENTRALIZADO E TOLERANTE A FALHAS PARA REDES PEER-TO-PEER

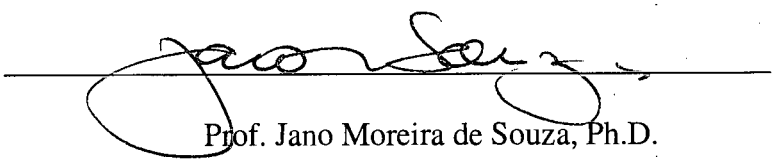
Bruno da Rocha Braga

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

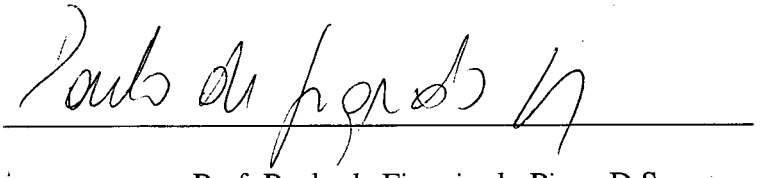


Prof. Geraldo Bonorino Xexéo, D.Sc.



Prof. Jano Moreira de Souza, Ph.D.

Prof[a]. Marta Lima Queirós Mattoso, D.Sc.



Prof. Paulo de Figueiredo Pires, D.Sc.

RIO DE JANEIRO, RJ – BRASIL.

NOVEMBRO DE 2005

BRAGA, BRUNO DA ROCHA

Um Esquema de Votação Dinâmica Descentralizado e Tolerante a Falhas para Redes Peer-to-Peer [Rio de Janeiro] 2005

XIV, 101p., 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 2005)

Dissertação – Universidade Federal do Rio de Janeiro, COPPE

1. Peer-to-Peer
2. Banco de Dados Distribuídos
3. Sistemas Distribuídos
4. Algoritmos Distribuídos
5. Gestão do Conhecimento

I. COPPE/UFRJ II. Título (série)

Aos meus pais, Eric e Elizabeth, pelo apoio e carinho que foram indispensáveis para minha formação como pessoa.

Aos meus avós, Néllio, Ilda, Antônio e Rita, que também responsáveis pela minha presença aqui hoje.

Agradecimentos

Aos meus pais, Eric Ormond Braga e Elizabeth Ferreira da Rocha Braga, sem os quais, definitivamente, eu não estaria aqui. Há muito que agradecê-los, tanto que seria suficiente para dobrar o número de páginas desta dissertação. Que eu tenha muitas oportunidades de retribuir todo o carinho e esperança que vocês depositaram em mim ao longo desses anos.

Um agradecimento especial a minha mãe, simplesmente por ela estar aqui hoje. Tudo o que você passou e superou no ano passado, certamente, tem pelo menos uma razão de ser: mostrar-me, mais uma vez, o quanto você é importante na minha vida.

A minha avó, Rita Ferreira da Rocha, que nos deixou em maio deste ano. Ela foi uma das pessoas mais importantes da minha vida. Ajudou a criar a mim e a minha irmã, mesmo com todas as dificuldades de sua saúde debilitada, por quase 25 anos. Muito obrigado, vovó Rita.

Ao meu orientador Geraldo Bonorino Xexéo e ao professor Jano Moreira de Souza pelas condições para o desenvolvimento do meu trabalho. Sem dúvida, estas tiveram grande influência nos resultados alcançados e no meu crescimento como profissional.

A todos os bons professores que tive ao longo da vida. Na impossibilidade de citá-los – felizmente, foram muitos –, expresso meu carinho pelas saudosas casas pelo qual passei: o Colégio Lobo da Cunha, o Colégio Militar do Rio de Janeiro e a Universidade Federal do Rio de Janeiro.

Aos meus amigos, que na impossibilidade de enunciar o nome de todos, deixo como referência o *Orkut*. Em especial, agradeço a Andressa Kalil, Rafael Leonardo, José Nogueira e Josué Nunes pelas discussões a respeito deste trabalho; a Rodrigo Mibielli pela troca de experiências sobre J2EE enquanto organizávamos o projeto em que trabalhávamos; a Thiago Cordeiro pela revisão de uma versão prematura deste texto; a Adriana Vivacqua pela revisão do meu inglês; a Pedro Cardoso e Luis Rigo

pelos livros emprestados; e a minha irmã, Bianca, e a minha tia Nelilda pela ajuda no leva-e-traz que se tornou esta dissertação quando eu fui para Brasília.

Por fim, agradeço a todos os que chamam para si a responsabilidade de não apenas preservar como também expandir as fronteiras do conhecimento. Essas pessoas contribuíram, de maneira direta ou indireta, para a realização deste trabalho.

Prefácio

Acredito que nós vivemos em uma Era ímpar. Alguns podem achar que disseram o mesmo no Pós-guerra, na Revolução Industrial, ou na Renascença, mas estes novos tempos são realmente diferentes dos demais em pelo menos um aspecto. Todos os marcos da nossa história se caracterizam por um grande avanço que alguma área do conhecimento humano dera em relação à época anterior. Este conhecimento, porém, sempre esteve restrito a uma minoria. E esta Era se destaca justamente pela oportunidade que temos de, pela primeira vez, disseminar o conhecimento em larga escala. Podemos compartilhar o pouco que conhecemos para conhecermos ainda mais. Podemos ajudar a fincar um novo marco, do outro lado do mundo, sem nem mesmo tomarmos conhecimento disto – e a recíproca é verdadeira.

No entanto, somente conhecimento basta para fincarmos marcos indeléveis na história? No Egito, Imhotep foi um pioneiro: construiu a primeira pirâmide do Saara. Obra esta que foi possível graças à visão do faraó Djoser. A história mostra que este padrão se repetiu em quase todas as grandes obras da humanidade. As navegações do rei Dom Manuel – não por acaso, conhecido como "O Venturoso" –, a modernização do Japão pelo Imperador Meiji – que em japonês, significa "Governo Iluminado" – e a indústria baseada em linha de montagem de Henri Ford, que começou com seu primeiro emprego de engenheiro na Edison Illuminating Company – mesmo sem possuir educação formal. Estes são apenas mais alguns exemplos do que pessoas com boas idéias e pessoas com vontade de apoiar boas idéias podem, juntas, realizar.

Nenhuma destas conquistas seria possível sem a coragem de inovar. Sempre será preciso ter pessoas com coragem para ousar e fazer a inovação, e pessoas com coragem para acreditar e apoiar a inovação. O trabalho intelectual jamais deve ser tolido, e ainda precisa ser constantemente incentivado. Porém, o verbo inovar é sucinto demais para sozinho expressar toda a complexidade do processo criativo. Este não começa apenas com um projeto, se forma ao longo de uma vida. Não é resultado exclusivo do esforço de um, nem de alguns, mas de toda uma civilização, através dos séculos de conhecimento e cultura acumulados. Por isso, não podemos esperar que a inovação "brote" da mente das pessoas: temos que aprender a cultivá-la.

A educação teve e continua a ter um papel fundamental no estabelecimento dos alicerces onde se apóiam os marcos levantados. A navegação portuguesa começou baseada no conceito de escola, a *Escola de Sagres* – ainda que esta nem mesmo existisse fisicamente –, o projeto de futuro de Meiji começou com o ensino de valores nacionais a todas as crianças do Japão, enquanto Ford, já consagrado, investiu socialmente na construção de várias escolas que enfatizavam uma educação vocacional.

E justamente aqui reside o valor da oportunidade única que esta Era nos oferta. Não há mais barreiras físicas, tecnológicas ou financeiras para nada, visto que todo o conhecimento do mundo e todas as pessoas que o dominam estão ao alcance de apenas alguns "cliques". A principal matéria-prima de toda a inovação (informação) é, mais do que nunca, abundante. Projetos podem ser executados por pessoas que nem mesmo se conhecem pessoalmente, em qualquer parte do planeta. As barreiras que restam são, portanto, humanas. Precisamos ter mais mecenas ávidos por encorajar artistas dispostos a fazer da pedra bruta sua obra-prima. Precisamos ter mais escolas e espaços propícios para fazer o tipo de transformação que o mundo mais precisa: a de idéias. Porém, mesmo na falta de tudo isto, pelo menos já podemos contar com tecnologias que permitem às pessoas, com conhecimentos e objetivos afins, cooperar em favor de um objetivo comum, não importando quem são ou onde estejam.

Assim, espero que este trabalho seja mais um pequeno marco, tanto como objeto quanto meio da disseminação de conhecimento. Esse é o tema que mais me fascinava antes de iniciá-lo e, sem dúvida, esse será também o grande avanço que marcará esta Era. Se apenas meu singelo objetivo tiver sido alcançado, então, este trabalho terá valido a pena.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

UM ESQUEMA DE VOTAÇÃO DINÂMICA DESCENTRALIZADO E TOLERANTE A FALHAS PARA REDES PEER-TO-PEER

Bruno da Rocha Braga

Novembro/2005

Orientadores: Jano Moreira de Souza

Geraldo Bonorino Xexéo

Programa: Engenharia de Sistemas e Computação

Aplicações Peer-to-Peer estão cada vez mais populares, ainda que restritas a um pequeno número de domínios de aplicação, em especial, o compartilhamento de arquivos. No entanto, ainda é difícil desenvolver aplicações para disponibilizar e consumir dados na rede, em face da inexistência de um conjunto de serviços fundamentais a exemplo daqueles prestados pelos frameworks de linguagens de programação e pelos bancos de dados tradicionais. Assim, este trabalho procurou projetar, implementar e testar modelos conceituais para suprir estas demandas e ainda chegou a uma proposta de um mecanismo de controle de concorrência descentralizado e tolerante a falhas silenciosas adequado ao ambiente instável das redes Peer-to-Peer.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

A DECENTRALIZED AND FAULT TOLERANT DYNAMIC VOTING SCHEME FOR PEER-TO-PEER NETWORKS

Bruno da Rocha Braga

November/2005

Advisors: Jano Moreira de Souza

Geraldo Bonorino Xexéo

Department: Systems and Computer Engineering

Peer-to-Peer applications are increasingly popular, despite restrict to a small number of application domains, in special, file sharing. Therefore it is still difficult to develop applications to provide and to consume data in the network, in face that there is no set of basic services as those provided by the current application frameworks and data banks. So, this work has designed, implemented and tested conceptual models to supply these demands and it has still proposed a decentralized and fault tolerant concurrency control mechanism adapted for the instable environment of Peer-to-Peer networks.

Sumário

Capítulo 1 – Introdução

1.1 – Cenário Atual

A classe de sistemas Peer-to-Peer (P2P) popularizou-se rapidamente na forma de aplicações de troca de mensagens instantânea (*Instant Messaging* ou IM), como o ICQ e o MSN Messenger, e de compartilhamento de arquivos (*File Sharing* ou FS), como Napster, Gnutella, e-Mule, Kazaa, iMesh, Morpheus. Muitas outras aplicações começam a surgir, tais como sistemas para *streamming* multimídia, computação distribuída, comércio eletrônico, entre outras; indicando que o conceito de P2P já está bem estabelecido e ganhando cada vez mais força.

No entanto, a maioria das aplicações consideradas Peer-to-Peer não possui uma legítima arquitetura Peer-to-Peer. Uma das principais características desta arquitetura de software é a ausência de controle centralizado, ou seja, de um ou mais servidores. Não é o caso, no entanto, de todas as aplicações de IM existentes, bem como da maioria das aplicações de FS citadas. Questões técnicas e comerciais levam a centralização de parte ou mesmo toda a operação destes sistemas. Porém, como todas estas aplicações se dispõem a atender um grande número de usuários, a descentralização da tarefa de coordenação e manutenção da rede P2P é um requisito importante do projeto. É claro, porém, que o grau de importância do mesmo depende da aplicação que se pretende desenvolver, visto que o impacto da centralização para sistemas de IM é pequeno — pelo contrário, esta abordagem facilita a implantação do mecanismo de monitoramento de presença dos usuários (ou peers) — enquanto é bem maior no caso dos sistemas de FS, considerando que é alta taxa de crescimento da oferta de arquivos compartilhados. Porém, não importando a finalidade da aplicação, um princípio básico da Internet se reflete nas redes P2P: controle centralizado implica na existência de um ou alguns pontos-de-falha, tornando possível a queda de todo ou parte do sistema, bem como se tornar o "gargalo" do desempenho do mesmo.

A arquitetura Peer-to-Peer se opõe diretamente à arquitetura Cliente-Servidor (C/S) tradicional. De fato, há vantagens na descentralização da tarefa de compartilhamento de recursos, de qualquer natureza, quando estes existem e crescem em grandes proporções. Estudos indicam que mesmo o mais completo dos engenhos de busca na Internet baseados em *Web Crawlers*[1] não é capaz de indexar mais que 16% dos documentos disponíveis publicamente na rede [Lawrence 1999]. A intenção da atualização desta base parte apenas do sistema de *crawler* e depende do relacionamento dos novos documentos com os anteriormente indexados — e, conseqüentemente, da profundidade alcançada no grafo de busca.

Sistemas de arquitetura P2P possuem como vantagens inerentes a sua arquitetura, a distribuição do armazenamento e a replicação de arquivos, a ausência de controle centralizado sobre o sistema, execução não-supervisionada de tarefas — quanto à transferência de arquivos, indexação, consulta e outros — e grande escalabilidade. Estas são as principais razões da intensa atividade de pesquisa e desenvolvimento de aplicações baseadas neste modelo.

Toda a atenção devida ao tema Peer-to-Peer é muito positiva, mas está gerando um novo tipo de problema: a falta de interoperabilidade entre as aplicações. Não apenas aplicações de categorias diferentes são incapazes de interoperar, mas também as de mesma categoria, como por exemplo, a base de arquivos compartilhados no Gnutella não estar disponível aos clientes Kazaa. Caminha-se na direção oposta do princípio que guiou o desenvolvimento da Internet: o estabelecimento de padrões de comunicação entre os nós da rede.

Uma das propostas de protocolos-padrão para redes P2P emergiu de um projeto da Sun Microsystems. O JXTA, acrônimo de *Juxtapose*, apresentado em abril de 2001 e re-lançado em maio de 2003 [Traversat 2003], é um conjunto de protocolos para a

[1] Programas que periodicamente percorrem a Web como um grafo direcionado, onde os nós são as home pages e as arestras são os links entre elas, tal que se torna possível indexar novas páginas a partir da referência incluída em páginas já indexadas anteriormente.

criação de uma camada de rede virtual *ad-hoc* sobre a infra-estrutura de rede existente, seja ela qual for. Tal rede virtual admite a troca de mensagens em XML entre peers, independentemente de sua localização real (atrás de firewalls, NATs, redes não baseadas em TCP/IP ou mesmo Ethernet). Estas mensagens são roteadas de maneira transparente, sobre diferentes protocolos de transporte ou transferência de dados, ignorando a topologia ou tecnologia da rede e permitindo que os nós da mesma (os peers) sejam móveis, ou seja, que sua identificação na rede não dependa de sua localização física — ao contrário do IPv4 atualmente em uso na Internet.

Apesar de desenvolvido pela Sun e originalmente em Java, o JXTA é um protocolo aberto e se candidata a padrão para redes P2P. De fato, ele foi concebido de modo a ser independente de plataforma de software ou hardware, bem como da finalidade da aplicação a ser desenvolvida, visto ser suficientemente genérico, portável e adaptável aos mais diferentes requisitos de projeto.

1.2 – Motivação

A motivação original que levou ao desenvolvimento de sistemas Peer-to-Peer é social. Idéia de computação colaborativa permeia aplicações de compartilhamento de arquivos, como um dos pioneiros, o Napster[2]. Problemas complexos que já foram resolvidos em sistemas baseados na arquitetura Cliente/Servidor (C/S) são ainda mais complexos de serem resolvidos em sistemas baseados na arquitetura Peer-to-Peer. Por outro lado, é evidente que para muitas aplicações, critérios como anonimato, resistência à censura e compartilhamento do controle da informação são imprescindíveis.

No entanto, questões técnicas também incentivam a pesquisa nesta área. Com o barateamento dos custos de hardware, qualquer PC possui mais recursos de processamento e armazenamento de informação que os antigos mainframes das décadas

[2] http://www.napster.com/

http://campus.acm.org/public/membernet/stories/September_2004/induce.pdf

de 70 e 80. Para muitos usuários, toda esta fartura de recursos computacionais acaba sendo subutlilizada. Ao mesmo tempo, em muitas organizações, a taxa de crescimento da demanda por recursos computacionais pode ser superior a de evolução da tecnologia ou mesmo a redução no período de renovação da planta de TI da mesma pode ser proibitiva em termos de custo/benefício.

Este enorme crescimento, em direções opostas, tanto da demanda como da oferta de espaço de armazenamento de informação, apenas contribui para o colapso das atuais estratégias de gerenciamento. É necessário um novo modelo de administração da informação. Em linhas gerais, precisamos saber:

- Onde há espaço disponível para armazenamento;
- Qual o melhor local para armazenar os dados (considerando capacidade de armazenamento, de processamento, latência do canal de comunicação, segurança, custo total da solução, etc);
- Qual o grau de disponibilidade desta informação será necessário, e como garantir que ele seja alcançado através da replicação dos dados;
- Como determinar com precisão onde se encontra a informação procurada e como garantir que ela não se perca, ou melhor, que não percamos a sua referência;
- Como realizar o versionamento e a consistência entre as réplicas.

Os rumos que a pesquisa em banco de dados tradicional seguiu nas três últimas décadas influenciaram fortemente a pesquisa em busca e recuperação de informação distribuída. De fato, em redes Peer-to-Peer, a grande meta pode ser sintetizada como a disponibilização de um banco de dados distribuído entre os peers. Porém, os requisitos de outros ramos de pesquisa em redes P2P impõem a revisão de alguns conceitos típicos a respeito de bancos de dados, como é o caso de computação desconexa e gerência de dados em redes Peer-to-Peer de larga escala.

Note ainda que a própria tecnologia Peer-to-Peer adiciona novos integrantes ao conjunto de itens que tradicionalmente precisamos gerenciar. Não são apenas arquivos

ou dados a serem gerenciados, mas também peers, grupos, componentes de software a até a própria rede, ou seja, os próprios recursos responsáveis por explorar as capacidades da tecnologia e prover estes novos serviços idealizados. Logo, como será possível gerenciar tanto o conhecimento distribuído quanto o próprio sistema que almeja nos prover toda esta funcionalidade?

Busca e Recuperação de Informação Distribuída

Pesquisas como Chord [Stoica 2001], CAN [Ratnasamy 2002] e Tapestry [Zhao 2001] foram pioneiras no roteamento de consultas entre peers de maneira eficiente e escalável, através de mecanismos de indexação distribuída probabilísticos (*hashing*). Estes trabalhos abriram caminho para a pesquisa de busca e recuperação de informação em redes Peer-to-Peer, ainda que sobre uma base de informações imutável. As primeiras aplicações destas tecnologias também se limitaram à busca através de palavras-chave. Assim, estas duas características limitantes deram origem a dois grandes ramos de pesquisa em sistemas Peer-to-Peer.

Metadados e Ontologias para Lidar com o Conhecimento

Os conceitos de metadados e de ontologias têm sido as abordagens mais comuns nos projetos de gerência de conhecimento, independente do domínio de aplicação ou tecnológico adotado. Em redes Peer-to-Peer, os projetos U-P2P [Mukherjee 2002], JXTA CMS e JXTA Search [Waterhouse 2001] propõem o uso de metadados para descrição de recursos compartilhados, mas foi o U-P2P[3] que apontou a necessidade de gerência dos próprios metadados como meio de facilitar a interação humana com o sistema. Tal conceito de gerência de metadados, como será mostrado adiante, se assemelha ao conceito de gerência de classes dos objetos armazenados em um banco de dados. Assim como ocorre em várias linguagens de programação orientadas a objetos, podemos considerar a própria classe como uma instância de uma metaclasse. O projeto

[3] O projeto U-P2P foi implementado utilizando Web Services e apenas simula uma rede P2P, visto que utiliza uma abordagem centralizada, o UDDI, na localização dos peers.

JXTA CMS é focado na transferência de arquivos sobre a plataforma JXTA e lida com um único pequeno metadado para descrição destes arquivos. O projeto JXTA Search é mais adaptado à realidade de redes Peer-to-Peer, pois concentra em um peer especial (chamado *hub*) uma espécie de resumo ou descrição genérica de anotações sobre recursos com alguns campos do metadado em comum e, com isso, reduz o espaço de busca consideravelmente. Vale observar, porém, que a decisão de como serão essas descrições genéricas é em nível de projeto e, dessa forma, também é possível prover uma gerência de metadados como proposta pelo U-P2P, especificando um metadado para os metadados.

Os projetos Edutella, [Arumugam 2001] e SWAP [Ehrig 2003] são propostas para gestão de conhecimento em redes Peer-to-Peer usando ontologias. O Edutella nasceu da idéia de prover anotações para conteúdo educacional provido pelos diversos departamentos, de maneira independente, e dentro de um ambiente universitário. A aplicação de ontologias neste projeto é bastante similar aos projetos usando metadados, pois se limita à descrição de material educacional compartilhado na rede P2P. Porém, a vantagem do uso de ontologias vem de sua base teórica em lógica formal que permitiu o estabelecimento de cinco níveis de abstrações lógicas com poder de expressão crescente e aplicável conforme a necessidade de refinamento da consulta a ser realizada. O InfoQuilt provê um framework para a formulação de requisições mais complexas, envolvendo múltiplas ontologias e o suporte à descoberta de conhecimento na base. Ontologias menos acessadas ainda sofrem uma espécie de seleção natural e podem desaparecer da base com o tempo. Já o projeto SWAP, conduzido por um consórcio de universidades européias, apesar de não trazer muitas inovações, visa aplicar o conhecimento gerado em outros projetos de pesquisa e fornecer um framework de alta qualidade para gestão de conhecimento usando ontologias, mas ainda é bastante focado no desenvolvimento de processos de gestão de conhecimento distribuído.

O que existe de comum em todos os projetos estudados é o foco em processos de gestão do conhecimento e descrição de arquivos compartilhados. Note que consideramos a anotação e compartilhamento de recursos computacionais distribuídos em uma rede P2P um caso específico de gestão de conhecimento distribuído, visto que

se trata apenas da associação de recursos computacionais (arquivos, web services, componentes, etc) a objetos de informação[4] de alguma natureza.

Mesmo com o aumento da expressividade advindo do uso de ontologias, este benefício acarreta em maior carga de processamento durante a busca e, dependendo dos requisitos da aplicação a ser desenvolvida, este pode ser um fator que impossibilite a utilização deste tipo de tecnologia para a gerência de uma grande quantidade de itens de informação. Ciente da ocorrência deste problema em bases de conhecimento de larga escala, a maior delas a própria *Web Semântica* [Berners-Lee 2001], o projeto SHOE [Heflin 2000] propõe uma pequena extensão à linguagem HTML que permite autores inserirem anotações para interpretação via software. Ainda que seja uma solução adequada para a implementação da Web Semântica em engenhos de busca (como o Google), SHOE não aborda o problema da distribuição de dados para o consumo de aplicações e a ainda carece de poder de expressão. Segundo estudos divulgados em [Lawrence 1999], o uso de metadados é pouco disseminado, com cerca de 30% das páginas utilizando *metatags* da HTML e apenas 0.3% utilizando metadados no formato Dublin Core. Claro que no caso de redes Peer-to-Peer, como o processamento da consulta propriamente também é distribuído, é possível ousar atribuir um maior nível de detalhamento nos metadados utilizados.

Por fim, nenhum dos projetos citados anteriormente foi criado para tratar a questão de manutenção da consistência entre réplicas de um recurso compartilhado, tal como é o caso da distribuição de bases de dados para processamento por aplicações. Se a quantidade de itens de informação indexados for tão grande a ponto de não admitir o uso de ontologias, certamente, a manutenção da consistência entre réplicas será também crítica. Assim, notamos a demanda por uma infra-estrutura para uma efetiva gerência de objetos de informação em redes Peer-to-Peer.

---

[4] Objeto de Informação é o termo adotado neste trabalho para os itens de informação estruturada a serem consultados, modificados e gerenciados. São formados por um metadado e uma estrutura de dados (*payload*). Neste trabalho, tais objetos podem ou não estar associados a um recurso computacional, quando neste caso, tem a função de descrevê-lo. Técnicas comuns de implementação deste conceito são metadados XML, ontologias e objetos de conhecimento (*Knowledge Objects*).

Gerência de Dados Distribuídos em Redes Peer-to-Peer

A necessidade apontada não é apenas de gerência de objetos de informação ou de dados, mas também de eficiência na consulta, modificação e manutenção de consistência desta base. É inegável a importância de ontologias e metadados para a lide humana com o seu conhecimento, porém, entendemos que a melhor abordagem na migração destas técnicas para um ambiente totalmente descentralizado é a de primeiro desenvolver uma efetiva gerência descentralizada de dados distribuídos para servir de infra-estrutura para um sistema de gestão de conhecimento disperso na rede.

Em [Gribble 2001] encontramos um estudo focado na definição do problema de alocação de objetos em peers da rede em função de algum objetivo a ser alcançado (disponibilidade, desempenho em consulta, desempenho em transferência, entre outros). Os autores destacam que este problema em redes Peer-to-Peer é similar ao de Banco de Dados Distribuídos; exceto no fato de que não se pode contar com uma administração central ou esquema unificado. Portanto, a idéia a ser perseguida é o provimento de um sistema de gerência de dados distribuídos, sem coordenação central, que se aproxime dos requisitos clássicos de sistemas gerenciadores de banco de dados e seja autogerenciado, não importando o modelo de dados utilizado.

Os protótipos existentes com esse perfil, de sistemas de gerência de dados distribuídos para redes Peer-to-Peer, podem ser divididos em dois grupos: os de *larga escala* e os de *pequena escala*. O primeiro grupo visa à gerência de dados em muitos peers dispersos globalmente. Considerando a alta latência dos canais de comunicação, a baixa disponibilidade dos peers e o conseqüente dinamismo da configuração da rede; o desafio para alcançar os requisitos de bancos de dados é consideravelmente maior. Os principais projetos de larga escala encontrados, o CFS [Kaashoek 2001], OceanStore [Rhea 2003] e o APPA [Akbarinia 2004] são bastante recentes e não cumprem todos os requisitos de um SGBD.

A abordagem de pequena escala assume um número limitado de peers e alguma garantia de qualidade de serviço de comunicação entre os peers como pré-requisitos para alcançar os objetivos estabelecidos. Exemplos de pesquisas nesta área são o Farsite [Bolosky 2000] e Oasis [Rodrig 2003].

Projeto de Gerência de Dados Distribuídos em Larga Escala	Projeto de Gerência de Dados Distribuídos em Pequena Escala
APPA	COPPEER
CFS	Farsite
OceanStore	Oasis

Tabela 1 – Projetos de pesquisa classificados nas duas grandes áreas identificadas.

1.3 – Objetivos da Pesquisa

Esta dissertação de mestrado seguiu a abordagem de pequena escala, conforme o projeto Oasis, da Stanford University. O que pretendemos no início do trabalho era projetar um sistema em que um número limitado e dinâmico de participantes pudesse colaborar com o repositório distribuído de informações — quando e como preferissem —, que o estabelecimento e manutenção desta base de dados distribuída fossem automáticos e ainda que o processamento das consultas, inclusive para modificação dos dados, fosse transparente ao usuário — fazendo se passar por um único banco de dados centralizado.

O COPPPER, um framework para desenvolvimento de aplicações Peer-to-Peer, surgiu da constatação da dificuldade em se programar aplicações utilizando o JXTA, dado o baixo nível de abstração de suas funcionalidades. Este fato foi constatado durante a implementação do protótipo desta dissertação, e também motivou o desenvolvimento de outros projetos semelhantes como JAL/EZEL[5], JXTA XML-RPC[6] e JXTA SOAP[7].

[5] http://ezel.jxta.org

No entanto, o objetivo do trabalho, desde o início, era permitir que recursos informacionais, replicados em nós da rede, pudessem ser atualizados, e tais atualizações propagadas por todas as réplicas da rede. Para tanto, focamos no projeto de um algoritmo de controle de concorrência descentralizado e tolerante a falhas que cumprisse os requisitos de seus similares implantados em sistemas gerenciadores de bancos de dados federados. Assim, a consistência da base de dados deveria ser garantida em qualquer hipótese.

Os conceitos desenvolvidos, no entanto, encontram outros campos de aplicação: a gerência de transações em frameworks de desenvolvimento de objetos distribuídos; repositórios descentralizados de arquivos e a edição colaborativa de documentos ou ontologias. Este último, inclusive, serviu de prova-de-conceito tanto do modelo de arquitetura de aplicação P2P quanto para o serviço de gerência de transação propostos e implementados no protótipo de COPPEER.

1.4 – Organização da Dissertação

No **capítulo 2**, discutimos as redes Peer-to-Peer, as abordagens para indexação e busca de informação compartilhada — atualmente, a mais popular aplicação da tecnologia P2P — e apresentamos o JXTA, a plataforma de redes Peer-to-Peer adotada para a implementação do sistema proposto. No **capítulo 3**, apresentamos o conceito de federação de banco de dados e mostramos o porquê dele servir de referência para sistema de Gerência de Dados Distribuídos proposto. No **capítulo 4**, é apresentado o principal resultado desta pesquisa, uma proposta de mecanismo de controle de concorrência adaptado ao cenário de gerência de dados distribuídos em pequena escala para redes P2P. No **capítulo 5**, apresentamos os detalhes de implementação do COPPEER propriamente dito – o framework para desenvolvimento de aplicações Peer-

[6] http://xmlrpc.jxta.org

[7] http://soap.jxta.org

to-Peer construído sobre a plataforma JXTA. No **capítulo 6**, refletimos sobre os resultados desta pesquisa bem como apontamos direções para trabalhos futuros em desenvolvimento de sistemas distribuídos e gerência de dados em redes Peer-to-Peer.

Capítulo 2 – Redes Peer-to-Peer e o JXTA

Redes Peer-to-Peer têm atraído muita atenção de pesquisadores, da indústria e mesmo de usuários leigos. Entretanto, tal conceito não costuma ser definido com exatidão. De acordo com o Webopedia[8], redes Peer-to-Peer são "*um tipo de rede de computadores onde cada estação possui capacidades e responsabilidades equivalentes; o que difere da arquitetura cliente/servidor, no qual alguns computadores são dedicados a servirem dados a outros*". Esta definição, porém, ainda é demasiado sucinta para representar tudo o que existe em termos Peer-to-Peer. Portanto, neste capítulo, vamos explorar o conceito e mostrar um panorama geral dos trabalhos nesta área.

2.1 – Histórico

Geralmente, uma rede Peer-to-Peer (P2P) é constituída por computadores ou outros tipos de unidades de processamento que não possuem um papel fixo de cliente ou servidor, pelo contrário, costumam ser considerados de igual nível e assumem o papel de cliente ou de servidor dependendo da transação sendo iniciada ou recebida de um outro peer da mesma rede. Assim, o contraste natural deste tipo de arquitetura é com o modelo cliente/servidor.

Os nós da rede Peer-to-Peer podem diferir em termos de configuração local, capacidade de processamento, capacidade de armazenamento, largura de banda, entre outras características particulares. O primeiro uso da expressão "Peer-to-Peer" foi em 1984, com o desenvolvimento do projeto *Advanced Peer to Peer Networking Architecture* na IBM.

O termo é utilizado em diferentes tecnologias que adotam um modelo similar ao descrito, tal como o protocolo NNTP (para Usenet News), SMTP (para envio de e-

[8] http://www.webopedia.com

mail), e Instant Messaging (ICQ, MSN). Porém, o termo tornou-se popular com o surgimento de aplicações de compartilhamento de arquivo (*file sharing*). Em 1999, Shawn Fanning criou o Napster[9], e trouxe o conceito de Peer-to-Peer para a mídia.

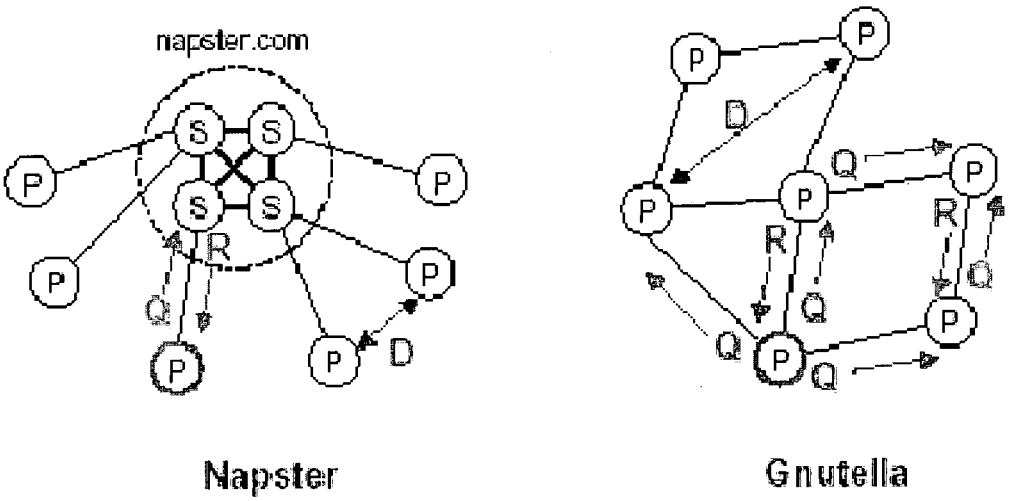


Figura 1 – Topologias Peer-to-Peer Centralizada e Totalmente Descentralizada.

O Napster e outras aplicações intituladas Peer-to-Peer, tais como o IRC e o ICQ, são baseadas em uma arquitetura cliente/servidor, pelo menos para algumas tarefas críticas, como indexação de informação. Por outro lado, redes como Gnutella e Freenet usam uma arquitetura Peer-to-Peer pura, sem nenhuma centralização de tarefas (Figura 1). Aparentemente mal empregado, o termo Peer-to-Peer, quando usado para aplicações como o Napster, ressaltam a característica de incremento na importância do papel exercido pelos nós da rede. Estes passam a servir como provedores de informação, e não apenas consumidores passivos; ainda que de acordo com pesquisadores do Xerox PARC, 70% dos usuários não contribuem com nenhum arquivo, enquanto 1% dos peers é responsável por 50% dos arquivos disponíveis.

Aplicações Peer-to-Peer "puras" são raras. A maioria das arquiteturas Peer-to-Peer é híbrida, utilizando alguns elementos centralizadores na execução de tarefas cujo desempenho é crítico. Redes completamente descentralizadas já foram usadas anteriormente em aplicações de propósito específico, como a USENET (1979) e

[9] http://www.napster.com

FidoNet (1984). Porém, as questões de desempenho induzem a uma centralização parcial das atividades em peers de maior capacidade. Outras técnicas têm sido desenvolvidas visando o aperfeiçoamento de sistemas Peer-to-Peer.

Há importantes projetos de pesquisa na área, alguns dos quais já bastante populares: o Chord, uma tabela de Hash distribuída; o P-Grid, uma rede auto-organizável que suporta distribuição arbitrária de índices, load-balancing e roteamento eficiente de consultas; e CoopNet, um sistema de distribuição de conteúdo. Além disso, os sistemas Peer-to-Peer devem expandir suas fronteiras para além dos microcomputadores, alcançando PDAs e telefones celulares com acesso à Internet.

2.2 – A Plataforma JXTA

Diversas plataformas para o desenvolvimento de aplicações Peer-to-Peer já foram desenvolvidas, cada uma com seu próprio foco, entre as quais podemos citar o Pastry [Druschel 2001] e XNap [Berger 2003]. Porém, o que parecia mais promissor no momento que iniciamos este trabalho era o JXTA.

JXTA (do inglês *juxtapose*) é uma especificação independente de linguagem e plataforma para comunicação entre dispositivos sem considerar sua localização física e tecnologia de rede no qual se encontram instalados.

A plataforma JXTA foi especificada na forma de uma arquitetura *microkernel*, ou seja, mesmo os serviços mais básicos estão implementados como módulos, restando para o kernel propriamente dito poucas funções, em geral, de comunicação entre os próprios módulos. Portanto, serviços básicos como a descoberta de peers ou de recursos na rede, comunicação entre dois ou mais peers, entre outros, todos são providos por módulos específicos.

A seguir, apresentamos os conceitos, protocolos e serviços do JXTA *core*, ou seja, a plataforma básica do JXTA. Sobre eles (ou ao lado deles), são construídos outros

serviços, alguns dos quais muito importantes para o desenvolvimento de aplicações Peer-to-Peer.

2.2.1 – Conceitos Básicos

Aplicações desenvolvidas para suportar o trabalho em grupo devem, obviamente, organizar seus usuários em *grupos*. Os grupos podem ser organizados segundo diferentes critérios e esta, bem como outras atividades, é parte do projeto da aplicação colaborativa. Para tornar esta aplicação realidade, devemos ter entidades na plataforma JXTA para os quais possam ser mapeados os principais conceitos de computação distribuída. Por isso, nesta seção, detalharemos o que constitui uma rede P2P na visão do JXTA.

2.2.1.1 – Peer e Peer Group

Uma rede virtual JXTA consiste de alguns tipos de peers, sendo que um peer conectado a rede pode, em teoria, assumir qualquer um destes papéis descritos a seguir:

- **Edge Peers** — São os peers simples, podendo tanto ser computadores desktop, conectados por uma LAN ou modem à Internet, e outros dispositivos computacionais.

    - **Minimal Peers** — Dispositivos com restrições de recursos, como celulares e palms, são chamados *minimal peers* e, geralmente, não possuem toda a funcionalidade disponível pela plataforma JXTA aos peers.

    - **Proxy Peers** — Peers instalados em computadores que realizam funções de proxy (para mensagens da plataforma JXTA) para minimal peers que não possuem endereço IP; para peers que, mesmo possuindo IP, não

podem realizar operações intensivas em termos de recursos computacionais; ou para peers localizados atrás de um firewall — neste caso, todas as requisições são transmitidas através de HTTP.

- **Rendezvous Peers** — Papel que costuma ser assumido por peers com maior poder computacional, com endereço IP fixo, que atuam como cache de informação (advertisements) sobre os peers conectados, facilitando a descoberta de recursos e provendo operações de resolução, tal como resolução de nome de peer para endereço IP.

- **Relay Peers** — Peers que adquirem informação de roteamento, bem como realizam passagem de mensagens para outros peers atrás de um firewall, um NAT ou, simplesmente, através de roteador. Normalmente, os papéis de Rendezvous e Relay são assumidos por um mesmo peer.

Peers se organizam em grupos (*Peer Groups*). A especificação não define o que esses grupos são ou porque eles existem. Porém, em geral, grupos são usados para definir um conjunto de serviços e recursos, prover uma região de acesso controlado, criação de escopo, monitoração de membros, entre outras aplicações deste conceito.

2.2.1.2 – Advertisements

Todas as entidades da plataforma JXTA, incluindo peers, groups, pipes e serviços, são representadas usando *advertisements*; documentos XML bem formados contendo informação à respeito dessas entidades (um metadado). Todas as entidades possuem um ID único e universal, além de informações adicionais específicas. Quando um recurso é disponibilizado na rede, na verdade, é porque seu advertisement foi criado e publicado, ou seja, enviado para os demais peers on-line.

Um advertisement possui um tempo de vida (*lifetime*) que visa evitar descrever entidades que já não existem na rede; uma vez que advertisements podem ser

armazenados nos caches locais dos peers. A cada inicialização de um peer, os advertisements expirados são eliminados.

Note que um advertisement só existe enquanto algum peer on-line armazenar uma cópia do mesmo. Logo, o recurso associado a este advertisement só estará disponível se este também estiver.

A plataforma JXTA define seis adverstisements básicos: Peer, Peergroup, Pipe (canal virtual de comunicação ponto-a-ponto), Service (abstração para serviço oferecido por um Peer ou Peergroup), Content (abstração para conteúdo publicado) e Endpoint (pontos de conexão de um pipe).

2.2.1.3 – Pipes e Messages

Peers transmitem mensagens apenas através de pipes, canais virtuais que são, em geral, unidirecionais e não-confiáveis, anexáveis a um ponto de entrada e outro de saída (*end points*). Pipes possuem IDs únicos, e não são associados a nenhum dispositivo de rede real, havendo um serviço de resolução de IDs para dispositivos de rede. Também estão disponíveis pipes bi-direcionais ou confiáveis, implementados sobre os pipes convencionais.

Mensagens são documentos XML bem formados, que possuem roteamento baseado no ID da fonte, carregando em seu cabeçalho (header) a informação de roteamento necessária, tal como a seqüência de peers a ser percorrida.

2.2.2 – Protocolos Básicos

São sete os serviços básicos fornecidos pela plataforma JXTA e, a princípio, providos por qualquer peergroup criado: *Discovery*, *Membership*, *Access*, *Pipe*, *Resolver*, *Monitoring* e *Rendezvous*.

2.2.2.1 – Peer Discovery Protocol

Peers utilizam este protocolo para descobrir recursos do JXTA dinamicamente. Em uma rede IP, a implementação deste protocolo consiste de duas tarefas: o envio de uma mensagem multicast através da rede local do peer e de Rendezvous Peers para a descoberta de peers além da rede local. Este protocolo é implementado pelo *Discovery Service*.

Alguns peers presentes na rede podem não responder uma mensagem de consulta a um recurso, pois o protocolo é não-confiável. Felizmente, quanto mais peers existirem na rede, mais rápida será a descoberta dos mesmos, pois quando um peer responde a consulta, ele envia todos os advertisements relacionados a mesma que ele tenha descoberto anteriormente.

Os Rendezvous Peers são usados para armazenar advertisements de recursos que ele conhece, incluindo peers. Alguns Rendezvous são providos pela própria Sun com o objetivo de permitir a localização de Rendezvous dinâmicos, ou seja, aqueles que assumem dinamicamente este papel. Cada grupo pode, inclusive, fixar uma taxa de Rendezvous presentes, caso sirva a algum propósito específico.

2.2.2.2 – Peer Resolver Protocol

Permite o envio de uma consulta genérica a outros peers (unicast ou multicast). Este protocolo serve de infra-estrutura para outros protocolos do JXTA, tais como o Peer Information (PIP) e o Peer Discovery (PDP).

2.2.2.3 – Peer Information Protocol

Coleta informações sobre o estado de um peer, sendo útil para *accounting* (para consumo de serviços providos), monitoramento de desempenho da rede, execução de algoritmos que baseadois em informação global, entre outras aplicações. O PIP provê

uma funcionalidade de *polling* para checar se um dado peer está on-line, bem como solicitar o seu advertisement.

2.2.2.4 – Rendezvous Protocol

É o protocolo responsável por propagar mensagens dentro de um grupo e controlar esta propagação, bem como permitir a conexão a serviços. O RVP é base para dois outros protocolos: o Peer Resolver (PRP) e o Pipe Binding (PBP).

2.2.2.5 – Pipe Binding Protocol

É o protocolo responsável por conectar um *pipe* a seus dois *endpoints*. Uma mensagem de consulta é enviada pela rede para encontrar um pipe endpoint já conectado ao pipe desejado.

2.2.2.6 – Endpoint Routing Protocol

Estabelece um conjunto de mensagens de busca usadas para encontrar informações de roteamento, antes da execução do envio de uma mensagem entre peers. As rotas encontradas são armazenadas localmente, e incluem informações sobre o Peer ID do remetente, Peer ID do destinatário, o *time-to-live* (TTL) e a seqüência ordenada de peers na rota.

2.2.2.7 – Membership Protocol

Utilizado para serviços de validação de peers para entrada em grupos. A implementação padrão deste protocolo é precária, consistindo de uma senha única para entrada no grupo.

2.2.3 – Principais Projetos em JXTA

Nesta seção, apresentamos os principais projetos em produção relacionados a gerência de informação desenvolvidos sobre a plataforma JXTA na data da elaboração deste texto. A idéia é fornecer um panorama geral sobre os serviços disponíveis para utilização em aplicações P2P.

2.2.3.1 – JXTA CMS

O *Content Manager Service*[10] é um serviço que permite às aplicações sobre a plataforma JXTA compartilhar, buscar e recuperar arquivos dentro de um mesmo Peer Group. Cada item compartilhado é representado por um ID único (*Content Identifier*) e possui um advertisement (*Content Advertisement*) provendo meta-informação sobre o seu conteúdo, mais especificamente, seu nome, tamanho, tipo MIME e descrição. A busca por conteúdo é realizada apenas por palavra-chave, sobre o domínio dos quatro atributos do metadado definido para o CMS — abordagem similar às adotadas pelas principais aplicações de *file sharing*, como Napster, Gnutella e Kazaa.

Apesar de ainda prover um serviço de transferência de arquivos eficiente e confiável, a funcionalidade de busca do CMS original é extremamente simples, e não tratou importantes quesitos de alta demanda atualmente, tais como:

- Utilização de diferentes metadados para descrição de diferentes grupos de arquivos semelhantes entre si.

[10] http://cms.jxta.org

- Indexação distribuída, um problema tratável através de busca por palavras-chave usando a estrutura de dados *Distributed Hash Table*[11] (DHT), mas que não foi implementada no CMS original. Assim, não é possível realizar o roteamento de consultas na base de metadados de um grupo.
- Estabelecimento de relacionamentos entre os metadados dos arquivos indexados, bem como de semântica para os atributos; o que se resume à utilização de uma *Ontologia* para descrição dos recursos.

O identificador único associado a cada arquivo é gerado por uma função de hash de 128 bits chamada MD5 [Rivest 1992]. A vantagem no seu uso reside no fato de um mesmo arquivo costuma ser compartilhado por vários peers que, por sua vez, têm total liberdade para mudar o nome do arquivo e sua descrição. Assim, com o ID de 128 bits, é seguro afirmar que dois arquivos de mesmo tamanho e ID são iguais. Tal recurso é também útil para identificar as várias fontes (peers) de um determinado arquivo encontrado e viabiliza o download de diferentes partes do mesmo apartir de cada fonte distinta — no entanto, isto ainda não é suportado pelo CMS.

Em [Xiang 2003] foi apresentada uma extensão ao serivço CMS básico que permite a busca por conteúdo usando um metadado fixo baseado no padrão Dublin Core [Hillmann 2003], que adota 15 atributos considerados necessários para a descrição genérica de arquivos compartilhados. O Metasearch foi implementado como uma nova camada (um novo serviço) sobre o CMS original.

2.2.3.2 – JXTA Search

[11] Um método de roteamento em redes baseado em tabelas hash que perimitem a identificação e recuperação de informação em sistemas distribuídos, tais como redes Peer-to-Peer. A tabela em si é distribuída através dos nós que fazem parte da rede. Projetos de pesquisa que em DHT incluem Freenet, Chord, The Circle e P-Grid.

O JXTA Search [Waterhouse 2001] consiste do principal sistema de busca e indexação de informações disponível para a plataforma JXTA. Nele, alguns peers especiais (chamados *Hub*) assumem o papel da indexação dos metadados (em XML) enviados pelos demais peers participantes de um grupo. Tais metadados constam apenas uma descrição genérica — por exemplo, apenas a estrutura do metadado, ou apenas com alguns campos preenchidos — do que está indexado em um peer particular servindo, portanto, de uma orientação para o roteamento da consulta.

No JXTA Search, um XML Schema designa a um *Query Space*, ou seja, arquivos descritos por instâncias deste esquema podem ser consultados segundo valores para os atributos do descritor. O problema desta abordagem é que ela é muito rígida: se precisarmos estender um descritor XML, ele terá um outro esquema, com outro *namespace*, e não haverá nenhuma associação entre ele e seu esquema-pai. Neste caso, a busca também pode ser feita por predicados com o mesmo nome, mas isso é péssimo, por exemplo, o predicado *nome* está presente em quase todos os modelos. Pior ainda, pode haver predicados com mesmo nome e semânticas diferentes. Outras deficiências desta abordagem são: pequeno poder de expressão das consultas; ausência de suporte para tratamento semântico da informação; e inadequado para aplicações que modificam os dados constantemente.

2.2.3.3 – Edutella

O Edutella [Nejdl 2002] é o segundo principal projeto de busca e recuperação de informação em JXTA e, provavelmente, o maior projeto já desenvolvido usando esta plataforma. Este projeto buscou suprir as deficiências apontadas para o JXTA Search, principalmente na questão de agregar semântica à informação. Porém, acabou se adequando a um domínio de aplicação diferente do JXTA Search.

O Edutella nasceu da idéia de prover *anotações* para conteúdo educacional provido pelos diversos departamentos, de maneira independente, e dentro de um ambiente universitário. A aplicação de ontologias neste projeto é bastante similar aos projetos usando metadados, pois se limita à descrição de recursos compartilhados na

rede P2P, mas a vantagem do uso de ontologias vem de sua base teórica em lógica formal, o que permitiu o estabelecimento de cinco níveis de abstrações lógicas com poder de expressão crescente e aplicável conforme a necessidade de refinamento da consulta a ser realizada.

O Edutella especificou um serviço de consulta e um modelo de dados, chamados de ECDM. O mecanismo de consulta (*Query Service*) do Edutella foi projetado para tornar as bases acessíveis como uma base isolada ou parte de uma base distribuída. Um repositório RDF (*Knowledge Base*) consiste de *RDF statements* — fatos, do predicado *statement* — que descrevem metadados em esquemas RDF-S.

O Edutella ainda descreve cinco modelos de linguagem de consulta, com níveis crescentes de expressividade:

- **RDF-QEL-1** — Consultas constituídas de fórmulas conjuntivas, representáveis como um grafo direcionado (*RDF statements*).
- **RDF-QEL-2** — Consultas constituídas por disjunção de conjunções, representáveis como uma família de grafos ou por meio da notação UML-like do *Conzilla Query Interface* (especialização/OR, agregação/AND) que é mais amigável.
- **RDF-QEL-3** — Consultas constituídas por disjunção, conjunção e negação. Esta linguagem compreende as características da Álgebra Relacional. Podem ser expressas em Datalog[12].
- **RDF-QEL-4** — Consultas podem expressar também Fecho Transitivo (*Transitive Closure*) e recursividade linear; sendo, portanto, compatível com a SQL3 (Relacional-Objeto). Podem ser expressas em Datalog.

[12] Datalog é uma linguagem lógica, subconjunto da Prolog, que compreende todos os princípios da álgebra relacional. Datalog estende linguagens de consulta à banco de dados, como SQL, ao permitir definições de consulta recursivas usando o princípio de fecho transitivo. Datalog restringe Prolog ao não permitir termos funcionais e ao impor restrições de estratificação (sobre o uso de negação e recursão) visando à unicidade de interpretação; garantindo com isso a existência de um algoritmo de resolução bottom-up — que termina em tempo finito.

- **RDF-QEL-5** — Recursividade estratificada ou dinamicamente estratificada.

O *Edutella Query Service* permite que os peers consultem ou registrem as consultas que eles são capazes de responder, fornecendo os esquemas de metadados que eles suportam, ou apenas propriedades individuais destes esquemas, ou ainda pares do tipo (propriedade, valor). De qualquer maneira, esta publicação reflete os arquivos que o peer detém pela descrição de seus conteúdos. As consultas trafegam pela rede P2P até um subconjunto de peers que anunciaram, previamente, possuir conteúdo com descrição compatível com a desejada.

O ponto fraco deste projeto é não ter explorado muito o potencial da tecnologia Peer-to-Peer. As consultas são direcionadas a um único peer escolhido da lista de peers on-line provida pelo *Discovery Service*, quando poderiam ser direcionadas a todos os peers por *broadcasting*.

2.3 – Conclusão

As pesquisas desenvolvidas a área de sistemas Peer-to-Peer e os projetos baseados na plataforma JXTA nos permitiram identificar as demandas dos usuários e dos desenvolvedores dos mesmos. Apesar do JXTA já prover meios para interoperabilidade de aplicações em plataformas, tecnologias de redes e propósitos distintos; tornou-se clara a necessidade de prover abstrações de maior nível para o processo de especificação e implementação das aplicações Peer-to-Peer.

Adicionalmente, a interoperabilidade de aplicações não é suficiente se nos depararmos com um universo de aplicações implementadas como programas isolados. Não é eficaz manter e utilizar inúmeras aplicações em um computador, bem como é desperdiço de recursos humanos e financeiros recriarem determinados requisitos não-funcionais, comuns e ao mesmo tempo indispensáveis, como a validação de usuário, busca de peers, grupos e recursos na rede, entre outras. Logo, há uma demanda para um

framework para desenvolvimento e execução de aplicações Peer-to-Peer, a exemplo do que já existe para sistemas distribuídos cliente/servidor.

Este trabalho também procurou atender esta demanda através da proposta de um framework básico, como veremos no capítulo seguinte.

Capítulo 3 – Sistemas de Gerência de Dados Distribuídos para Redes P2P

Neste capítulo, enunciaremos as diretrizes e desafios de um projeto desta categoria, estabelecendo um contexto para a proposta: um sistema de Gerência de Dados Distribuídos, com coordenação descentralizada, para redes Peer-to-Peer.

3.1 – Definição

Um sistema de Gerência de Dados Distribuídos (*Distributed Data Management*) com coordenação descentralizada é aquele que administra não somente dados distribuídos em um conjunto de computadores como também distribui a responsabilidade desta administração entre eles. Normalmente, tais dados são organizados em unidades estruturadas, descritas através de metadados, levando-nos a defini-los também como objetos de informação. Assim, neste texto, nos referimos à gerência de dados, de objetos de dados e de objetos de informação de forma intercambiável.

3.2 – Federações de Banco de Dados em Redes Peer-to-Peer

As tecnologias tradicionais de federação e de distribuição de bases de dados foram possíveis graças à solução de alguns problemas técnicos tais como Decomposição de Consultas, Localização dos Dados, Gerência de Transação, Controle de Concorrência, Manutenção da Consistência e Segurança.

Há pelo menos três características principais que distinguem os tradicionais sistemas de federação de banco de dados de seus similares a serem implementados sobre uma arquitetura Peer-to-Peer:

- A arquitetura de coordenação, outrora centralizada em pelo menos algumas partes críticas dos algoritmos utilizados, deve ser descentralizada entre os participantes.

- A não-confiabilidade dos servidores participantes da federação, que exige o desenvolvimento de mecanismos mais fortes para a garantia de consistência da base de dados distribuída. Entre os fatores que influenciam a confiabilidade, podemos citar o período de disponibilidade (atividade), a alta rotatividade no conjunto de participantes ativos, e a inconsistência e a incorreção nos dados de uma partição da base localizada em um determinado peer.

- A maior latência[13] e *jitter*[14] dos canais de comunicação entre os peers, que requer a minimização das mensagens trocadas entre eles no contexto de execução do sistema de Gerência de Dados Distribuídos.

Neste trabalho, nos concentramos no problema de *Gerência de Transação*, mais precisamente, *Controle de Concorrência* e *Manutenção da Consistência*, que são os que sofrem maior impacto destas novas características da infra-estrutura de rede.

Note ainda que ao usarmos o termo base de dados, não estamos nos atendo a nenhum modelo de dados específico (relacional, orientado a objetos, dados semi-estruturados, ontologias, etc), visto que os conceitos discutidos são comuns a qualquer sistema de Gerência de Dados Distribuídos. A própria implementação dessas idéias foi conduzida de maneira independente do modelo de dados a ser utilizado, ou mesmo se algum modelo de dados será utilizado.

---

[13] Intervalo de tempo entre o momento em que uma instrução é passada ao sistema processador até a sua execução ou retorno de um resultado.

[14] Variância da latência do canal de comunicação.

3.2.1 – A Abordagem Adotada

A princípio, reduzimos o problema à distribuição da gerência de *recursos compartilhados*, ou seja, de um recurso — tal como um arquivo, um serviço ou uma impressora — localizado em um participante e disponibilizado para uso por outros participantes da federação, em operações de leitura e escrita, exclusivas ou não. Um recurso pode estar replicado, visando prover um nível de garantia de disponibilidade, em face da baixa confiabilidade dos participantes. Neste caso, qualquer operação realizada sobre uma cópia deve se refletir sobre as demais, ou seja, a consistência mútua das réplicas passa a ser um requisito.

A cada recurso também está associado um metadado, que provê informação para operação do algoritmo de controle de concorrência e dos demais mecanismos necessários ao sistema de gerência. Tanto o recurso quanto o metadado são modificáveis, não necessariamente ao mesmo tempo. Assim, o recurso e o seu metadado possuem seus próprios números de versão. Por exemplo, ao criar mais uma réplica de um objeto, assinalamos este fato no metadado de todas as demais réplicas, incrementando-lhe a versão, neste caso, porém, não haverá mudança de versão do recurso por conta deste evento.

Uma vez definido este problema de controle de concorrência de recursos compartilhados, de modo a garantir o acesso exclusivo dos mesmos, o projeto do sistema de Gerência de Dados Distribuídos parte para suas etapas seguintes. De fato, a principal idéia desta abordagem adotada é considerar a própria base de dados como um recurso compartilhado [Rodrig 2003]; como veremos adiante.

3.2.2 – Fatores de Influência no Sistema

Características intrínsecas de redes Peer-to-Peer, tais como largura de banda restrita e atrasos de propagação de mensagens; implicam em diretrizes específicas do projeto a partir do momento em que nosso o objetivo estabelecido é o alinhamento com os requisitos de sistemas gerenciadores de banco de dados convencionais. Note que nem todos os fatores são igualmente importantes para a Gerência de Dados Distribuídos, mas dependendo do(s) objetivo(s) principal(is) da aplicação deste sistema, alguns fatores tornam-se mais importantes que outros. A seguir, comentamos em detalhes os fatores de influência levantados, baseado em [Gribble 2001], [Tanenbaum 2002] e [Haas 2002]:

Grau de Disponibilidade dos Participantes da Federação

O dinamismo do conjunto de participantes em uma rede Peer-to-Peer é uma das principais características a serem consideradas. As pesquisas em banco de dados distribuídos e federações sempre consideraram a alta disponibilidade dos computadores participantes, e os mecanismos de recuperação de falhas eram meios de garantir a consistência dos dados perante uma eventual ausência de um dos computadores participantes.

Porém, dependendo da aplicação construída sobre a rede Peer-to-Peer, o grau de disponibilidade exigido de seus participantes pode ser maior ou menor. É difícil estabelecer mecanismos que minimizem os efeitos da instabilidade da rede sem comprometer o desempenho da aplicação. Logo, admite-se certo grau de disponibilidade dos participantes para garantir certo desempenho. Por exemplo, um sistema de compartilhamento de arquivos admite maior indisponibilidade dos computadores participantes, individualmente, que um sistema de federação de banco de dados — mecanismos para contornar este problema podem ter reflexos desastrosos no desempenho do sistema se o mesmo for mais instável que o aceitável pelo projeto original.

Grau de Disponibilidade dos Recursos Compartilhados

O número de réplicas de um recurso, em diferentes peers, influencia no grau de disponibilidade do mesmo. Porém, um alto número de réplicas aumenta o risco de ocorrência de versões diferentes de um mesmo objeto, aumentando também o tráfego de mensagens de controle de versão e, conseqüentemente, piora o desempenho do sistema.

Ainda por cima, a disponibilidade de um objeto não é apenas a presença ou não de uma de suas cópias na rede, mas também desta ser a versão mais atual do objeto. Estas duas questões estão intimamente ligadas e mecanismos para o seu tratamento são tema atual de pesquisa [Kubiatowicz 2003].

Granularidade dos Objetos de Informação

A granularidade a considerar para os objetos gerenciados pelo sistema é um requisito do mesmo. Por exemplo, no problema de controle de concorrência de um sistema de gerência de dados relacional, devemos aplicar o bloqueio (*lock*) a um registro, uma tabela ou a toda a base de dados? O controle de versão deve incidir sobre uma ontologia inteira ou sobre seus termos individualmente? O que nestes casos deve ser considerado um objeto de informação, o todo ou suas partes?

Escopo da Tomada de Decisão

As tarefas a serem executadas podem ter um escopo restrito a um peer, entre dois peers ou entre todos os peers de um grupo. Uma mesma tarefa pode ter escopos diferentes dependendo do estado atual do sistema, como por exemplo, a busca pelo metadado de um recurso, que pode ser feito em escopo local ou global — caso não seja encontrado localmente.

A escolha do escopo de execução de uma tarefa, bem como o modo pela qual esta execução será conduzida, influencia no desempenho da mesma e é um requisito de projeto importante.

Extensão do Compartilhamento de Informação

Quanta informação a respeito do estado do sistema, quer seja a respeito dos recursos compartilhados ou dos peers, um participante detém no momento de tomar uma decisão dentro da tarefa que está executando? A solução de determinados problemas através de algoritmos distribuídos pode ser inviável em muitos casos devido à informação incompleta ou imprecisa [Barbosa 1995]; e isto tem implicações na escolha dos algoritmos em função das características do domínio de aplicação, bem como no desempenho de execução dos mesmos.

Heterogeneidade dos Participantes do Sistema

Seguramente, teremos peers com diferentes arquiteturas de computador, sistemas operacionais, capacidades de processamento e transferência de dados. Outro aspecto de heterogeneidade advém do modelo de dados utilizado, o que implica na utilização de conversores entre os modelos.

3.2.3 – Principais Funcionalidades do Sistema

Desconsiderando os fatores intrínsecos ao ambiente no qual o sistema será executado, as funcionalidades da federação Peer-to-Peer não diferem das federações centralizadas. No entanto, a implementação destas não pode desconsiderar os fatores enunciados, o que terá impacto na maneira como serão desenvolvidas.

Controle de Concorrência

As modificações em recurso editável, assim como no caso centralizado, requerem um bloqueio (*lock*) para prevenir modificações concorrentes. No ambiente P2P, este *lock* deve ser propagado a todos os peers ativos.

Gerência de Metadados

Os recursos compartilhados são descritos por um metadado que, por sua vez, segue as regras de formação estabelecidas em um esquema (um *XML Schema*, por exemplo). Tais metadados são importantes porque neles consta a "memória" do algoritmo de controle de concorrência que apresentaremos, bem como de outros mecanismos do sistema proposto.

Note que, sendo modificáveis, os próprios metadados poderiam ser também recursos compartilhados! Porém, eles só não o são porque estão diretamente associados aos recursos que descrevem. Portanto, um metadado só poderá ser modificado quando o próprio recurso compartilhado o puder — ou seja, quando ele estiver bloqueado pelo controle de concorrência.

Manutenção da Consistência da Base de Dados

Alguns peers podem estar ou ficar indisponíveis durante a execução de transações. Mecanismos devem ser criados para evitar que a integridade da base de dados seja afetada na ocorrência deste problema, ou seja, tais peers outrora indisponíveis devem ser notificados de todas as operações de escrita realizadas durante o período em que estiveram *off-line*. Portanto, é preciso manter um histórico das operações realizadas, sobre qualquer recurso compartilhado, ordenadas por seu instante de execução, visando à atualização das réplicas em peers inativos.

Garantia de Disponibilidade dos Recursos Compartilhados

Não há garantia absoluta de completude da consulta em um ambiente Peer-to-Peer [Gribble 2001] — a certeza de que todos os dados similares aos critérios de busca estabelecidos foram encontrados. Quanto maior o dinamismo da rede, ou seja, a taxa de entrada e saída de peers, menor a probabilidade da base de dados de estar completa em um dado instante.

Replicação é a principal estratégia de garantia de disponibilidade de um recurso compartilhado e, portanto, o principal meio de alcançar a garantia de completude da

consulta realizada sobre a base de dados distribuída [Kubiatowicz 2003]. No entanto, uma heurística de replicação, ou seja, o mecanismo responsável por decidir quando e para onde um recurso compartilhado deve ser replicado, não foi considerado neste trabalho.

3.2.4 – Gerência de Transação Distribuída

A motivação para Gerência de Transação já existia nos bancos de dados centralizados. Como os bancos são multiprocessados, o primeiro motivo é a necessidade de impedir a escrita simultânea sobre um mesmo registro. Em outros casos, uma consulta não pode ser expressa como uma única sentença da álgebra utilizada e, portanto, é preciso estabelecer uma *transação* para garantir que essa seqüência seja executada *atomicamente*. Portanto, o sistema precisa garantir a recuperação do estado anterior da base de dados perante a interrupção da transação ou ocorrência falhas do sistema durante a execução da transação — já que estas são apenas logicamente atômicas e, na prática, consistem de inúmeras sentenças e estas de várias operações sobre a base.

Além de garantir a consistência da base de dados — respeitar as restrições de integridade — é preciso garantir também a consistência das transações, ou seja, que a execução de transações concorrentes seja realizada de acordo com a especificação, e não implique em um estado da base de dados diferente do esperado, devido à influência das demais transações em execução.

3.2.5 – Confiabilidade

Quando falamos em confiabilidade em um ambiente distribuído, implicitamente, consideramos questões como garantia de manutenção da consistência da base de dados, garantia de disponibilidade do estado mais recente desta base de dados e garantia de que

uma consulta será realizada sobre esta versão da base de dados, bem como sobre todos os dados que compõem essa versão (completude da consulta).

No ambiente de redes P2P, a princípio, não há como fornecer garantias absolutas a todos estes critérios. Diversos mecanismos devem ser desenvolvidos visando maximizar estas garantias, sendo a mais importante para o nosso projeto, a consistência da base de dados. A partir dela, as demais garantias são alcançáveis através de uma adequada política de garantia de disponibilidade. Porém, sem um grau adequado de estabilidade dos peers da rede, não há como afirmar que nosso sistema se comportará com um banco de dados — segundo seus requisitos indispensáveis — durante a maior parte do tempo em que estiver em operação.

Portanto, para alcançar tais garantias de confiabilidade, mesmo em um banco de dados distribuído tradicional, é necessário implementar *protocolos de confiabilidade*, em geral, mecanismos que realizam um controle de versão da base de dados replicada e que garantem a consistência mútua das cópias, mesmo após falhas isoladas de servidores [Oszu 1999].

Em qualquer sistema de banco de dados distribuído, a parte principal da gerência de transação é o controle de concorrência, mas não é a única. Portanto, como parte do sistema de gerência de transação, é preciso prover uma espécie de histórico de transações executadas sobre os recursos compartilhados para que as operações de escrita executadas em um recurso sejam refletidas nas suas réplicas inativas no momento de seu despacho. Junto com o módulo de garantia de disponibilidade dos recursos compartilhados, estes são os principais mecanismos do protocolo de confiabilidade do sistema.

3.3 – Histórico de Transações

As transações constituem uma seqüência ordenada e atômica de operações, transmitidas através de mensagens. Uma vez confirmadas (*commit*), elas devem ser

executadas, nesta ordem, em todas as réplicas do objeto, ativas ou não. Portanto, é preciso armazená-las para atualização dos objetos inativos.

Repare que estas mensagens, uma vez enviadas e executadas, jamais sofreram alguma modificação. Portanto, após o commit, o peer cliente pode ordenar que cada peer que possui uma réplica do recurso compartilhado em uso publique, localmente, as operações de sua transação como advertisements, de modo que permaneçam disponíveis para as réplicas inativas mesmo quando o cliente da transação não estiver ativo.

Note que sendo um recurso editável, como um documento ou uma base de dados, esta muda de versão a cada transação confirmada. Assim, ao notar a diferença de versão de sua réplica deste recurso, um peer, outrora inativo, procura advertisements para operações neste objeto que resultaram nas versões posteriores, e executa-as na ordem explicitada. Como advertisements, o histórico de transações é, portanto, mantido pelo *Discovery Service*, já implementado na plataforma JXTA.

3.4 – Teoria de Algoritmos Distribuídos

3.4.1 – Tolerância a Falhas

Um sistema vem a falhar quando seu comportamento, em um dado momento, não está de acordo com a sua especificação. Um sistema distribuído possui pontos-de-falha advindos de sua própria natureza e que se somam aos pontos-de-falha dos sistemas centralizados que o constitui.

Falhas podem ser classificadas como transientes, intermitentes, ou permanentes. *Falhas transientes* ocorrem uma vez e desaparecem, mesmo após várias repetições da operação onde ocorrera. *Falhas intermitentes* ocorrem periodicamente. *Falhas permanentes* existem enquanto o componente defeituoso permanecer no sistema.

O problema das falhas de componentes em engenharia de software é tão antigo quanto à própria atividade de programar. Em [Dahl 1972], um texto clássico de programação estruturada, encontra-se uma argumentação para a vulnerabilidade dos sistemas: se um programa tem N partes, onde cada uma tem uma probabilidade *c* de funcionar corretamente, então a probabilidade de que todo o sistema funcione é de c^N, onde $0 < c < 1$. Note que quando N é grande, ou seja, um sistema é composto de muitas partes, temos a probabilidade de que ele funcione sem falhas se aproximando de zero.

Em sistemas distribuídos, estamos interessados em tornar o sistema resistente à falhas de um ou mais de seus componentes e não apenas em minimizar a probabilidade de ocorrência das mesmas, visto ser esta última abordagem muito mais difícil em um domínio instável por natureza. Existem falhas que são intrínsecas a este tipo de sistema, como por exemplo, a saída inesperada de um peer da rede. Para estas falhas intrínsecas, o sistema deve ser capaz de manter a normalidade de sua operação frente a certo número de ocorrências das mesmas.

Ainda quanto ao efeito observável, as falhas podem ser classificadas como *falhas silenciosas* e *falhas bizantinas*. O primeiro grupo abrange as falhas notáveis, onde o componente simplesmente pára de responder. O segundo grupo é constituído das falhas mais perigosas, aquelas em que o componente continua respondendo, mas em desacordo com a sua especificação e, portanto, emitindo resultados errados.

Uma terceira classificação das falhas é de acordo ao modo pelos quais os componentes deste sistema se comunicam. Em sistemas *síncronos*, componentes sempre respondem em tempo finito à chamada de outro componente. Em sistemas *assíncronos*, não existe um tempo limite estabelecido para a chegada da resposta, devido à latência da comunicação via mensagens e da carga no componente destino. Portanto, não é possível, nestes sistemas, afirmar com absoluta certeza a ocorrência de uma falha silenciosa.

Tolerância à falhas silenciosas, em sistemas distribuídos, é obtida através de *redundância* e de *sobressalência*. Por exemplo, se uma computação muito importante e

demorada precisa ser feita em um prazo determinado, convém realizá-la em mais de um sistema computacional. Por outro lado, sistemas de consulta a dados precisam apenas de máquinas sobressalentes que entram em atividade mediante a falha de uma máquina.

Falhas bizantinas requerem mecanismos especiais para a detecção de sua ocorrência. Tais algoritmos costumam envolver todos os nós participantes em um processo de chegada ao consenso, sendo denominados *Bizantine Distributed Agreements* [Lynch 1992]. Ainda assim, o consenso não é garantido em todos os cenários de aplicação, e a validade dos algoritmos é função de alguns valores para os parâmetros de entrada, como confiabilidade da entrega de mensagens, confiabilidade dos nós e modo de comunicação (síncrono ou não).

3.4.2 – Algoritmos de Chegada ao Consenso

Vários problemas práticos admitem resolução por algoritmos distribuídos de chegada ao consenso; tais como eleição de líder, divisão de tarefas, sincronização e controle de concorrência. O princípio de solução desses problemas é o mesmo da identificação de uma falha bizantina em um sistema distribuído, podendo ser exemplificado pelo *Problema dos Dois Exércitos* (Figura 2). Nele, temos um exército azul no meio de um vale e um exército vermelho dividido nas duas colinas laterais. As duas tropas do exército vermelho só vencerão se atacarem o exército azul conjuntamente. Para decidir o dia e hora do ataque, os exércitos se comunicam através de mensagens assíncronas levadas por um mensageiro que atravessa o vale dos azuis e, portanto, corre o risco de ser preso. O dilema aqui é: como saber que o mensageiro enviado chegou ao seu destino ou foi aprisionado sem que, para isso, seja preciso enviar um outro mensageiro a partir do destino?

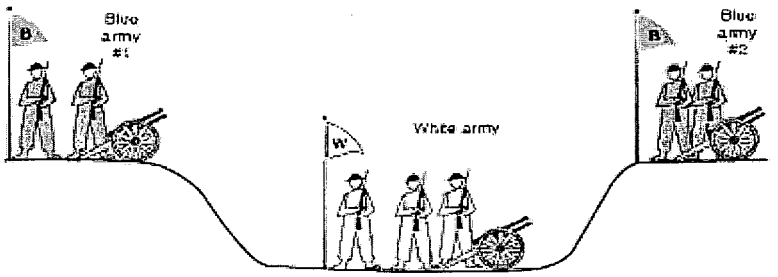


Figura 2 – O Problema dos Dois Exércitos.

Já foi provado que é *impossível* chegar a um acordo entre apenas duas partes através da troca de mensagens; logo, o problema anterior não tem solução. Porém, em [Lamport 1982] foi apresentado um algoritmo para solução de problemas bizantinos gerais, onde temos *n* nós perfeitos, sendo *m* nós com falha. Seguindo a temática dos exércitos, considere n=4 e m=1. Em uma primeira etapa, os quatro nós vermelhos enviam uma mensagem informando quantos homens dispõem individualmente para todos os demais. Só que um dos nós, o falho, é na verdade um espião do exército azul, que envia um valor diferente para cada um dos outros três. Na etapa final, cada nó envia aos demais um vetor com os valores recebidos. Facilmente, os três nós vermelhos concluirão quais são os valores corretos e qual é o errado, identificando o nó com falha (espião azul).

O algoritmo apresentado, porém, só é válido para certa restrição dos parâmetros *n* e *m*, a saber: $\mathbf{n = 3m + 1}$. Portanto, o algoritmo funciona corretamente quando menos de 1/3 dos nós apresenta falha. Há ainda uma restrição quanto à topologia da rede: se faz necessário que cada nó esteja conectado a pelo menos $\mathbf{2m + 1}$ outros. Em [Fischer 1985] ainda foi provado que não é possível garantir o funcionamento deste algoritmo em sistemas assíncronos.

3.4.3 – Sincronização

A tarefa de sincronização pode ser definida como o estabelecimento de uma ordem temporal aos elementos de um conjunto de eventos produzidos por processos concorrentes [Barbosa 1995]. Entre aplicações comuns de sincronização estão:

administrar a atividade conjunta de processos que realizam uma computação distribuída e serializar o acesso concorrente (operações) a recursos compartilhados.

	Centralizado	**Distribuído**
Baseadas em Exclusão Mútua	Semáforos / Barreiras	Circulação de Token
Sem Exclusão Mútua	Relógio Físico Contagem de Eventos	Relógio Físico Relógio Lógico

Tabela 2 – Técnicas de Sincronização e seus Domínios de Aplicação.

Sincronização pode ser centralizada ou distribuída. A centralizada prevê um relógio central coordenando as atividades e um exemplo de sua aplicação é encontrado em sistemas de memória compartilhada. Na sincronização de sistemas distribuídos, não existe um relógio central para auxiliá-la. A ausência de um relógio central implica na impossibilidade da ordenação total dos eventos do sistema.

Porém, em [Lamport 1978] é apresentado um algoritmo de ordenação parcial de eventos (*Lamport's Timestamping Algorithm*) que permite calcular, para cada evento, quais outros são pré-requisitos deste. Assim, temos uma espécie de *relógio lógico*, não associado ao tempo real, mas que garante a ordem entre os eventos pela designação de um valor numérico (*timestamp*) a cada um deles.

Graficamente, podemos representar processos concorrentes como eixos paralelos, seus eventos como pontos neste eixo — pertencentes ao conjunto de eventos V — e a relação *acontece-imediatamente-antes* (→) como uma aresta direcionada (Figura 3).

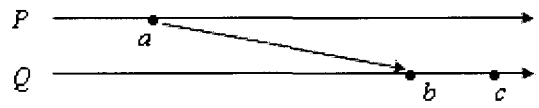


Figura 3 – Esquema gráfico de eventos seqüenciais

Neste modelo, podemos assumir que dois eventos são concorrentes se e somente se $\mathbf{a} \not\rightarrow \mathbf{b}$ e $\mathbf{b} \not\rightarrow \mathbf{a}$. Caso contrário, um evento *casualmente afeta* o outro. Ainda podemos concluir que esta relação é *irreflexiva* ($\mathbf{a} \not\rightarrow \mathbf{a}$). Adicionalmente, podemos definir a relação *acontece-antes* ($\rightarrow^+$) como um subconjunto A^+ em VxV que contém o sub-conjunto A da relação *acontece-antes*. Assim, concluímos que além de irreflexiva, essa nova relação é também *transitiva* ($\mathbf{a} \rightarrow^+ \mathbf{b}$, $\mathbf{b} \rightarrow^+ \mathbf{c}$ então $\mathbf{a} \rightarrow {}^+\mathbf{c}$). Note ainda que A^+ é o *fecho transitivo*[15] de A.

Considere o formato de mensagens m(a, T_m, i).

Regra 1: Cada processo P_i incrementa C_i entre dois eventos sucessivos.

Regra 2: A mensagem *m* enviada por um processo P_i recebe um timestamp $T_m = C_i(a)$, onde **a** é o evento ocorrendo neste processo.

Regra 3: Ao receber uma mensagem *m*, o processo P_j ajusta $C_j = \max(C_j, T_m) + 1$.

Estas regras garantem a ordenação total dos eventos pois $\mathbf{a} \rightarrow \mathbf{b}$ se e somente se $C_i(a) < C_j(b)$ ou $C_i(a) = C_j(b)$ quando $i < j$, onde $\mathbf{a} \in P_i$ e $\mathbf{b} \in P_j$.

Lamport's Timestamping Algorithm.

3.5 – Controle de Concorrência

Controle de concorrência é um conceito muito usado na teoria sistemas operacionais. Nestes, os programas — quando em execução, denominados *processos* — são executados como que simultaneamente, mesmo havendo apenas um processador. *Threads* e *Active Objects* são outras denominações para códigos de execução concorrente. A necessidade de controle da concorrência dos processos é encontrada em

[15] Seja R uma relação e P um conjunto de propriedades. O fecho de R em relação à P é a menor relação que contém R e satisfaz as propriedades em P. Por exemplo, uma relação *filho_de* e o conjunto de propriedades {transitividade} teriam como fecho uma nova relação que, considerando a semântica, poderíamos chamar *descendente_de*. Este tipo de fecho é chamado *fecho transitivo*, pois a única propriedade considerada é a transitividade. Outro tipo comum é o *fecho transitivo-reflexivo*.

diversas atividades como comunicação interprocessos, compartilhamento e competição por recursos, e sincronização de processos. Erros conceituais em programação concorrente são comuns — *data coherence*, *deadlock* e *starvation* são os principais [Tanenbaum 2001].

Em banco de dados, as transações são decompostas em operações primitivas da álgebra utilizada sobre a base de dados. Como tais transações podem ocorrer simultaneamente utilizando registros em comum desta base, é preciso garantir que a ordem de chegada dessas transações se reflita sobre as operações individualmente. Para isso, identificam-se as dependências entre as operações em um processo, o que se denomina *scheduling* (agendamento). O objetivo da decomposição de transações em operações primitivas e da serialização destas operações é otimizar a concorrência da execução, atingindo o máximo possível em desempenho, e evitando a reserva de um objeto de dados (um recurso) por tempo além do necessário.

Dependências existem e são determinadas quando duas ou mais operações estão em conflito, ou seja, quando pelo menos uma dessas operações que acessam um mesmo registro é de escrita. É preciso então determinar qual a ordem de execução dessas operações de modo que o efeito final sobre o estado da base de dados seja o esperado — considerando a execução das transações serial e não concorrente.

Consistência Seqüencial é a característica necessária a sistemas multiprogramados ou multiprocessados para que o resultado da execução de transações concorrentes seja o mesmo caso estas fossem executadas em alguma ordem seqüencial. Portanto, é necessário garantir a consistência seqüencial das operações primitivas e verdadeiramente atômicas em que são decompostas as transações concorrentes — o que também é chamado consistência da transação. Esta é a principal finalidade da gerência de transação, e o papel dos algoritmos de controle de concorrência.

Há duas grandes famílias de técnicas de controle de concorrência: as baseadas em *Locking* — registro explícito de que um recurso está reservado para um processo — e as baseadas em *Timestamp Ordering* — um *timestamp* é um identificador único

designado a cada transação permitindo ordená-las, sendo que este valor é gerado segundo uma função monotonicamente crescente. Tais técnicas podem ser ainda classificadas de acordo com alguma característica que se julgue importante notável nos principais casos de uso. De acordo com [Oszu 1999], para o nosso problema, as principais são:

- Arquitetura do Sistema de Coordenação — responsável pela criação do *lock* ou a designação do *timestamp*.
    - *Centralizado* — existe o papel de coordenador, assumido por um ou mais (alternativamente) integrantes da federação.
    - *Descentralizado* — característica recomendada para sistemas P2P, onde não há o papel de coordenador ou este é assumido momentaneamente pelos integrantes para a execução de uma operação.

- Estratégia de Execução das Operações — de modificação (escrita) da base.
    - *Otimista* — aplica as modificações na base de dados no menor tempo possível; contando que no caso da detecção de inconsistência na base de dados em um momento futuro, o *log* das transações realizadas auxilie na recuperação do último estado consistente da base. Exemplos: Bayou [Terry 1995], Coda [Kistler 1992], Deno [Cetintemel 2001].
    - *Pessimista* — procura impedir as conseqüencias do problema de Particionamento da Rede[16] (*Network Partitioning*), não permitindo que as modificações solicitadas sejam efetivadas sem a certeza de que estarão sendo em todos os servidores em funcionamento. Há uma troca de disponibilidade (das modificações) por confiabilidade (garantia da consistência da base de dados). Exemplos: Oasis [Rodrig 2003] e OceanStore [Rhea 2003].

[16] Uma rede de computadores é dita *particionada* (partitioned) se há dois ou mais conjuntos disjuntos de computadores tal que as máquinas de conjuntos diferentes não podem se comunicar uma com a outra. Cada conjunto disjunto é dito uma *partição* (partition), e o particionamento é causado por queda ou falha de canais de comunicação.

Na área de Banco de Dados, normalmente, os algoritmos de controle de concorrência adotam a estratégia pessimista, visto que ela garante a consistência da base de dados em qualquer instante. Algoritmos otimistas são mais comuns em projetos de computação móvel que aceitam operação desconexa, implicando em poucas inconsistências a resolver entre as réplicas de um recurso. Portanto, como estamos interessados no problema de Federação de Banco de Dados, nos limitamos à análise de técnicas comuns para a implantação daquela estratégia de controle de concorrência.

	Centralizado	**Descentralizado**
Pessimista	Federações de Banco de Dados	Oasis
Otimista	Bayou, Coda	Deno

Tabela 3 - Sistemas Distribuídos e suas Técnicas de Controle de Concorrência.

Como vimos, alguns requisitos foram levantados para o sistema de Gerência de Dados Distribuídos proposto. O modo de operação assíncrono e completamente descentralizado procura tornar o sistema mais flexível e tolerante a falhas, ou mesmo a ataques ao sistema. A replicação foi adotada como estratégia para garantia de disponibilidade da base de dados e, conseqüentemente, da completude da consulta; devendo, portanto, ser suportada. O perfil pessimista para o controle de concorrência sobre os recursos compartilhados do sistema foi uma escolha conservadora, normalmente empregada em federações de bancos de dados, visando à maximização das garantias de consistência da base de dados, já que os recursos são modificáveis. Assim, o algoritmo de controle de concorrência a ser escolhido deve atender a todos estes requisitos do sistema.

Na literatura sobre o assunto, uma família de algoritmos de controle de concorrência pessimista é comumente utilizada em federações de bancos de dados e aplicações com requisitos semelhantes: os esquemas de votação. Na próxima seção, apresentaremos os membros desta família projetados para gerência de dados

descentralizada, e explicaremos quais características destes algoritmos são indicadas para o domínio de redes Peer-to-Peer.

3.6 – Esquemas de Votação

Os algoritmos baseados no esquema de votação são um meio de prover controle de concorrência descentralizado, sendo indicados para uso onde são admitidos recursos compartilhados de grande granularidade, como bases de dados e serviços. Estes algoritmos suportam ainda a replicação de um recurso compartilhado, fornecendo o controle de concorrência sobre todas as cópias ativas e garantindo a sua consistência mútua. Proposto em [Thomas 1979], o consenso por maioria é o mais primitivo esquema de votação.

Quando se deseja acessar um recurso compartilhado com exclusividade, é preciso obter o *votum*, ou seja, a soma de votos de um conjunto de nós que estão ativos na rede naquele momento. O votum é bem sucedido quando esta soma de votos é superior a um limite mínimo, chamado *quorum*.

O quorum é atingido quando, a partir das respostas positivas dos nós, formamos um *coterie* [Garcia-Molina 1984] — o conjunto dos conjuntos mínimos de nós necessário para realizar um bloqueio bem sucedido nas réplicas de um recurso, ou seja, são as configurações possíveis para o quorum.

Seja um coterie C, constituído de todas as configurações de quorum possíveis c_i. Então, podemos afirmar que:

- c_i não é um conjunto vazio se $N > 0$ (*Empty Set Condition*).
- Para $i \neq j$, $c_i \cap c_j \neq \varnothing$ (*Intersection Condition*).
- Para $i \neq j$, não há c_i e c_j tal que $c_i \subset c_j$ (*Minimality Condition*).

Por exemplo, para N=4, temos C = { {n1, n2, n3}, {n2, n3, n4}, {n1, n3, n4}, {n1, n2, n4}} como o conjunto de todas as configurações possíveis para o quorum.

Esta primeira fase dos algoritmos, de obtenção dos votos, é chamada *prepare-to-commit*. O objetivo é formar algum conjunto de nós contido no coterie. Isso pode não ser possível caso o recurso já se encontre bloqueado, ou caso ocorra o timeout do processo de votação.

3.6.1 – Esquema de Votação com Pesos

Como um algoritmo clássico para prover controle de concorrência, a votação com pesos (*Weighted Voting*) foi proposta em [Gifford 1979] . Através da definição de objeto de dado (*data object*) como o recurso compartilhado a ser controlado, este algoritmo admite sua replicação e versionamento através de um grupo de repositórios de objetos que chamamos genericamente de servidores. Chamamos de clientes os participantes do grupo aptos a realizar operações de escrita ou leitura em qualquer dos objetos armazenados no grupo de servidores.

O algoritmo consta de um esquema baseado em *quorum*, onde a maioria dos servidores que contém uma réplica do objeto anuncia estar apto a atualizá-lo. Cada réplica R_i possui um número de votos V_i. As réplicas também possuem um número de versão que permite ao cliente determinar qual a cópia mais recente do objeto. Assim, o número total de votos atribuídos a todas as réplicas de um determinado objeto é:

$$N = \sum_i V_i$$

Uma operação de leitura requer um *quorum* de R votos para ler um objeto, enquanto uma operação de escrita requer W votos. O *quorum* é estabelecido através da aquisição de um *lock* em cada réplica do objeto até atingir pelo menos R ou W votos — dependendo da operação a ser realizada sobre o mesmo. A votação com pesos garante a consistência seqüencial se e somente se os parâmetros do algoritmo atendem a seguinte restrição: $\mathbf{R + W > N}$. Assim, nenhuma operação de leitura pode completar sem ter lido ao menos uma das réplicas atualizadas na última operação de escrita, já que $R > N - W$.

A distinção do quorum para operações de escrita e leitura visa ganho de eficiência. Após o bloqueio de uma operação de leitura bem sucedido, e a garantia de que todas as réplicas estão atualizadas, apenas uma das cópias precisa continuar bloqueada para leitura. Note que ela estará fora do próximo quorum e, portanto, ficará desatualizada se este for de escrita. Reparamos assim que, além desta cópia dever ser uma das mais atualizadas, deveria ser também, entre estas, uma das que tivesse o menor peso. Assim, dado este algoritmo se caracterizar por permitir múltiplos acessos de leitura concorrentes com um único acesso de escrita, estaremos maximizando o número de acessos de leitura concorrentes possíveis.

A vantagem de se atribuir pesos diferentes a réplicas do mesmo objeto reside em decidir que característica é mais importante ao sistema em questão. Alta performance pode ser alcançada atribuindo pesos maiores à servidores com maiores performances. Alta disponibilidade pode ser alcançada atribuindo pesos maiores a servidores com maior disponibilidade. As variáveis R e W também podem ser ajustadas visando características do sistema. Por exemplo, fazendo R=1 e W=N temos a máxima performance para operações de leitura; basta acessar um único servidor aleatoriamente para obter a versão mais recente do objeto, visto que a operação de escrita só ser efetuada quando todos os servidores estiverem prontos. Para garantir a máxima tolerância à falhas, basta fazer R e W próximos à N/2 o que permite até N/2 - 1 dos servidores falharem sem comprometer a disponibilidade da base de dados. Note ainda que W < R não é utilizado, pois não garantiria a utilização da cópia mais atual nas operações de leitura.

3.6.2 – Esquema de Votação com Pesos Descentralizada

O algoritmo de Gifford foi proposto para aplicação em bancos de dados distribuídos e, por isso, estabelece a existência de um *coordenador* para centralizar as requisições aos servidores. Ele também é responsável por coletar o *quorum* e armazenar informações sobre os objetos (metadado). O metadado é especificado para cada objeto

— e, portanto, devendo ser o igual em cada réplica — consiste de: N, R, W, versão do metadado, versão do objeto, número de votos (peso), e a lista de participantes que possuem réplicas.

No caso de redes P2P, os peers podem assumir o papel de clientes ou de servidores alternadamente. Porém, dada a baixa disponibilidade dos peers, não é recomendado que nenhum deles assuma o papel de coordenador. Por essa razão, em [Rodrig 2003] foi proposta uma revisão do algoritmo de Gifford que acaba com o papel de coordenador, distribuindo a sua responsabilidade entre todos os peers. A descentralização é alcançada pela replicação dos metadados em cada peer que contém uma réplica do objeto de dado, em vez de mantê-los no coordenador. Assim, como todos os peers têm o mesmo status, temos um sistema completamente descentralizado. As restrições são $\mathbf{R + W > N}$ *e* $\mathbf{W \geq R}$ — esta última visando garantir a consistência seqüencial para o caso em que não há coordenação central. Fazendo $W \geq R$, já que nenhuma operação de leitura é executada enquanto pelo menos R votos forem obtidos, garantimos que pelo menos um dos servidores reservados possui a versão mais recente do objeto — a que já sofrera a última operação de escrita realizada.

O fluxo de execução do esquema de votação descentralizada consiste, essencialmente, de três passos:

1. **Requisição de metadado**

Para o objeto de interesse, o peer cliente precisará descobrir qual dos demais peers possui a versão mais recente do metadado.

2. **Estabelecimento do *quorum* com o *lock* de réplicas**

Após a obtenção do metadado, o peer cliente está apto a enviar a solicitação de lock do objeto para os demais peers da rede. Cada peer que receber a mensagem responderá se pode ou não conceder o *lock* — um outro peer pode ter feito uma requisição de lock sobre o mesmo objeto. Se o número mínimo de mensagens positivas (R ou W) for alcançado, o *lock* estará estabelecido.

3. **Confirmação do Lock**

Uma vez obtido o quorum mínimo de peers que possuem um *lock* sobre o objeto em questão, a versão mais recente do objeto estará determinada e as cópias desatualizadas receberão as operações de modificação necessárias para a atualização. Depois, é seguro ordenar a execução de qualquer operação sobre o objeto bloqueado, visto que a maioria dos peers on-line consentiu este bloqueio.

O restante dos peers, mesmo que ainda não estejam aptos a executar a operação, serão notificados do consenso. Note que este conjunto de peers — que é minoritário — havia recebido um pedido de *lock* de outro peer qualquer, motivo pelo qual negaram o pedido de bloqueio. Como é garantido que este outro processo de bloqueio concorrente não alcançará o consenso, ele não enviará operações sobre o objeto.

Um histórico das operações, ordenadas cronologicamente, deve ser armazenado e replicado em todos os peers visando o auxilio a atualização dos peers que estiverem inativos ou defasados. Este histórico não requer ser explicitamente sincronizado visto que isto ocorrerá como conseqüência da própria atualização das bases de dados destes peers que, por sua vez, devem apenas se preocupar em obter este histórico do peer que possui a versão mais atual do objeto. E esta operação será executada sempre que um bloqueio sobre o recurso for obtido.

Operações de leitura implicam no *lock* de apenas uma réplica, a mais atualizada, segundo a apuração realizada na etapa 2. Apenas a operação de escrita requer o *lock* de todas as cópias do objeto. Note, porém, que o quorum de R peers é necessário para garantir a localização da réplica mais atual. Caso esse quorum não seja possível, qualquer cópia estará sempre disponível para consulta, porém não há garantias de que seja a versão mais recente do objeto.

3.6.3 – Problemas Previstos

O esquema de votação apresentado em [Rodrig 2003] implica em alguns riscos de falhas quando executados em um ambiente instável como aquele de redes Peer-to-Peer. Os peers individualmente podem ficar sem comunicação, mesmo que temporariamente, pois não há garantias de qualidade de serviço (QoS) da rede. Este fato impõe quatro cenários de falha do sistema de Gerência de Dados Distribuídos:

- **Problema A**: Queda do peer cliente após quorum parcial ou completo: manteria as cópias bloqueadas indefinidamente, mas é facilmente solucionável com um aborto do bloqueio (ou da transação) após a ocorrência de um evento de timeout.
- **Problema B**: Queda súbita de peers servidores antes da votação — caso o número de peers ativos caia para menos que o quorum (R ou W), o processo de votação não poderá resultar em bloqueio do recurso, tornando indisponível o sistema. Em outras palavras, o sistema não se adapta ao novo cenário de réplicas ativas resultante da saída de alguns peers.
- **Problema C**: Particionamento da Rede — assim como no caso de queda dos peers servidores, a solução proposta em [Rodrig 2003] pode levar à indisponibilidade do sistema quando o número de réplicas dos peers ativos é inferior ao quorum (R ou W).
- **Problema D**: Risco à completude da consulta — uma heurística de replicação da base de dados é uma solução para garantia de disponibilidade da mesma, e esta deve identificar as articulações do grafo da rede, onde pode haver particionamento da mesma, e garantir que a possível partição majoritária tenha cópias de todos os recursos compartilhados em uso. Este problema, no entanto, não é abordado neste trabalho.

A queda súbita do peer cliente (*Problema A*) é solucionável pelo mecanismo de timeout. A desvantagem desta abordagem é que ela deteriora o desempenho do sistema, já que o evento de timeout só ocorre após um longo período de tempo, prolongando um processo de votação fadado ao insucesso e prejudicando pedidos posteriores sobre o mesmo recurso oriundos de outros peers clientes. Portanto, a saída de um peer da rede sem aviso prévio poderia ser detectada, ou melhor, a presença de cada peer poderia ser monitorada constantemente.

Este ação de monitoramento preventivo também é importante para os *problemas B e C*. Eles levam o sistema a um estado de indisponibilidade desnecessariamente. O quorum da votação poderia ser adaptado ao novo conjunto de réplicas ativas no caso de queda de um ou mais peers servidores, uma vez que estes estajam sendo constantemente monitorados.

Porém, o cenário descrito no *Problema C* não é solucionável apenas com a monitoração de presença dos peers. O particionamento da rede é uma falha mais séria e difícil de detectar. Ele ocorre quando algumas rotas são interrompidas, deixando a rede dividida em duas ou mais partes isoladas, completamente sem comunicação com as demais. Mesmo que se mantenha um monitoramento de peers, conforme proposto, não é óbvia a distinção entre o particionamento da rede e a queda coletiva destes peers. Afinal, em ambos os casos, os peers ficam inacessíveis.

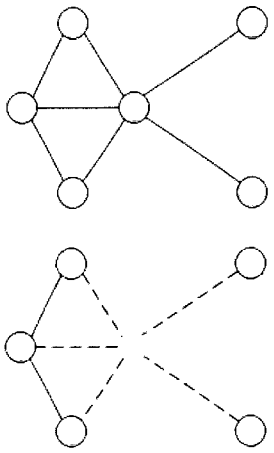


Figura 4 – Particionamento da rede.

A queda súbita de muitos peers em um curto espaço de tempo é bastante rara. O tempo médio de votação é proporcional à média do *round trip time* (RTT) de mensagens entre dois peers. Assim, nestes casos, podemos considerar a ocorrência do fenômeno do particionamento da rede. Porém, se pudéssemos saber qual a topologia da rede no instante anterior ao fenômeno do particionamento, poderíamos efetivamente inferir a sua ocorrência nos casos em que existe um único nó capaz de desconectar o grafo da rede (Figura 4).

Capítulo 4 – A Implementação do COPPEER

4.1 – Cenário Atual dos Sistemas Distribuídos

Ainda que o conceito de Peer-to-Peer e suas aplicações já estejam populares e largamente em uso, o cenário atual é de múltiplas aplicações para propósitos específicos, sem qualquer interoperabilidade entre elas, levando o usuário a executar diferentes aplicativos quando necessitar de diferentes serviços ao mesmo tempo. Até aplicações com a mesma finalidade, como compartilhamento de arquivos, possuem diversas implementações que estão completamente segregadas de seus pares, como é o caso do Kazaa, e-Mule, Gnutella, e tantas outras.

Novas tecnologias têm sido desenvolvidas para suportar o desenvolvimento destes sistemas distribuídos, visando inclusive uma possível interoperabilidade entre aplicações desenvolvidas por equipes distintas, através de um modelo bem definido, flexível, e comum a suas aplicações clientes. Entre as plataformas para desenvolvimento de aplicações P2P mais populares podemos citar a JXTA — usado neste estudo — e sua correspondente na plataforma .NET.

No entanto, mesmo dispondo de uma plataforma como a JXTA, os desenvolvedores ainda precisam projetar suas aplicações considerando funcionalidades muito elementares e não-funcionais, tais como validação do usuário, operações básicas sobre a rede Peer-to-Peer (criar grupo, localizar grupo, ingresso em grupo, localizar peer, localizar recursos, etc), aspectos de interface com o usuário, entre outros. Ainda que a plataforma JXTA, bem como suas pares, ou mesmo APIs de terceiros, de propósitos distintos, forneçam praticamente todos os recursos necessários para implementação destas funcionalidades; podemos considerar que este é um esforço desnecessário, gasto em requisitos não-funcionais de qualquer aplicação considerada. Portanto, tais funcionalidades deveriam estar disponíveis, sobre a plataforma de rede Peer-to-Peer considerada, de maneira reutilizável e adaptável.

Adicionalmente, para sistemas Peer-to-Peer, não existe um modelo de desenvolvimento elaborado a exemplo do que já existe para sistemas Cliente/Servidor. Atendo-se ao domínio do Java, o J2EE é com certeza a referência principal. Seu modelo é tão elaborado que ultrapassa os domínios da computação distribuída, abrangendo também *MVC Frameworks*, componentes de interface com o usuário, mecanismos de template, manipulação de documentos XML, mecanismos de persistência de objetos, entre outros, direta ou indiretamente associados à especificação J2EE.

Assim, uma das propostas deste trabalho foi o estabelecimento de diretrizes para o desenvolvimento de um *Peer-to-Peer Application Framework* sobre a plataforma JXTA, bem como fornecer uma implementação de referência. Sua definição compreende uma descrição conceitual e um conjunto de mecanismos de suporte que orientam como aplicações semelhantes podem ser arquitetadas e implementadas. Usualmente, um framework deste tipo inclui: uma proposta de arquitetura básica, padrões de projeto (*design patterns*), protocolos, APIs, entre outros recursos para desenvolvimento. Entre os benefícios advindos da adoção deste framework, podemos citar:

- Definição de uma linguagem comum para projetos de um mesmo domínio (de aplicação ou de tecnologia);
- Atendimento de requisitos não-funcionais, permitindo que o desenvolvedor se dedique mais ao modelo de negócio;
- Viabilização da interoperabilidade entre aplicações desenvolvidas por fornecedores distintos;
- Agilidade no desenvolvimento.

Entre diferenças notáveis de um Application Framework para aplicações Cliente/Servidor e um similar para aplicações Peer-to-Peer [Roussopoulos 2004], podemos destacar:

- Ausência de controle central dos computadores participantes da computação distribuída, parcial ou total, bem como dos processos e recursos localizados nos mesmos;
- Imprevisibilidade dos computadores participantes, tanto em número de participantes quanto em disponibilidade na rede;
- Alta latência e não-confiabilidade da comunicação entre os computadores participantes;
- Heterogeneidade de plataformas e poder computacional dos participantes;

A plataforma COPPEER, iniciada no contexto deste trabalho e atualmente em desenvolvimento pelo grupo de pesquisas em sistemas Peer-to-Peer da COPPE/UFRJ, visa atender a todos os requisitos de um típico *Application Framework* orientado à tecnologia Peer-to-Peer. Baseado na infra-estrutura de redes Peer-to-Peer da plataforma JXTA, o COPPEER foi originalmente concebido como uma extensão do projeto MyJXTA2[17]. Este é o projeto mais popular já desenvolvido utilizando JXTA, pois tem como objetivo servir de demonstração dos principais conceitos da plataforma para desenvolvedores iniciantes. Trata-se apenas de uma aplicação para bate-papo, porém, dada sua popularidade, funcionalidades e interface gráficas já disponíveis, foi adotada como ponto-de-partida para a implementação deste projeto.

O COPPEER não procura apenas abstrair a complexidade da API do JXTA; o projeto vai além quando adapta soluções de desenvolvimento de software reconhecidamente eficientes em outros domínios de aplicação. O princípio que guiou o COPPEER é o de entender o JXTA, provendo requisitos não-funcionais comuns às aplicações, e não ser uma camada sobre o JXTA, ocultando-o. Por exemplo, o COPPEER provê recursos de desenvolvimento rápido de interfaces com o usuário, desenvolvimento baseado em componentes, gerência de transações na rede, entre outros.

[17] http://myjxta.jxta.org/

Existem alguns trabalhos na área de frameworks para aplicações P2P. Considerando o domínio da plataforma Java, o COPPEER possui algumas semelhanças com o projeto XNap [Berger 2003], tal como o conceito de plug-ins e o esquema de interface gráfica da plataforma — ainda que este aspecto fora influenciado pelo projeto MyJXTA2, sobre o qual o COPPEER foi construído. Iniciado antes mesmo do JXTA, o XNap possui sua própria infra-estrutura de redes P2P. Sobre a plataforma JXTA, existe apenas um projeto, o LANCS P2P Framework, descrito em [Walkerdine 2004]. Ainda que desenvolvido de maneira independente e paralela ao COPPEER, também possui alguns conceitos em comum com este. O COPPEER, porém, aborda todos os aspectos do desenvolvimento de aplicações, enquanto o LANCS focou na independência da infra-estrutura de redes P2P — tais como Pastry e Gnutella, apesar de ter sido efetivamente implementado apenas sobre o JXTA — e ignorou muitos recursos avançados do JXTA ao adotar um modelo P2P que fosse universal e, portanto, simplificado.

4.1.1 – Requisitos Básicos da Plataforma

Como uma plataforma para desenvolvimento de aplicações P2P, o COPPEER precisa fornecer alguns serviços básicos para os programadores. Note que o termo "serviço" aqui não se refere necessariamente a um JXTA Service, mas a funcionalidade do *framework* útil para o desenvolvimento de aplicações P2P. Há um consenso deste esquema de serviços com a plataforma proposta em [Walkerdine 2004], ainda que ambos os trabalhos tenham sido desenvolvidos paralelamente.

- **Comunicação baseada em Mensagens:** Conforme visto, são duas as principais abordagens de comunicação entre peers na plataforma JXTA: *Pipes* e o *Resolver Service*. Como a utilização de pipes exige a gerência de pipes e seus end points, optamos por implementar um sistema de mensagens para o COPPEER usando o Resolver Service, conforme será mostrado adiante.

- **Troca de Dados Binários:** Para aquisição e instalação de aplicações (Plug-ins) em tempo real, replicação de objetos compartilhados visando garantia de disponibilidade ou simples compartilhamento de arquivos. Note que esta funcionalidade requer ainda a implementação de protocolos de confiabilidade da comunicação.

- **Indexação e Busca de Recursos:** Aplicação ao nível de usuário para indexação e busca de recursos para desenvolvimento de aplicações ou simples objetos compartilhados para consumo dos usuários.

- **Awareness e Lista de Interessados:** Como mostramos no capítulo anterior, é importante estar sempre ciente do estado da rede em termos de peers ativos; bem como da versão mais recente dos objetos compartilhados. O mecanismo de controle de presença de peers (awareness) e de lista de interessados nos objetos compartilhados.

- **Autenticação de Usuários:** Tanto para ingresso no ambiente de execução da plataforma quanto para o ingresso nos grupos disponibilizados.

- **Gerenciamento de Transação**: Visa garantir a confiabilidade da comunicação e consistência do estado do sistema quando uma mesma operação tiver de ser executada identicamente em diversos peers.

- **Ambiente de Gerência e Monitoramento da Plataforma:** Visando facilitar a instalação de aplicações através da composição de serviços em grupos, gerenciamento de plug-ins instalados, entre outros requisitos.

Este trabalho projetou e implementou apenas alguns destes serviços, os mais essenciais, a saber: comunicação baseada em mensagens, autenticação de usuário, parte do sistema de gerência de transação e de gerência da plataforma.

4.2 – Um Framework para Aplicações em Redes P2P

O projeto de um *Application Framework* para aplicações P2P deve levar em consideração as notáveis diferenças em relação ao atual modelo de computação dominante, o cliente-servidor. Um único peer deve ser capaz de prover serviços aos demais e incorporar novas funcionalidades. Assim, um usuário requer executar apenas uma aplicação (o *framework*) para ter acesso a qualquer aplicação ou serviço, disponível na rede Peer-to-Peer.

A principal abstração é o **recurso**. Peers oferecem, buscam e contratam recursos oferecidos por outros peers. Estes recursos variam de serviços complexos até simples arquivos para *download*.

Um serviço é uma definição abstrata para a funcionalidade provida por um módulo do sistema. Os clientes de um serviço podem enviar-lhe requisições localmente (por chamadas a métodos) ou remotamente (por um *middleware* de computação distribuída). Porém, no COPPEER, bem como no JXTA, os serviços podem estar instalados em um peer ou grupo de peers. A implementação de um serviço do COPPEER é baseada em classes que estendem o modelo provido pelo JXTA — mais especificamente, o Resolver Service —, visando à simplificação de seu processo de implementação. Assim como serviços do JXTA recebem o nome de *JXTA Services*, também nos referimos à extensão desse modelo, provida por este trabalho, como *COPPEER Services*. As classes de um serviço do COPPEER devem ser empacotadas em um arquivo JAR, e este disponível nos peers de um grupo, quando todos os peers inscritos em um grupo devem prover os serviços associados.

Considere o seguinte exemplo didático da necessidade de serviços: se você deseja calcular a raiz quadrada de um número e não tem uma rotina para executá-la, deve localizar na rede um Peer Group associado a um serviço de cálculo da raiz quadrada que, portanto, é oferecido por todos os peers inscritos neste grupo. Após escolher um peer desse grupo, você pode remeter seu valor numérico para o mesmo e aguardar a resposta do peer, emitida quando este tiver terminado o processamento.

Um serviço é implementado como uma simples classe Java, que estende uma classe do framework, e está inscrito na plataforma para atender requisições. Um serviço é, portanto, uma unidade autônoma pronta para atender requisições remotas. Para acessar um serviço, remotamente, seu cliente deve usar um *stub*, ou seja, uma classe que também implementa a interface que define os métodos de negócio do serviço, localizada junto do cliente. Este componente de acesso local do serviço da rede pode ser antes localizado em algum serviço de oferta de componentes a desenvolvedores.

Para requisições locais de serviços, existe o conceito de **componente**, ou seja, uma peça de código reutilizável, seguindo o padrão convencionado para POJOs — *Plain Old Java Objects*, em outras palavras, Java Beans que incorporam métodos de negócio. Este é, na verdade, uma interface que especifica os serviços providos pelo componente, ou seja, os métodos de negócio. Pelo menos uma classe que implementa esta interface deve ser provida, que pode referenciar instâncias de outras classes, de modo que, juntas, constituem a implementação do componente. *Stubs* para um serviço remoto também podem ser tratados como um componente.

Componentes enviam e recebem requisições de outros componentes. Os **plug-ins** são componentes não re-utilizáveis, que enviam requisições a outros componentes, mas recebem requisições apenas do usuário, através da interface gráfica associada ao mesmo. Plug-ins também recebem eventos da plataforma COPPEER, para que possa tomar as medidas previstas. As requisições do usuário do plug-in também são eventos comunicados ao mesmo.

Componentes e plug-ins também são recursos, embora inexistentes no JXTA. Para os recursos originários da plataforma JXTA, é obrigatório a sua oferta na rede através da publicação de um advertisiment específico. A criação e publicação de um grupo, bem como a publicação de serviços nele, são irreversíveis, ao menos até a expiração do advertisement associado a estes recursos, o que não deve ser seriamente levado em consideração durante o desenvolvimento da aplicação usuária deste grupo. Portanto, é necessário projetar a aplicação antes de publicá-la; uma nova versão de um

único serviço corresponderá a um novo *Module Specification Advertisement* e um novo Peer Group.

Estas características da plataforma JXTA aumentam a complexidade de desenvolvimento em relação a outras tecnologias de computação distribuída. Optamos por não ocultar o JXTA sob uma nova camada de abstrações, mas estendê-lo. Assim, o COPPEER fornece meios de automatizar muito da "burocracia" inerente à programação para JXTA sem, com isso, tornar-se incompatível pela adoção de um modelo totalmente novo. Por exemplo, ainda que os três advertisements para um serviço ainda existam, sua geração pode é automatizada pela plataforma, bastando cadastrar a implementação do serviço COPPEER.

Estas unidades conceituais do COPPEER funcionam de maneira autônoma, como se existissem por si só. Uma presta serviço às outras, e da interação entre elas surge o sistema idealizado pelo projetista. Assim, o princípio de baixo acoplamento é levado ao extremo, o re-uso destas unidades torna-se implícito ao processo de desenvolvimento, e a implementação é simplificada, podendo fazer uso intensivo de casos de teste automatizados, como de fato, a própria implementação da plataforma o fez.

4.3 – Arquitetura de Aplicação

O desenvolvimento de aplicações para o COPPEER é baseado em componentes, ou seja, uma aplicação é composta de módulos menores e reutilizáveis que interagem uns com os outros visando implementar a funcionalidade desejada em nível de usuário.

Na primeira versão do COPPEER, procuramos definir um modelo de desenvolvimento baseado em componentes bastante simples, que por um lado facilita a implementação de aplicações, por outro, exige maior disciplina e bons hábitos de programação por parte do desenvolvedor. Definimos algumas entidades de programação conceituais no framework que possuem características distintas entre si e, portanto,

devem ser aplicados visando diferentes finalidades na composição de sua aplicação. Note, porém, que várias destas entidades não são explicitamente declaradas como tal no código de implementação das classes e, portanto, é de responsabilidade do usuário não corromper o modelo proposto através de utilizações indevidas dos conceitos que estas representam.

Esta flexibilidade que o COPPEER proporciona segue a filosofia dos padrões de desenvolvimento de software (*design patterns*): propõe ser uma orientação, não uma norma a ser estritamente seguida. Modificações ao modelo proposto são possíveis e até estimuladas sempre que forem mais adequadas para tratar o problema em questão.

As entidades especificadas no modelo de desenvolvimento de software do COPPEER são:

Componentes

Provêm serviços bem definidos cuja implementação pode depender dos serviços providos por outros componentes. O COPPEER provê um gerenciador de componentes que realiza o carregamento da classe, instanciação, acoplamento com outros componentes, registro de componentes por referência e controle de acesso. Todos os componentes de todas as aplicações coexistem no mesmo gerenciador de componentes, a menos que a aplicação solicite explicitamente o seu próprio gerenciador.

Nesta plataforma, existem dois tipos de componentes [Bass 2003]:

- ***Singletons***: São componentes que admitem a existência de uma única instância registrada em um mesmo gerenciador de componentes. Em geral, fornecem serviços que não exigem a manutenção do estado do componente — equivalem aos *Stateless Session Beans* do J2EE

- ***Prototypes***: São componentes que admitem existir mais de uma instância registrada em um mesmo gerenciador de componentes. Em geral, fornecem

serviços que exigem a manutenção do estado do componente — equivalem aos *Stateful Session Beans* do J2EE.

Note que componentes não devem armazenar dados da aplicação. Eles apenas devem implementar o modelo de negócio da aplicação. Ainda seguindo esta lógica, os componentes não devem ser persistentes.

Objetos de Dados

Os objetos de dados (*data objects*) se distingüem dos componentes não apenas pelo seu papel na aplicação que constituem, como também por não terem seu ciclo de vida controlado pelo gerenciador de componentes. Em geral, objetos de dados são instanciados e referenciados por um componente sendo, portanto, dependente deste.

Tais objetos mantêm informação relacionada a uma instância de componente ou mesmo à aplicação como um todo, bem como provê métodos que operam sobre estes dados.

O COPPEER não realiza nenhuma tarefa de persistência de objetos de dados. Há muitas alternativas de persistência além da persistência "manual" usando JDBC, tais como serialização, transações ao estilo de banco de dados orientados a objetos (JDO), etc.

Note que, mais uma vez, a responsabilidade de seguir bons hábitos de programação é deixada ao cargo do desenvolvedor. Por exemplo, é aconselhável a adoção do padrão *Flyweight* [Grand 1999] que preza pela separação entre dados e código em diferentes objetos visando a alta reutilização daqueles que contém a funcionalidade e que, por isso, tendem a assumir o papel de *Singleton Component.* Assim, reduzimos a utilização de recursos do sistema e evitamos o sobrecarregar o gerenciador de componentes.

4.3.1 – Microkernel

Componentes são concebidos de acordo com o padrão *Inversion Of Control* (IoC) [Martin 1996]. Tal padrão sugere que o acoplamento entre componentes não deve ser explicitamente codificado, ou seja, nenhum componente deve utilizar alguma *Factory* — através de um método estático da mesma — para obter instâncias de outro componente. A solução proposta pelo padrão é a declaração da dependência entre os componentes, através da enumeração das classes destes componentes como argumentos do construtor ou propriedades — ao estilo JavaBeans. O acoplamento propriamente dito é realizado pelo COPPEER, mais especificamente, pelo seu *microkernel.*

Microkernel é um conceito desenvolvido na teoria de sistemas operacionais (SO) [Tanenbaum 2001]. Um SO baseado em microkernel possui diversos serviços básicos (device drivers, I/O, gerência de memória, etc) implementados como *módulos*, restando ao kernel apenas o trabalho de garantir o carregamento e comunicação entre estes módulos. Um SO onde tais serviços pertencem ao kernel é chamado *Monolítico.*

Para implementar o microkernel do COPPEER, nós utilizamos o *Spring Framework* [Johnson 2004], uma implementação do padrão IoC para Java. As dependências entre componentes são resolvidas pelo Spring, devidamente descritas em um arquivo de configuração XML (components.xml). Os componentes dos quais depende um componente específico são passados via construtor ou via o par de métodos *set/get*, visto que componentes são, a princípio, JavaBeans.

Se um ou mais componentes necessários não forem encontrados, eles podem ser localizados usando os serviços de busca da plataforma COPPEER, usando como parâmetro o *full qualified name* (nome totalmente qualificado) de sua classe principal.

A classe *net.jxta.coppeer.plugin.ComponentManager* (Figura 5) encapsula e gerencia o microkernel, utilizando os recursos da API do Spring, e o termo *ComponentManager* é por vezes usado no lugar de microkernel na documentação da plataforma.

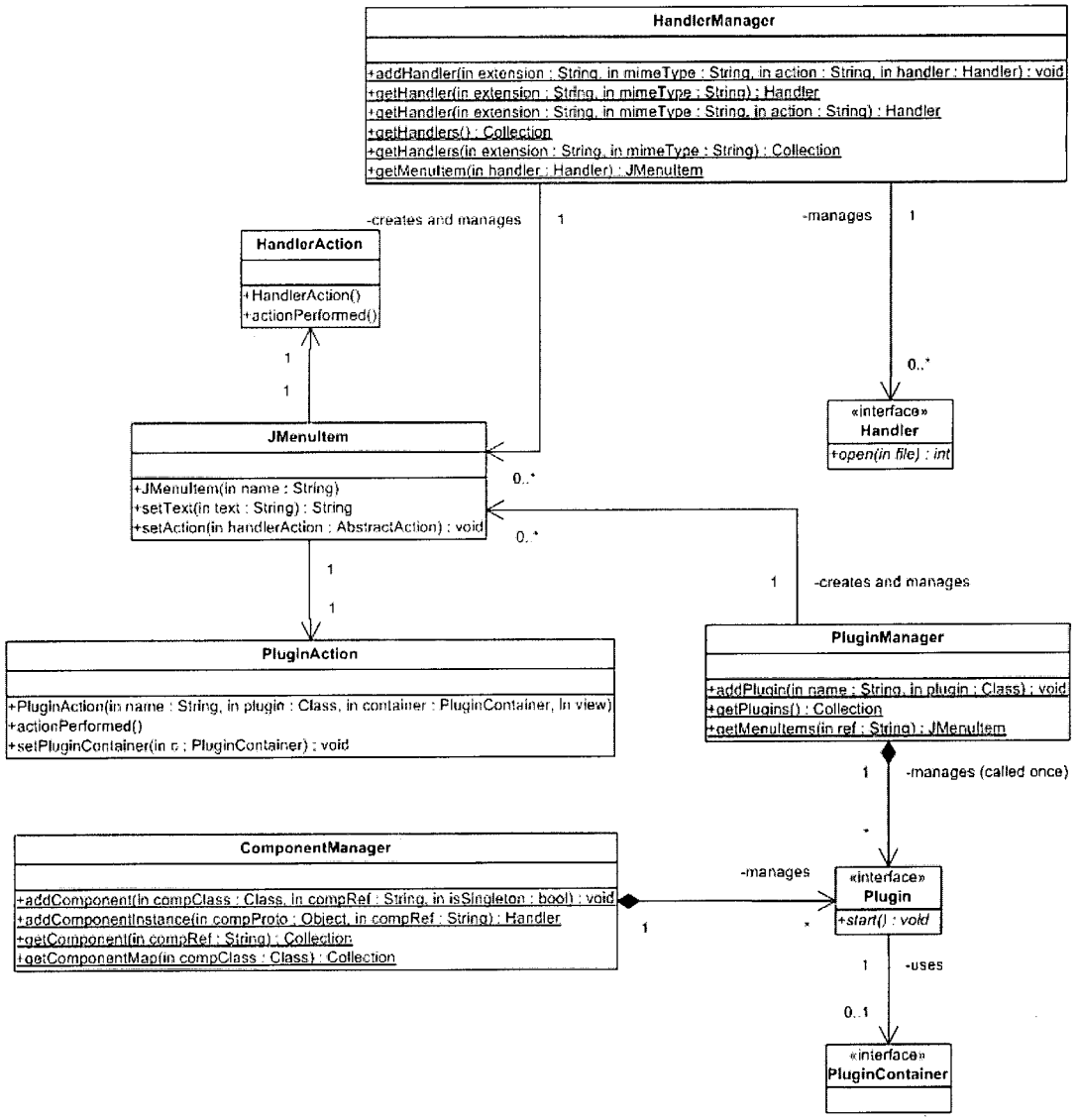


Figura 5 – Modelo de Classes do Microkernel do COPPEER.

4.3.2 – Componentes e Plug-ins

No COPPEER, um componente não precisa estender nenhuma classe, pelo menos a princípio, visto que existem *papéis* que um componente pode assumir no framework. Para estas situações, existem interfaces que a classe do componente deve implementar para que o container identifique o papel do mesmo.

Este é o caso dos *plug-ins* do COPPEER. Um plug-in é um tipo de componente que não-reutilizável, visto que serve apenas para implementar código de inicialização de aplicações instaláveis no container. Ao instalar um componente que implementa a interface *net.jxta.coppeer.plugin.Plugin*, o ComponentManager identifica o papel e toma providências específicas para plug-ins. Por exemplo, um item de menu para invocação do Plug-in é automaticamente instalado segundo dados disponíveis no metadado que descreve o plug-in.

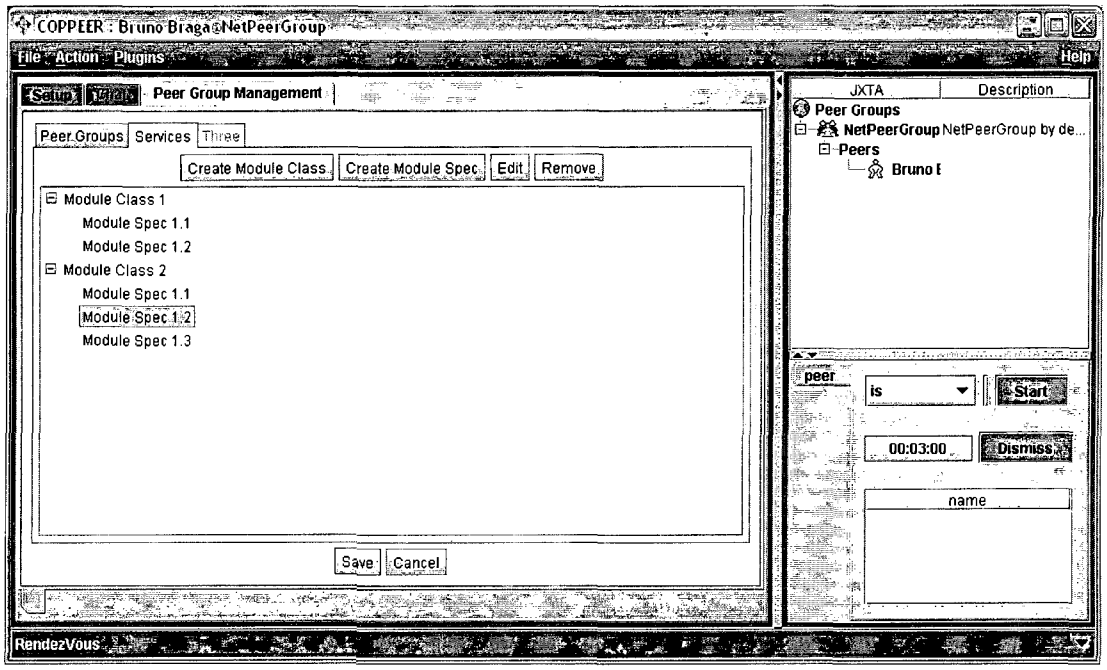


Figura 6 – Interface Gráfica do COPPEER e as Áreas de Trabalho para Plug-ins.

O plug-in é, portanto, apenas o elo entre a aplicação desenvolvida e a plataforma COPPEER. A implementação da plataforma pode e deve ser realizada em outras classes, como se não fosse ser executada no COPPEER, exceto quando invocasse algum componente padrão ou serviço remoto. A própria arquitetura interna da aplicação fica à cargo do desenvolvedor.

Um plug-in geralmente possui uma interface gráfica com o usuário (GUI). Esta interface gráfica é implementada sobre um JPanel utilizando *widgets* Swing ou por intermédio do *User Interface Framework* do COPPEER (Figura 6), sendo instalada em uma das duas áreas de trabalho providas — implementações da interface

PluginContainer. A interface de um plug-in também pode ser desenvolvida completamente alheia ao fato de ser ou não exibida nas áreas de trabalho do COPPEER.

4.4 – Arquitetura de Computação Distribuída

4.4.1 – Computação Orientada a Mensagens

Middlewares baseados em mensagens não constituem uma idéia necessariamente nova. Nestas arquiteturas, *mensagem* é a unidade básica de informação enviada de um processo para outro, em geral, localizados em máquinas diferentes. Os dados contidos na mensagem variam de simples textos a dados estruturados complexos. As principais vantagens de aplicações baseadas em mensagens são:

- Orientação a eventos;
- Possibilidade de comunicação *one-to-many*;
- Fácil implementação de interoperabilidade entre sistemas heterogêneos;
- Baixo acoplamento entre componentes (*loosely-coupled*);
- Comunicação confiável e transacional.

Um dos principais produtos para computação orientada a mensagens é o *Java Message Service* ou JMS [Hapner 1999]; e não se trata exatamente de um middleware para esta finalidade, mas uma especificação comum para interoperabilidade entre diferentes implementações de serviços de mensagens. Além das características enunciadas à cima, o JMS provê persistência de mensagens, visando à garantia de entrega mesmo quando o destinatário estiver ausente.

Modos de Comunicação

- *Síncrono* — O processo que requisita o serviço é bloqueado até o término do processamento e chegada da mensagem de resposta (*blocking call*). Os

processos envolvidos precisam estar ativos e confirmam o recebimento das mensagens (*acknowledgement*). Por essa razão, é muito susceptível à falhas. Exemplo: chamadas à métodos de objetos.

- *Assíncrono* — A invocação do serviço não espera o término do processamento ou chegada da mensagem, ocorrendo de maneira independente e paralela (*non-blocking*). É mais resistente à falhas e escalável, tanto em termos de número de mensagens quanto de número de processos participantes.

Modelos de Computação baseada em Mensagens

- Ponto-a-Ponto (*Point-to-Point*) — A comunicação ocorre entre dois processos, sendo modelo mais utilizado em qualquer tecnologia de comunicação.
- Publicação/Assinatura (*Publish/Subscribe*) — A comunicação ocorre *one-to-many* ou *many-to-many*, através do estabelecimento de *tópicos*, os quais classificam as mensagens de modo que os processos interessados (inscritos) em um determinado tópico recebem qualquer mensagem publicada e associada ao tópico em questão.

	Publicação/Assinatura	**Ponto-a-Ponto**
Síncrono	-	Invocação de método em objeto
Assíncrono	*Multicasting*	*Polling*

Tabela 4 – Aplicações segundo Modelos e Modos de Comunicação baseada em Mensagens.

Serviço de Mensagens no COPPEER

Os conceitos de computação baseada em mensagens apresentados usam a nomenclatura da especificação do JMS. Um modelo semelhante foi implementado na plataforma COPPEER, estendendo a funcionalidade básica provida pelo *Resolver Service*. Conforme visto, o este serviço do núcleo da plataforma JXTA permite

comunicação baseada em mensagens, entre os peers; sendo mais flexível que a abordagem usando JXTA Pipes, que não admitiria alguns dos conceitos mostrados aqui.

4.4.2 – Gerenciando Grupos e Serviços

Para facilitar o gerenciamento de PeerGroups e seus JXTA Services, parte principal do estabelecimento de comunidades e provisão de aplicações para seus usuários, desenvolvemos um plug-in que cadastra informações sobre estas entidades e permite invocar ações sobre as mesmas.

4.4.2.1 – Descritores de Grupos

- **PGA :** Atributos de um Peer Group Advertisement.
  - **GID :** Peer Group ID.
  - **MSID :** ModuleSpecID para este Peer Group.
  - **Name**
  - **Description**
  - **Passwd :** senha para inscrição de um peer neste grupo.
  - **Svc :** lista de JXTA Services a serem instalados neste grupo, ou melhor, dos MSID de versões deste serviço (com implementações em Java).

4.4.2.2 – Descritores de Serviços

No COPPEER, a instalação de serviços JXTA em um Peer Group é realizada usando informações armazenadas em um arquivo XML com o seguinte formato:

- **MCA :** Atributos de um Module Class Advertisement.
  - **MCID :** ModuleClassID.
  - **Name**
  - **Description**

- **MSA :** Atributos de um Module Specification Advertisement.
  - **MSID :** ModuleSpecID.
  - **Name**
  - **Creator :** nome da pessoa ou instituição criadora.
  - **Description**
  - **URI :** De referência ao serviço, tal como:
    `http://www.cos.ufrj.br/~brunorb/TransactionService`
  - **Version**
- **MIA :** Atributos de um Module Implementation Adverstisement
  - **Code :** um nome de classe totalmente qualificado, tal como:
    `net.jxta.coppeer.service.transaction.TransactionServiceImp`
  - **Description**
  - **Provider**

Note que existem outras informações necessárias para a instalação de um JXTA Service em um Peer Group (PeerGroupID, PeerGroupSpecID). Porém, estes dados devem ser fornecidos no ato da implantação do PeerGroup.

4.4.2.3 – Publicando Grupos e seus Serviços

A publicação de grupos e serviços deve ser conjunta na plataforma JXTA. Uma vez publicado um grupo, este deve apenas ser re-publicado sem modificações em sua configuração — apenas para o caso de vencimento do advertisement nos peers da rede. Caso contrário, corre-se o risco de ter diferentes advertisements (configurações) para o mesmo PeerGroupID. Os serviços disponíveis para os peers de um grupo fazem parte da configuração do mesmo, constando no ModuleImplAdvertisement do PeerGroup, e não podem ser modificados.

No COPPEER, implementamos uma classe (*ServiceDeployment*) que presta serviços de criação e publicação de um grupo e seus serviços. Suas instâncias devem ter definidas as propriedades de uma única tripla *(Class, Specification, Implementation)* correspondente a um único serviço de um grupo. Tal objeto pode ser reaproveitado para

associar este serviço a outro grupo que pretenda ofertá-lo, bastando entrar com as propriedades relativas ao novo grupo e repetir o procedimento de publicação do mesmo.

As propriedades de uma instância de *ServiceDeployment* são todas relacionadas ao serviço que ele representa, e constam de:

- String ModuleClassID
- String ModuleClassName
- String ModuleClassDescription (*opcional*)
- String ModuleSpecID
- String ModuleSpecName
- String ModuleSpecCreator (*opcional*)
- String ModuleSpecDescription (*opcional*)
- String ModuleSpecURI
- String ModuleSpecVersion
- String ModuleImplCode
- String ModuleImplDescription (*opcional*)
- String ModuleImplProvider (*opcional*)

4.4.3 – Arquitetura do Sistema de Mensagens

No COPPEER, a troca de informações entre os peers é realizada utilizando-se o Resolver Service da plataforma JXTA. Porém, a comunicação usando esse serviço ainda é consideravelmente precária pois não há informações de cabeçalho e a manipulação das mensagens fica a cargo do desenvolvedor. Visando o estabelecimento de um sistema de mensagens, a exemplo do *Java Message System* [Hapner 1999], o pacote *net.jxta.coppeer.service* provê um conjunto de classes para comunicação Peer-to-Peer em mais alto nível.

ServiceMessage

- **Header :** Informações de cabeçalho da mensagem, ou seja, um metadado para controle da plataforma COPPEER.
    - **MessageClass :** classe de implementação da interface *Message*, para manipulação do documento XML contido na tag <Body>.
    - **SrcPeerID :** JXTA's ID do peer que enviou esta mensagem.
- **Body :** contém o documento XML que corresponde a uma mensagem especificada a nível de aplicação; portanto, seu conteúdo não é regulado nesta especificação, bastando ser um documento XML bem formado.

Todas as propriedades do cabeçalho da mensagem do COPPEER Service são copiadas para propriedades da implementação padrão da interface Message, logo que a mesma é obtida através do método getBody() de ServiceMessage. Apenas o queryID, um valor de controle do ResolverService, é copiado fora deste método.

4.5 – Conclusão

No momento em que este texto foi escrito, a primeira versão do COPPEER estava em condições de suportar o desenvolvimento de aplicações baseadas em comunicação Peer-to-Peer. O protótipo do controle de concorrência do COPPEER-DB, que será apresentado adiante, já havia sido implementado e testado com sucesso nesta plataforma.

Capítulo 5 – Uma Proposta de Esquema de Votação Dinâmica Descentralizado e Tolerante a Falhas para Redes P2P

Neste capítulo, discutiremos o que há na literatura para controle de concorrência em redes Peer-to-Peer, revisaremos a fundação teórica de algoritmos distribuídos que justifica as decisões de projeto dos algoritmos referidos e apresentaremos a nossa proposta de algoritmos para o problema de controle de concorrência descentralizado e tolerante a falhas para redes Peer-to-Peer.

5.1 – A Proposta de um Esquema de Votação Resistente a Falhas para Redes Peer-to-Peer

Como visto, os esquemas de votação apresentados exigem um quorum de R ou W votos em relação a um número N de nós fixo. Como $W \geq R$, com menos de W nós ativos o sistema já ficaria indisponível, pelo menos para escrita. O esquema de votação adotado, portanto, deve suportar um número variável de réplicas de um objeto de dados, se adaptando ao novo conjunto de nós ativos, para que a disponibilidade do sistema em si seja maior.

Se simplesmente utilizarmos os esquemas de votação apresentados anteriormente, alterando N, quando o particionamento da rede ocorrer, os peers de cada partição poderão continuar realizando transações independentemente, já que não há como inferir qual a partição majoritária no decorrer do funcionamento do sistema. Apenas quando as partições se reencontrarem, será possível perceber que a consistência mútua entre as réplicas do recurso terá sido violada por duas ou mais partições em operação paralelamente. Como o custo da recuperação do estado da base de dados anterior à primeira inconsistência ocorrida é alto, devemos criar mecanismos que

previnam este cenário, ainda que estes estejam sujeitos a falso-positivos que tornem o sistema indisponível.

O esquema de votação dinâmica [Davcev 1985] lida com um universo de votantes variável, pois modifica o peso dos nós com direito a voto de modo a manter a proporção entre os seus pesos mediante um novo cenário de nós ativos. No entanto, a alteração dos pesos não é trivial se estivermos usando um algoritmo de votação descentralizado, e seria precedida de um processo de chegada ao consenso. É possível estabelecer um esquema de votação dinâmica onde a decisão de alteração de pesos é tomada autonomamente pelos nós da rede, porém, em redes Peer-to-Peer, não há garantias de que todos os peers têm a mesma noção de quantos de seus pares existem na rede em um dado instante. Os novos pesos são calculados em função desta informação, o que poderia afetar a manutenção da exata proporção dos pesos dos peers em alguns casos.

Uma alternativa seria adotar a estratégia de *Update Sites Cardinality* [Jajodia 1987] para votação dinâmica. Trata-se da inclusão de um parâmetro extra (*Cardinality*) que armazena a quantidade de nós que participaram da última votação bem sucedida realizada. Apenas estes nós podem participar da votação seguinte, e apenas eles contêm a versão mais recente do objeto de dados (Figura 7). Os peers desatualizados, que reencontrarem a partição ativa, podem ser informados deste fato logo que a próxima votação ocorrer, como nos esquemas anteriores. Eles são computados como participantes da votação para efeito da determinação do valor de *Cardinality*, mas para esta primeira votação são considerados como tendo peso zero.

	n_1	n_2	n_3	n_4	n_5
Versão	18	18	18	18	18
Cardinalidade	5	5	5	5	5

	n_1	n_2	n_3	n_4	n_5
Versão	19	19	19	18	18
Cardinalidade	3	3	3	5	5

	n_1	n_2	n_3	n_4	n_5
Versão	20	19	20	18	18
Cardinalidade	2	3	2	5	5

Figura 7 – Exemplo mostrando como o esquema de Jajodia se adapta a dois particionamentos de rede intercalados a dois processos de votação em seqüência.

Porém, admitindo a ocorrência de particionamentos da rede, o desempenho do sistema pode deteriorar se a partição majoritária — e, portanto, ativa — for sucessivamente particionada, chegando ao mínimo de dois nós na partição ativa. O último particionamento teria que decidir, por algum critério, tal como o menor ou maior ID, qual peer é a partição ativa, para que o sistema não fique indisponível. Este, no entanto, é um problema inerente do algoritmo, e constitui um caso particular.

Ainda assim, nós percebemos que a aplicação do esquema de votação dinâmica em sistemas Peer-to-Peer não é imediata. O **primeiro problema** é com o parâmetro de cardinalidade, que representa o atual número de réplicas atualizadas e ativas na partição da rede considerada. Devido a atrasos na entrega de mensagens em redes Peer-to-Peer, o quorum deve ser anunciado o mais breve possível, apesar da impossibilidade de calcular o novo valor de cardinalidade antes da chegada de todos os votos dos nós ativos – que são coletados até a ocorrência do evento de timeout, visto que alguns podem ser perdidos no caminho ou chegar com um atraso anormal.

Um **segundo problema** identificado é que a estimativa da máxima duração da votação (o tempo de timeout) pode gerar um atraso desnecessário em casos onde o processo de votação está fadado ao fracasso devido a um particionamento da rede ocorrido previamente – é o caso de qualquer partição que não seja a majoritária. A ocorrência deste problema tem um impacto pequeno, mas, como veremos, sua solução é conseqüência natural da solução do primeiro problema.

Um **terceiro problema** foi identificado na presença de particionamentos de rede sucessivos, que podem deixar o sistema inteiro em um falso estado de indisponibilidade. Por exemplo, duas ocorrências sucessivas de particionamento da rede no intervalo entre duas votações subseqüentes podem ser confundidas com um único evento de particionamento que resultou em três partições distintas (Figura 8). Neste caso, se

nenhuma destas partições tiver o necessário número de votos para alcançar o quorum, o sistema entrará em estado de indisponibilidade sem uma razão real.

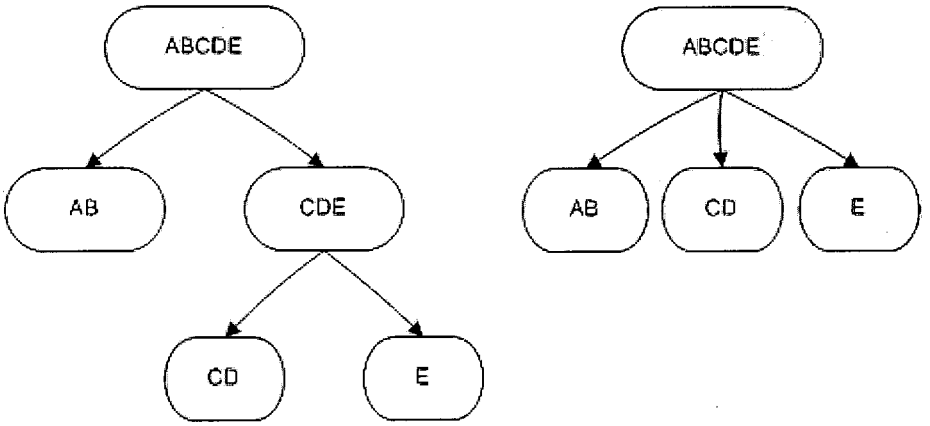


Figura 8 – Seqüências de particionamento da rede distintas podem ser confundidas.

Vale ressaltar que mesmo sendo possível que tais problemas ocorram, o esquema de votação de Jajodia e Mutchler funciona. Porém, ao os resolvermos, principalmente o primeiro, estaremos reduzindo o tempo médio de execução da votação (problema 1 e 2) ou eliminando hipóteses de falsa indisponibilidade do sistema (problema 3). Haverá um ganho de performance e disponibilidade.

Para resolver o primeiro e segundo problemas citados, nós propomos o uso de um mecanismo de controle de presença para gerar e manter uma lista de peers ativos e monitorar a integridade da partição. Além da entrada e saída de nós, a configuração topológica da rede pode ser modificada pela remoção ou adição de um link entre dois nós ativos, que podem corresponder a eventos de particionamento ou de reintegração, ou seja, a saída ou a entrada coletiva de nós. Com esta medida, a cardinalidade pode ser contabilizada antes da aquisição do quorum, sendo enviada na mensagem de requisição de votação aos demais peers. Simultaneamente, será possível saber se há um número suficiente de nós ativos antes mesmo de iniciar a votação, prevendo o cenário onde a votação está fadada ao insucesso antes mesmo de começar.

Porém, para detectar particionamento da rede em tempo real, e evitar o terceiro tipo de problema identifiicado, nós propomos estender o mecanismo de controle de presença para um mecanismo de monitoramento da topologia da rede. Note que o único

momento em que os peers chegam a um consenso é durante o estabelecimento do quorum. Seria preciso, portanto, a cada evento de modificação da rede, atualizar a topologia da rede nos nós de maneira paralela e independente das votações. A idéia é permitir seja possível identificar as articulações e, conseqüentemente, os pontos de risco de cisão da rede.

Note que, considerando o esquema de votação de Jajodia e Mutchler, reduzimos o nosso problema em criar e provar de um mecanismo de monitoramento da topologia da rede com a finalidade de prover o parâmetro de cardinalidade e detectar o particionamento da rede.

5.2 – A Proposta de um Mecanismo para o Monitoramento da Topologia da Rede

O projeto de um protocolo de monitoramento do estado de uma rede Peer-to-Peer e de seus participantes ativos deve considerar alguns requisitos: (1) informar a todos os peers sobre os eventos de entrada e saída de um peer, bem como o estabelecimento de novos links na rede; (2) minimizar o tempo para o alerta da ocorrência destes eventos a todos os participantes; (3) minimizar o número de mensagens trocadas entre os participantes para atingir o objetivo estabelecido.

Perceber a entrada de um novo peer é fácil: basta que o mesmo envie uma mensagem para todos os demais na rede. O problema maior é como perceber a saída de um peer da rede. Não podemos assumir toda saída como sendo intencional e precedida de uma comunicação, a exemplo do que ocorre com o evento de entrada. Quedas individuais e involuntárias de peers precisam ser percebidas, mesmo que na verdade haja apenas uma falha na comunicação entre peers.

Vale ressaltar que por motivo de simplicidade, considera-se que todos os nós da rede considerada contenham uma réplica do recurso em questão. Assim, não há diferença, nesta seção, de falar-se em réplica de recurso ou em nó. Além disso, na

explicação da solução, separamos em cada item as atividades que o mecanismo deve realizar para que ele, continuamente, preste o serviço desejado: manter os nós informados da topologia da rede de que participam e, com isso, contabilizar o parâmetro de cardinalidade.

5.2.1 – O Mecanismo de Monitoramento de Presença dos Nós

Considerando que o nosso domínio é o das redes Peer-to-Peer, desejamos um mecanismo de monitoramento de presença de peers totalmente descentralizado. Logo, é preciso que a responsabilidade de controle da presença de peers na rede seja dividida entre todos, através da troca periódica de mensagens entre pares de peers (*polling*) e a identificação de novos nós vizinhos por mensagens de broadcast. Cada aresta do grafo para a rede – considerando haver uma aresta entre cada par de peers que se comunicam diretamente – corresponde a um processo de polling mútuo entre os pares.

Os nós dos extremos da rede Peer-to-Peer ficam localizados em redes locais, e são conectados à rede por um ou mais nós roteadores. Sendo todos conectados entre si, podemos representar esta situação como um grafo completo. Também pode haver grafos completos no interior da rede Peer-to-Peer.

Após k retransmissões de uma mensagem de polling sem ter havido resposta, podemos assumir ter havido queda do canal de comunicação entre estes dois nós. Considerando o timeout igual à média do intervalo de tempo entre o envio de uma mensagem de polling e o recebimento de outra mensagem como resposta (ERTT) acrescida de n vezes o seu desvio-padrão (*Deviation*), temos uma probabilidade p_1 de o atraso de qualquer mensagem não ser superior ao valor de timeout. Como a variação do RTT é dada por uma distribuição normal, fazendo n=4 temos praticamente 100% das amostras (SRTT) abaixo do valor especificado para o timeout. Neste caso, resta apenas o caso de perdas de mensagem de polling. Mantendo uma estatística de mensagens perdidas isoladas, ou seja, excetuando-se aquelas perdidas no cenário de particionamento de rede, podemos estimar uma probabilidade p_2, que na prática tende a

um valor baixo. A variável aleatória que rege o número de mensagens transmitidas até que uma chegue ao seu destino é descrita por uma distribuição geométrica em p_2. Assim, podemos calcular k tal que $P(X>k) = 1 - \sum_{i=0}^{k} p(1-p)^{i-1} < p_3$, para uma probabilidade p_3 de falso-positivos, que desejamos ser bastante baixa.

5.2.1.1 – Monitoramento em Estruturas de Anel

O polling dentro dos grafos completos não precisa ser de todos para todos os nós; e concentrá-lo em um único nó não é recomendado para os grafos completos no interior da rede Peer-to-Peer.

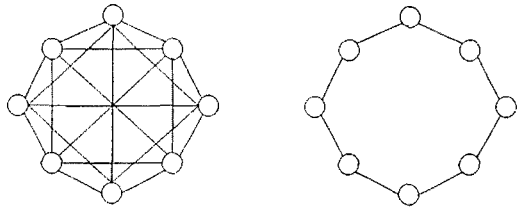


Figura 9 – Um grafo completo e o respectivo grafo anelar.

Uma estrutura para descentralização do monitoramento na forma de um *anel* (Figura 9) surge como uma solução para minimizar o número de parceiros na troca de mensagens de *polling* a dois.

5.2.1.2 – Cálculo do Timeout

Quando um peer vizinho hospeda o serviço de monitoramento de presença, intercepta a mensagem de polling, responde com outra mensagem de polling, e não permite que a primeira seja propagada. Caso o peer não hospede este serviço, ele propaga a mensagem e não é considerado vizinho.

Quando um peer entra em um anel da rede, sua posição no mesmo é calculada por ordenação crescente simples dos Peer IDs – calculado por uma função de hash MD5 e, portanto, considerado único na rede.

$$SRTT = t_{polling\_sent} - t_{polling\_received}$$
$$ERTT = (1-x).ERTT + x.SRTT$$
$$Deviation = (1-x).Deviation + x.|SRTT - ERTT|$$
$$Timeout = ERTT + 4.Deviation$$

onde x = 0.125

Figura 10 – Cálculo do Timeout em função do último RTT da mensagem de polling (SRTT) e o valor médio do RTT acumulado (ERTT).

O timeout especificado para a chegada da resposta a uma mensagem de polling (Figura 10) é calculado por uma fórmula comum em aplicações de rede [Kurose 2000].

5.2.2 – A Comunicação de Eventos de Modificação da Topologia

Sempre serão dois os nós diretamente envolvidos no evento de criação ou remoção de um link na rede. Porém, diversos poderão ser os nós afetados por este evento, passando a ser ou deixando de ser articulação. No caso de criação de um link, autonomamente, apenas um deles se torna responsável pela comunicação do evento aos interessados – o que tiver menor ID, por exemplo. Porém, no caso de remoção de links, ambos os nós devem emitir a mesma mensagem de comunicação do evento, pois pode ter havido particionamento da rede.

Todos os nós da rede devem ser notificados de todos os eventos. Porém, esta operação é lenta e freqüente, portanto, deve ser realizada paralelamente à operação de votação. É importante, no entanto, garantir que:

1. A cardinalidade estimada antes da votação seja fiel à realidade da configuração da rede no instante da votação, para solução do problema 1;

2. Seja possível distingüir eventos de particionamento da rede sucessivos, intercalados entre duas obtenções de quoruns, para evitarmos considerar uma falsa indisponibilidade, que nos referimos como problema 3;

Note, porém, que a cardinalidade é contabilizável a partir do grafo que representa a topologia da rede, e este é atualizado em função das mensagens de eventos de modificação da topologia recebidos pelos peers. Portanto, precisamos manter, em paralelo ao serviço de votação, uma espécie de *log* de eventos de modificação da topologia da rede.

A técnica de atualização da topologia da rede adotada é inspirada no algoritmo de roteamento de pacotes *Link State* [Kurose 2000]. Para prover informações aos nós-roteadores de modo que eles possam calcular as tabelas de roteamento com os melhores caminhos a cada outro nó-roteador da rede, ao ocorrer um evento de modificação da topologia da rede, o nó imediatamente envia uma mensagem (*Link State Advertisement*) aos nós vizinhos, diretamente conectados. A mensagem é então propagada por toda a rede (*flooding*) [Barbosa 1995], e considerada apenas se o seu número de seqüência é superior ao da última mensagem recebida daquele nó, ou seja, os nós da rede devem manter o número da última mensagem recebida de cada nó presente na rede. Em cada um dos nós, o evento de chegada desta mensagem executa um algoritmo que re-cria o grafo da rede e, a partir deste, usando um algoritmo de caminho mínimo, re-cria a tabela de roteamento. A principal diferença é a nossa necessidade de manter a topologia para identificar articulações e a cardinalidade, ao passo que não efetuamos nenhum cálculo de melhores rotas (a segunda parte do *Link State*). Outra diferença é a necessidade de que um nó confirme o recebimento da mensagem por ele re-propagada por todos os seus vizinhos, anexa (*piggybacked*) na mensagem de polling; caso contrário, um novo evento de queda de link deve ser disparado. Por fim, como o procedimento de distribuição da informação é o mesmo do *Link State,* a prova de convergência do *flooding* é análoga a daquele algoritmo.

O conteúdo da mensagem de polling citada deve transmitir informação suficiente para a construção do grafo da rede nos nós envolvidos e, depois, pelos demais nós. Assim, ela deve ter a seguinte estrutura:

```
MensagemDePolling
        NóOrigem : Nó;
        NóDestino : Nó;
        VizinhosDoNóDestino : Conjunto de Nó;
```

Como o envio das mensagens de modificação da topologia ocorre pela técnica de inundação (*flooding*), esta informação se propaga como uma onda através da rede, e não atomicamente. Assim, em um dado momento após o evento, um grupo formado pelos nós que tivessem recebido a mensagem do evento teria uma representação da topologia da rede diferente do grupo que ainda não recebeu a mensagem. Se esta mensagem for de queda de uma articulação, o impacto na fidedignidade da cardinalidade contabilizada em um peer que não recebeu a mensagem do evento é considerável, pois todo um segmento da rede, ou seja, mais do que um único nó pode ter se tornado inacessível. Então, a questão que precisa ser respondida é: isso pode afetar o algoritmo?

Lembremos que a cardinalidade só é efetivamente registrada no metadado do recurso após o quorum de operações de escrita ser alcançado, e ainda desejamos que o peer cliente envie, no início da votação, o valor de cardinalidade a ser registrado no quorum. Isso significa, para o esquema de votação, consenso da maioria – em relação ao valor da cardinalidade registrado no quorum anterior. Suponha então o caso citado de ter havido, instantes antes, um evento de particionamento da rede cuja mensagem enviada por inundação ainda não tenha alcançado o peer cliente iniciador da votação. Ora, se ele alcançou o quorum, é porque pertence à partição majoritária resultante do evento de particionamento da rede. Assim, registrando o valor da cardinalidade anterior, é como se postergássemos a ocorrência do evento de particionamento da rede para registro somente no próximo quorum. Um exemplo do cenário descrito pode ser visto na figura 11.

	A	B	C	D	E
Versão	51	51	51	51	51
Cardinalidade	5	5	5	5	5

Peer Cliente: A
Cardinalidade Monitorada:
C(A) = 5, C(B) = C(C) = 3.

	A	B	C	D	E
Versão	52	52	52	51	51
Cardinalidade	5	5	5	5	5

Peer Cliente: A
Cardinalidade Monitorada:
C(A) = C(B) = C(C) = 3.

	A	B	C	D	E
Versão	53	53	53	51	51
Cardinalidade	3	3	3	5	5

Figura 11 – Registro de cardinalidade desatualizada durante o quorum não implica em falha no esquema de votação.

No esquema de Jajodia e Mutchler, em qualquer votação, apenas os votos dos nós com a réplica mais atual são considerados. Os demais hajem como testemunhas da votação – conforme a técnica de *voting with witenesses* [Borghoff 2000] –, e seus votos são considerados apenas para contabilizar a cardinalidade. Podemos interpretar a proposta de adaptação deste esquema de votação como uma antecipação dos votos de testemunhas através do monitoramento da topologia. Como os nós estão, continuamente, avisando os demais da sua presença, a mais recente destas mensagens serve como voto de peso zero para qualquer votação que a suceda. Isto fica implícito para os demais nós da rede, pois o polling de um nó é descentralizado, sendo explicitamente realizado apenas pelos seus vizinhos. Este *status* implícito de voto com peso nulo das mensagens de polling só será revogado se e somente se: (1) for identificada a sua saída da rede, justamente pelos critérios definidos na seção 5.5.1; ou (2) for recebido, em tempo, um voto explícito deste nó, em resposta à votação iniciada. Estes dois eventos serão garantidamente comunicados, conforme descrito anteriormente.

Como votos de testemunhas não são considerados para o quorum, dado uma determinada cardinalidade registrada no metadado, o número mínimo de votos necessários para o quorum é determinado em função dela e deve ser alcançado para prosseguir à atualização da mesma. Isso é uma propriedade do esquema de Jajodia e Mutchler original, mantida no esquema adaptado proposto.

Concluída a formulação do nosso algoritmo, fornecemos seu pseudocódigo:

```
Seja
```

- C' a cardinalidade contabilizada pelo monitoramento da topologia da rede.
- C a cardinalidade registrada no metadado do recurso, presente em cada uma das réplicas do mesmo.
- Q(C') o número mínimo de nós para obtenção do quorum estabelecido em função da cardinalidade contabilizada (ou monitorada).
- Q(C) o número mínimo de nós para obtenção do quorum estabelecido em função da cardinalidade registrada.

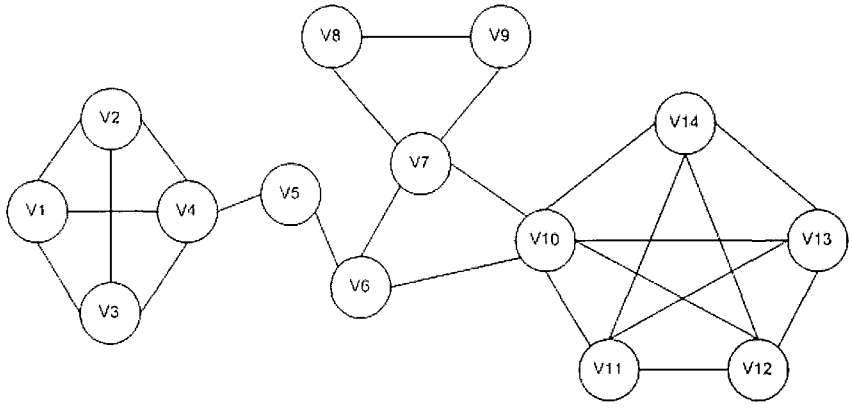
```
No nó inicidador da votação:
    Se C' = C então
        OBS: O esquema funciona como o Jajodia e Mutchler original.
        Aguardar Q(C) votos positivos para o quorum.
    Se C' < C então
        Se C' < Q(C) então
            OBS: O sistema está indisponível;
                Não estamos na partição majoritária.
        Senão
            Enviar o novo valor de C := C';
                OBS: note que Q(C) < C' < C.
            Aguardar Q(C) votos positivos para o quorum.
    Se C' > C então
        OBS: Novas réplicas podem ter ingressado no sistema;
            Elas serão testemunhas da votação.
        Faça
            Enviar o novo valor de C := C' (onde Q(C) < C' < C).
            Aguardar Q(C') votos positivos para o quorum;
                OBS: Q'(C') > Q(C), o quorum é mais rigoroso;
```

Note que na última instrução de aguardar, se C' > C não for condição verdadeira, o quorum não será alcançado. Provavelmente, um evento que reduz o valor de C' está a caminho – lembre-se que a técnica de inundação garante

Por fim, vale lembrar que um evento de particionamento da rede significa a perda de um link com um nó-articulação e, por conseqüência, das rotas para os demais nós isolados por este. Um evento de queda do link com um nó não-articulação decrementa C' em uma unidade, enquanto o mesmo evento para um nó articulação decrementará C' de várias unidades. Nos eventos de criação ou re-criação do link, os nós envolvidos trocam informações e os C' de cada um são incrementados.

5.2.3 – A Identificação dos Pontos de Risco de Particionamento

É preciso identificar os nós que desconectam a rede e criam regiões que não se comunicam, as *articulações*. A identificação de articulações, ou seja, dos nós cuja queda provoca particionamento da rede, começa através do cálculo da interseção entre os conjuntos de vizinhos dos nós diretamente conectados a ele. Um nó pode ser articulação se ele for o único vizinho em comum de todos os seus vizinhos. Porém, é possível que haja um caminho alternativo entre as supostas partições que resultariam da queda deste nó. Por exemplo, na rede dada na Figura 12, se houvesse uma ligação entre v_2 e v_8, então v_4, v_5, v_6 e v_7 não seriam articulações.



$C_5 = \{ v_4, v_5, v_6 \}$
$C_6 = \{ v_5, v_6, v_7, v_{10} \}$
$C_7 = \{ v_6, v_7, v_8, v_9, v_{10} \}$
$C_{10} = \{ v_6, v_7, v_{10}, v_{11}, v_{12}, v_{13}, v_{14} \}$

$A_6 = C_5 \cap C_7 \cap C_{10} = \{ v_6 \}$

Figura 12 – Cálculo para identificação de um candidato à articulação do grafo para a rede P2P.

Para saber se um candidato à articulação o é de fato, precisamos realizar uma busca por caminhos alternativos entre nós de partições distintas que resultariam da queda do nó analisado (Figura 13). Como cada nó tem uma noção estática do estado atual da rede, ele pode manter um vetor com todas as arestas – canais de comunicação ponto-a-ponto – da mesma. Considere cada aresta como uma relação denominada *x alcança_diretamente y* (reflexiva). Eliminando todas as arestas contendo o candidato à articulação, devemos verificar a validade da relação *x alcança y* – que é o fecho-transitivo de *alcança_diretamente*, portanto, reflexiva e transitiva – para cada par de vizinhos do nó-candidato para os quais a relação *alcança_diretamente* é falsa.

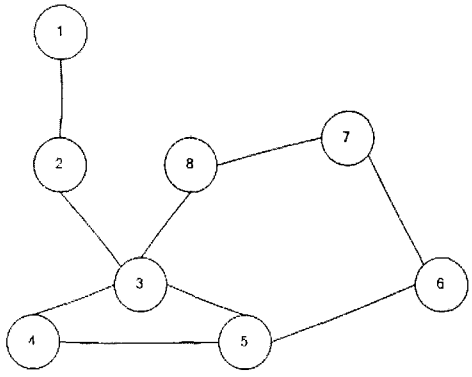


Figura 13 – O nó 3 é identificado como candidato à articulação mas não é articulação.

Um algoritmo computacionalmente eficiente para a identificação das articulações de grafos, proposto em [Tarjan 1972], realiza uma busca em profundidade por caminhos alternativos para cada nó da árvore, cuja raiz é o nó que o executa. Cada nó da rede executa este algoritmo sempre que recebe um evento de modificação no estado da rede.

Seja a função lowpt(v_i) o nó mais próximo da raiz da árvore que pode ser alcançado a partir de v_i caminhando para baixo zero ou mais arestas e terminando com zero ou uma para cima – aresta de retorno. Segundo o algoritmo de Tarjan, um nó v_i é articulação se e somente se ele possui algum filho v_j tal que lowpt(v_j) é o pai v_i ou o próprio de v_j. O algoritmo é correto porque se v_i é articulação, não pode haver aresta unindo algum descendente a ascendente de v_i.

Note que com este mecanismo de identificação das articulações, quando um nó vizinho inferir a saída de um nó-articulação por falha do polling, ele poderá inferir também a saída de todos os demais nós isolados por este, e comunicar o evento aos demais da rede. O evento comunicado é o de perda do link com o nó-articulação. A inferência de particionamento fica a cargo de cada nó, baseado na informação topológica que detenham.

5.3 – Análise do Desempenho do Esquema de Votação Proposto

As simulações a seguir correspondem ao esquema de Votação Dinâmica de Jajodia e Mutchler adaptado para Redes Peer-to-Peer, conforme a proposta desta dissertação de mestrado. A simulação foi realizada usando um modelo analítico implementado no *Maple* 8.0[18]. A duração da votação referida compreende o envio da mensagem de pedido de bloqueio até a chegada do número necessário de respostas.

O tempo de votação é variável porque devido ao monitoramento dos nós da rede, a cardinalidade pode ser estimada antes da votação. O resultado da votação é anunciado assim que se torna possível presumi-lo. Portanto, em todos os cenários de testes, a grande maioria das votações realizadas foi encerrada antes do limite estabelecido para timeout.

Considere os gráficos a seguir, sendo dois para cada uma das três simulações. Os gráficos da esquerda apresentam um histograma do experimento de 1000 votações realizadas aleatoriamente, para os quais a *coordenada X* representa a *n*-ésima votação realizada e a *coodernada Y* indica a correspondente duração da *n*-ésima votação em milisegundos[19]. Nos gráficos da direita, mostramos a consolidação das amostras de

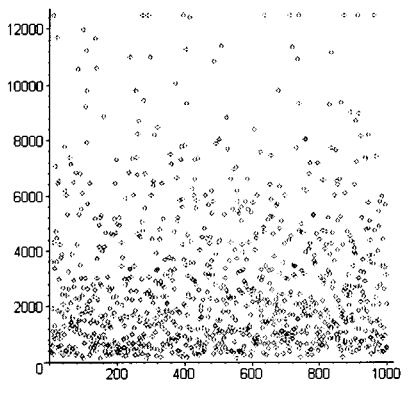
[18] http://www.maplesoft.com/

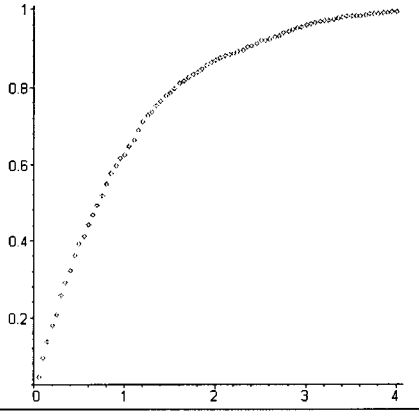
[19] Dadas as 1000 amostras de duração do evento votação, foi possível calcular a média em 2500 ms. Este cálculo é iterativo, ou seja, mantido no decorrer do experimento, conforme a fórmula da Figura

duração da votação obtidas, como uma função de distribuição de probabilidades. Portanto, a *coordenada X* indica medidas de duração das votações, de 0 ms até 10000 ms (4 desviões padrões da variável aleatória para a duração média) e a *coordenada Y* indica o percentual de amostras de duração das votações cujo tempo gasto foi igual ou inferior ao valor indicado no eixo X.

Os eventos de votação são independentes e o resultado positivo ou negativo para a solicitação de bloqueio do recurso é indiferente para a análise do algoritmo. No entanto, consideramos sempre todos os votos positivos para a solicitação de bloqueio de recurso para não permitir que o fator de aleatoridade *voto proferido* (uma variável aleatória de Bernoulli) influencie nos gráficos da simulação. Pelo mesmo motivo, consideramos a topologia da rede estável, visto que a propagação de eventos de modificação desta pode atrasar a resposta de alguns nós e prolongar a duração da votação. Em síntese, queremos apenas medir o impacto de se conhecer previamente o valor da cardinalidade ao se executar uma votação de Jajodia e Mutchler em um ambiente de rede com alta latência de *jitter* como o Peer-to-Peer.

Simulação 1: Um único nó servidor.





Número de Votações: 1000
Duração Média de Votação (μ): 2500 ms
Número de Nós/Votos: 1

Função de Distribuição (X de 0 a 4μ)
$F_X(1\mu) \approx 0.65$, $F_X(2\mu) \approx 0.85$,
$F_X(3\mu) \approx 0.95$, $F_X(4\mu) \approx 0.99$.

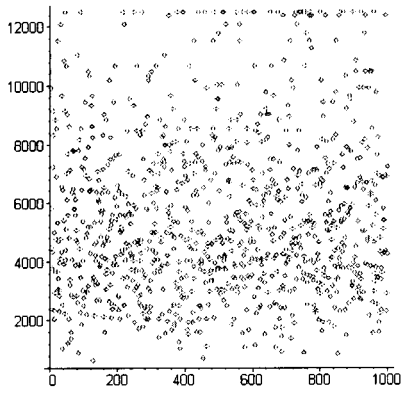
12. Como o timeout é definido em função desta média, em média, seu valor foi de 12000 ms. Todas as votações que não concluíram dentro deste prazo foram abortadas, e estão assinaladas com os pontos em vermelho.

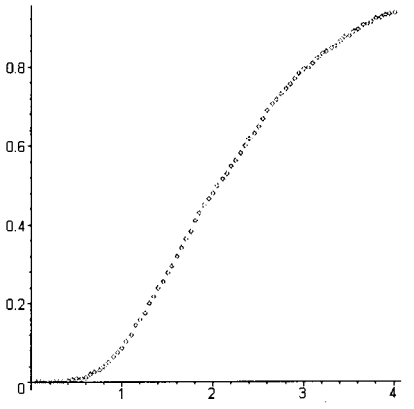
Taxa de Nós/Votos para Quorum: 1.0
Todos os votos eram positivos para o bloqueio.

Figura 14 – Gráficos da Simulação 1.

Nestas condições, a votação consta de um RTT, de uma única troca de mensagens. Por essa razão, as amostras para a duração da votação são descritas por uma distribuição exponencial de média 2500.

Simulação 2: Configuração para máxima tolerância à falhas.





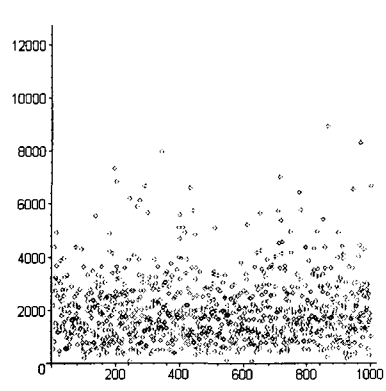
Número de Votações: 1000
Duração Média de Votação (μ): 2500 ms
Número de Nós/Votos: 10
Taxa de Nós/Votos para Quorum: 1.0
Todos os votos eram positivos para o bloqueio.

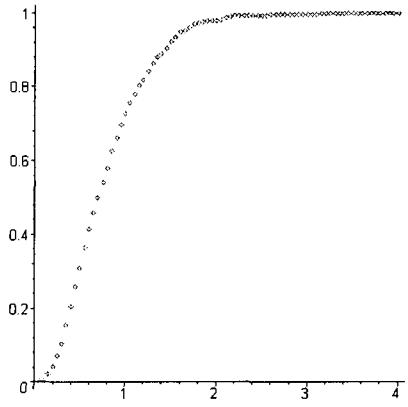
Função de Distribuição (X de 0 a 4μ)
$F(1\mu) \approx 0.14$, $F(2\mu) \approx 0.36$,
$F(3\mu) \approx 0.72$, $F(4\mu) = 0.91$.

Figura 14 – Gráficos da Simulação 2.

Dado que a obtenção de todos os votos continua sendo requisito para o quorum, mesmo que fosse possível anunciar o resultado da votação antes disso, mais votações têm sua duração determinada pelo nó com maior RTT, ou seja, as durações cada vez mais próximas do valor de timeout. Em outras palavras, esta é a configuração de pior desempenho para o sistema proposto, pois corresponde a executar o algoritmo de Jajodia e Mutchler sem a determinação prévia do parâmetro *cardinality*.

Simulação 3: Configuração para máxima performance.





Número de Votações: 1000
Duração Média de Votação (μ): 2500 ms
Número de Nós/Votos: 10
Taxa de Nós/Votos para Quorum: 0.51
Todos os votos eram positivos para o bloqueio.

Função de Distribuição (X de 0 a 4μ)
$F(1\mu) \approx 0.69$, $F(2\mu) \approx 0.99$,
$F(3\mu) \approx 1.0$, $F(4\mu) = 1.0$.

Figura 15 – Gráficos da Simulação 3.

É nesta configuração que os resultados das adaptações realizadas ao algoritmo de Jajodia e Mutchler são mais visíveis, pois com a chegada de "metade mais um" de votos positivos a votação pode ser encerrada. Praticamente todas (99%) as votações realizadas terminaram com até o dobro da duração média. Em contraste, na simulação anterior, mais da metade (64%) das votações terminaram com mais que o dobro da duração média. Em outras palavras, esta é a configuração com melhor desempenho para o sistema de controle de concorrência em redes P2P.

Note que o esquema de votação da proposta, bem como os demais da classe de votação descentralizada, é adequado para aplicações de pequena escala. A aplicação tema desta dissertação, um mecanismo de controle de concorrência para redes P2P, pertence a esta categoria. De fato, não é viável aplicar algoritmos de controle de concorrência pessimista em aplicações de larga escala, e nem é objetivo destas aplicações se alinharem a todos os requisitos típicos de bancos de dados.

Assim, embora resistente à alta latência, ao alto *jitter*, e ao particionamento da rede, o desempenho final deste esquema é influenciado também pelo número de votantes e, principalmente, pelas diferenças de desempenho dos canais de comunicação aos quais estes têm acesso. A carga de utilização do sistema, ou seja, a taxa de

solicitações de bloqueio emitidas, também influencia fortemente o fator de avleatoriedade do voto e, portanto, o desempenho geral do sistema.

5.4 – Interpretação da Aplicação do Esquema de Votação no Sistema de Gerência de Dados Proposto

O algoritmo de votação baseada em pesos é um algoritmo de sincronização; seguindo critérios baseados nas suas variáveis de controle (votos para leitura e escrita, pesos, e etc).

Sendo um algoritmo de controle de concorrência pessimista, ele não permite operação mediante particionamento da rede, trocando disponibilidade do sistema por forte garantia de consistência da base de dados. Abordagens otimistas poderiam resolver estas inconsistências por fusão (*merge*) ou recuperação do estado anterior à inconsistência (*rollback*). Tais operações, no entanto, são custosas e tradicionalmente difíceis de implementar [Rodrig 2003], especialmente o amálgama de versões conflitantes, a exemplo de sistemas de controle de versão.

Como um algoritmo de controle de concorrência, ele estabelece o bloqueio de todas as cópias de um recurso compartilhado ao longo do tempo. Uma vez conquistado o quorum, o peer solicitante possui direitos exclusivos de execução de operações sobre este objeto; a serem executadas em seqüência após o *commit*. Como visto, este seria apenas um componente do sistema de gerência de transações.

Porém, considerando as bases de dados federadas[20] como cópias de um mesmo recurso, a semântica do efeito deste algoritmo não é bloquear a base para uso exclusivo,

[20] Ou seja, estamos considerando bases de dados isoladas que compartilham um mesmo esquema e, portanto, podem processar consultas isoladamente; porém, visando serem seus resultados parte de uma consulta global, à federação propriamente.

mas marcar o início de uma transação. Temos, portanto, um *meta-banco de dados*, visto que os serviços que caracterizam o mesmo são providos pelos bancos de dados individualmente, e o efeito desejado é alcançado apenas porque as operações são executadas na mesma seqüência em todos os peers. O algoritmo de votação baseada em pesos, portanto, age como um relógio lógico para ordenação dos pedidos de transação, não para as operações sobre a base de dados em si, emitidas dentro de uma transação isoladamente. A serialização das operações de todas as transações que ocorrem em paralelo, sobre uma mesma base, fica a cargo dos SGBDs individualmente. Note também que por esse mesmo motivo, a gerência de deadlocks é realizada pelo SGBD, a menos que mais de uma base de dados (recurso compartilhado) seja alocada para a transação, neste caso, necessitando da política para evitar deadlocks, em nível de alocação de recursos – por exemplo, alocação dos recursos, previamente tidos como necessários, em ordem crescente do número de identificação universal (ID).

A confirmação da transação (commit) deve ser conduzida por um algoritmo centralizado, o 2PC. Isso porque, para uma transação especifica, o peer cliente exerce o papel de coordenador. Durante o período de operação da transação, os metadados das réplicas bloqueadas permanecem o mesmo. Como falhas de nós durante o processo podem levar a base de dados a um estado inconsistente, é necessário um controle de concorrência para confirmação da atualização da base. Apenas as réplicas bloqueadas, e ainda ativas, estão sujeitas a este protocolo de confirmação de transação. Portanto, esta é mais um requisito a ser atendido por um mecanismo de controle de presença dos peers.

Por fim, vale ressalvar que os esquemas de votação garantem que dois pedidos de bloqueio ou de execução de transação, cada um emitido por dois peers distintos, sejam atendidos, na mesma ordem, em todos os peers ativos da rede. No entanto, não há garantias sobre qual dos dois pedidos será o primeiro a ser atendido. Isso depende de quem obterá maioria dos votos primeiro, e é um evento imprevisível. No entanto, os peers que ocupam uma posição central na rede, ou seja, aqueles que têm a menor distância média para todos os demais peers, tendem a obter o quorum mais rapidamente. Este problema será retomado adiante.

5.5 – Conclusão

Como visto, o protocolo de controle de concorrência projetado é genérico, servindo para qualquer tipo de aplicação e recurso compartilhado, desde que estes necessitem efetivamente de um controle pessimista, descentralizado e resistente a falhas de peers na rede. Disponibilizar uma federação de bancos de dados foi motivação deste trabalho, mas poderia ter sido controle de versão de arquivos, gerência de utilização de recursos computacionais, edição colaborativa de documentos ou outra qualquer.

Diversos problemas, no entanto, não foram tratados. Nossa abordagem para controle de concorrência é direcionada a aplicações de pequena escala. Existe um limite máximo para o número de peers e para a latência do canal de comunicação dos mesmos para que o sistema não tenha seu desempenho deteriorado a ponto de tornar-se inutilizável — a arquitetura proposta funciona bem para a aplicação de federação de banco de dados, mas não é solução para todas as aplicações possíveis. Técnicas de detecção ou prevenção de deadlocks mais sofisticadas podem ser implementadas, bem como o sistema de monitoramento de presença também poderia ser sofisticado para aplicações de larga escala — um sistema de *Instant Messaging* sem servidor, por exemplo.

Capítulo 6 – Conclusão e Trabalhos Futuros

Como dito, a classe de sistemas Peer-to-Peer herda os principais desafios da área de Banco de Dados e de Busca e Recuperação de Informação. Muito da pesquisa realizada até aqui envolve a migração e adaptação de técnicas de bancos de dados distribuídos, ontologias, indexação de informação, entre outras.

Este trabalho apenas alcança a ponta de um crescente *iceberg*. O problema de gerência de transações distribuída é fundamental, e encontra aplicação em diversas áreas, como banco de dados distribuídos, sistemas de transporte de mensagens, sistemas de computação distribuída, sistemas de controle de versão e configuração, editores colaborativos, sistemas de memória compartilhada, entre inúmeros outros. Todos estes problemas já foram resolvidos segundo o paradigma Cliente/Servidor, porém, os requisitos específicos de usuários de sistemas Peer-to-Peer (anonimidade, resistência à censura, disponibilidade alta e adaptativa, etc) são reais e já demandam soluções de qualidade.

O momento em que este trabalho foi iniciado era bastante oportuno. As pesquisas em Gerência de Dados Distribuídos começavam a se preocupar com o alinhamento aos requisitos de bancos de dados, como pudemos ver em projetos como o Oasis e APPA. Adicionalmente, a plataforma JXTA, com o lançamento de sua segunda versão em maio de 2003, já ensaia a se firmar como *língua franca* para os peers da rede. Felizmente, muito ainda há para ser feito. Tanto o COPPEER quanto o COPPEER-DB abrem espaço para muitos novos trabalhos, bem como exigem sua própria evolução.

6.1 – COPPEER

O primeiro framework para desenvolvimento de aplicações Peer-to-Peer sobre JXTA foi lançado pela University of Lancaster durante a execução deste trabalho. Porém, como uma solução completa, abordando alguns dos principais aspectos do

desenvolvimento aplicações (interface gráfica, arquitetura de aplicação e comunicação), o COPPEER apresenta-se, para a plataforma JXTA, ainda sem concorrência.

A arquitetura de apresentação das aplicações – mais precisamente, desenho de interfaces gráficas – é baseado em no pacote Thinlet[21] modificado para alguns requisitos próprios do COPPEER, estando, inclusive, em um dos pacotes com os arquivos-fonte da plataforma. A arquitetura de aplicação é baseada na implementação do padrão *Inversion of Control* do pacote Spring Framework[22]. A arquitetura de comunicação foi projetada por nós, estendendo a arquitetura padrão fornecida pelo JXTA. Ainda sobre esta arquitetura de comunicação, foi implementado o protótipo do esquema de votação dinâmica. Todos estes elementos residem sobre a versão 2.1 do MyJXTA[23], que permite a implantação de aplicações usuárias destes recursos na forma de plug-ins.

No entanto, nosso framework precisa ainda explorar mais os recursos de interação com outros peers. A localização e instalação sob demanda de plug-ins são, naturalmente, um próximo passo. Porém, ela traz consigo preocupações quanto à segurança. Ainda que, assim como nas aplicações desktop, a maioria dos usuários não seja capaz de verificar se o programa que está usando contém código malicioso, ela possui meios de verificar a identidade do fornecedor e a integridade do software (assinaturas digitais), controlar o acesso do software à rede (firewalls), instalar extensões e atualizações por sugestão do próprio software, entre outros procedimentos. Assim, também a área de Sistemas Adaptativos certamente tende a migrar para o ambiente Peer-to-Peer em face dos seus evidentes benefícios na disponibilização e na localização de recursos de software.

6.2 – COPPEER-DB

[21] http://thinlet.sourceforge.net/

[22] http://www.springframework.org/

[23] http://myjxta2.jxta.org/

O COPPEER-DB é seguramente parte integrante do COPPEER. Porém, apesar do nome, o COPPEER-DB ainda não é um banco de dados. Nós projetamos e implementamos o módulo de controle de concorrência para um sistema de gerência de transações distribuída, de propósito geral. Como citado na conclusão do respectivo capítulo, há ainda o que ser pesquisado para que ele possa receber esta denominação.

A proposta de esquema de votação dinâmica descentralizada e tolerante a falhas constitui, essencialmente, da idéia de determinar o parâmetro de cardinalidade antes do início da votação, ao contrário do esquema de votação de Jajodia e Mutchler original, que determina o parâmetro após o recebimento de todos os votos e comunica seu valor na terceira e última etapa do algoritmo, o envio da mensagem de confirmação do quorum. Para tanto, propomos um mecanismo de controle de presença, que mantém a contabilidade dos nós ativos na rede para servir de valor da cardinalidade. Além disso, devido à possibilidade de ocorrência de particionamentos da rede distintos mas sucessivos, intercalados pela obtenção de dois quoruns de votação, percebemos que seria interessante estender o mecanismo de montoramento de presença para um mecanismo de monitoramento da topologia da rede. O mecanismo projetado, então, é similar ao mecanismo de roteamento *Link-State*, com algumas modificações necessárias para a diferente finalidade a que se propõe: determinar os nós onde há risco de particionamento da rede.

Quanto ao esquema proposto, é interessante perceber que ele constituído de dois módulos distintos: o esquema de votação e o mecanismo de monitoramento. Este último provê informações para a tomada de decisão do valor da cardinalidade usada pelo primeiro. O monitoramento em si pode ser apenas da presença dos nós, ou sua extensão, que monitora a topologia da rede. Este último, apesar de implicar em uma implementação muito mais complexa, elimina hipóteses de falsa indisponiblidade do sistema, que impedem a obtenção do quorum. No entanto, em termos de melhoria da performance do esquema de votação no ambiente Peer-to-Peer, ambos os tipos de monitoramento são equivalentes. Afinal, o ganho de performance está justamente em determinar o valor de cardinalidade na primeira etapa da votação (envio da convocação para votação), e não na terceira (envio da mensagem de confirmação do quorum). A

média e variância do tempo entre estas etapas são mínimas em implementações do esquema de Jajodia e Mucthler em redes locais – o ambiente onde rodam os sistemas de federação a que se aplica o algoritmo – mas é consideravelmente alto em redes Peer-to-Peer.

Duas frentes importantes encontram-se abertas no contexto do grupo de pesquisas do COPPEER: a busca por algoritmos de controle de concorrência mais eficientes, que permitam ampliar a escala da rede sob gerência, e o projeto de uma heurística de replicação de recursos visando à garantia da completude da consulta no caso de bases de dados distribuídas.

6.3 – Integração COPPEER e COPPEER-DB

O COPPEER-DB pode prover outras funcionalidades extremamente necessárias a aplicações Peer-to-Peer complexas. Por exemplo, para usuários serem capazes de gerenciar objetos compartilhados com esse sistema, ainda resta um módulo a ser projetado: um serviço de indexação, com interface de busca disponível para aplicações e para os usuários. Afinal, todos os procedimentos são realizados em função do identificador único do objeto compartilhado; e esta informação não é muito significativa para os usuários humanos deste sistema. O ambiente Peer-to-Peer induz a que muitos recursos computacionais necessitem de gerência, e o tratamento para cada tipo de recurso pode variar em alguns detalhes. São chaves públicas, documentos certificados digitalmente, arquivos de mídia, plug-ins, recursos computacionais; cada um com requisitos de utilização e segurança distintos. A oportunidade para unificar a gerência de todos os recursos computacionais é um avanço que não pode ser postergado, em face da proliferação deste tipo de solução em diversas áreas: arquivos (NFS, NETBIOS, Gnutella, JXTA CMS), gerência de chaves criptográficas (ISAKMP, IKE, OAKLEY, PGP), gerência de recursos de redes (SNMP), serviços de diretório de propósito geral em redes (LDAP, X.500) e para recursos de software (JNDI, UDDI), entre outros.

Para o COPPEER, a indexação de recursos é necessária para a gerência e instalação de aplicações sob demanda. O trabalho da University of Lancaster citado segue uma abordagem centralizada para indexação dos plug-ins e outros recursos compartilhados. Seguindo a filosofia do controle de concorrência descentralizado, o COPPEER-DB poderia permitr uma indexação totalmente distribuída e com garantias de manutenção da consistência dos metadados e recursos propriamente ditos. O desenvolvimento colaborativo de plug-ins poderia ser viabilizado pela própria plataforma COPPEER.

Alinhando-se aos requisitos típicos de SGBDs, o COPPEER-DB poderia tornar realidade o conceito de que a rede é o banco de dados. Ou seria de que o banco de dados é a rede?

Referências

[Akbarinia 2004] Akbarinia, R., Martins, V., Pacitti, E., Valduriez, P.: *Replication and Query Processing in the Appa Data Management System*, Int. Workshop on Distributed Data & Structures (WDAS), Lausanne, 2004.

[Alsberg 1976] Alsberg, P., Day, J.: *A Principle for Resilient Sharing of Distributed Resources*, Proc 2nd IEEE Int. Conf. on Software Engineering, 1976.

[Arumugam 2001] Arumugam M., Sheth A., and Arpinar I.: *Towards Peer-to-Peer Semantic Web : A Distributed Environment for Sharing Semantic Knowledge on the Web*, Technical report, Large Scale Distributed Information Systems Lab, University of Georgia, 2001.

[Babaogle 2002] Babaoglu, O., Meling, H., Montresor, A.: *Anthill: A Framework for the Development of Agent-Based Peer-to-Peer Systems*, In Proceedings of the 22th International Conference on Distributed Computing Systems (ICDCS '02), Vienna, Austria, July 2002.

[Barbosa 1995] Barbosa, V.: *An Introduction to Distributed Algorithms*, MIT Press, 1995.

[Bass 2003] Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice, 2nd Edition, Addison-Wesley Professional, 2003.

[Beckett 2004] Beckett, D.: *RDF/XML Sytnax Specification*, World Wide Web Consortium (W3C), Februrary 2004.

[Berger 2003] Berger F., Leist Y.: *The Xnap Handbook*, http://xnap.sourceforge.net/ , 2003.

[Berners-Lee 2001] Berners-Lee T., Hendler J., Lassila O.: *The Semantic Web*, Scientific American, May 2001.

[Brickley 2004] Brickley, D.: *RDF Vocabulary Description Language (RDF Schema)*, World Wide Web Consortium (W3C), Februrary 2004.

[Bolosky 2000] Bolosky, W., *et al*: *Feasibility of a Serverless Distributed File System Deployed on an Existing Set of Desktop PCs*, Proc of ACM Sigmetrics, 2000.

[Borghoff 2000] Borghoff, U., Schlichter, J.: *Computer-Supported Cooperative Work – An Introduction to Distributed Applications*, Springer-Verlag, 2000.

[Cetintemel 2001] Cetintemel, U., *et al*: *Support for Speculative Update Propagation and Mobility in Deno*, Proc of 21st IEEE Int Conference on Distributed Computing Systems, 2001.

[Chandy 1983] Chandy, K., Misra, J., Haas, L.: *Distributed Deadlock Detection.* ACM Trans. Comput. Syst. 1(2): 144-156 (1983).

[Connoly 2001] Connolly, D., Harmmelen, F., Horrocks, I., *et al*: *DAML+OIL Specification Reference Description*, World Wide Web Consortium (W3C), December 2001.

[Dabek 2001] Dabek F., Brunskill E., Kaashoek F., Karger D.: *Building Peer-to-Peer Systems with Chord, a Distributed Lookup Service*, MIT Laboratory for Computer Science, 2001.

[Dahl 1972] Dahl, Dijkstra, Hoare: Structured Programming, Academic Press, 1972.

[Davcev 1985] Davcev, D., Burkhard, W.: *Consistency and Recovery Control for Replicated Files*, SOSP 1985: 87-96.

[Dean 2004] Dean, M., Schreiber, G., *et al*: *OWL Web Ontology Language Reference*, World Wide Web Consortium (W3C), Februrary 2004.

[Druschel 2001] Rowstron, A., Druschel, P.: *Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems*. In R. Guerraoui, (ed.), Proc. Middleware 2001, volume 2218 of Lect. Notes Comp. Sc., pp. 329–350, Berlin, Nov. 2001. Springer-Verlag.

[Ehrig 2003] Ehrig M., Tempich C., *el al*: SWAP: *Ontology-based Knowledge Management with Peer-to-Peer Technology*, Proceedings of the 1st National Workshop Ontologie-basiertes Wissensmanagement (WOW2003).

[Fischer 1985] Fischer, M., Lynch, N.: *Impossibility of distributed consensus with one faulty process*, JACM, 32(2):374--382, 1985.

[Garcia-Molina 1984] Garcia-Molina, H., Bárbara, D.: *Optimizing the Reliability Provided by Voting Mechanisms,* Proc. 4th IEEE Int. Conf. on Distributed Computing Systems, San Francisco, CA, 1984.

[Gifford 1979] Gifford, D.: *Weighted voting for replicated data*, In Proceedings of the 7th Symposium on Operating Systems Principles, Pacific Grove, CA, 1979.

[Gray 1978] Gray, J.: *Notes on database operating systems*. In Operating Systems: An Advanced Course, Lecture Notes in Computer Science, volume 60, pages 393-481. Springer Verlag, Berlin, 1978.

[Grand 1999] Grand, M.: *Patterns in Java*, Wiley, 1999.

[Gribble 2001] Gribble S., Halevy A., Ives Z., Rodrig M., Suciu D.: *What can databases do for Peer-to-Peer*, WebDB, 2001.

[Jajodia 1987] Jajodia, S., Mutchler, D.: *Dynamic Voting*. In ACM SIGMOD Int'l Conf. on Management of Data, number 3, pages 227--238, 1987.

[Johnson 2004] Johnson, R., *et al*: *Spring - Java/J2EE Application Framework Reference Documentation*, http://www.springframework.org/documentation.html.

[Junginger 2002] Junginger, M., Lee, Y.: *The Multi-Ring Topology – High-Performance Group Communication in Peer-to-Peer Networks*, Proceedings of the Second International Conference on Peer-to-Peer Computing (P2P'02), IEEE, 2002.

[Haas 1997] Haas L., Kossmann D., Wimmers E., Yang J.: *Optimizing Queries Accross Diverse Data Sources*, Proceedings of the Conference on Very Large Data Bases (VLDB), August 1997.

[Haas 2002] Haas L., Lin E., Roth M.: *Data integration through database federation*, IBM Systems Journal (vol 41 no 4), 2002.

[Hapner 1999] Hapner, M., Burridge, R., Sharma, R.: *Java Message Service*, Sun Microsystems, Nov. 1999.

[Hardy 1979] Hardy, G., Weight, E.: *An Introduction to the Theory of Numbers*, 5th ed. Oxford, England: Oxford University Press, pp. 354-355, 1979.

[Heflin 2000] Heflin J., Hendler J.: *Searching the Web with SHOE*, AAAI-2000 Workshop on AI for Web Search. 2000.

[Hillmann 2003] Hillman, D.: *Using Dublin Core*, Dublin Core Metadata Initiative, 2003.

[Kaashoek 2001] Dabek, F. Kaashoek, M. *et al*: *Wide-area Cooperative Storage with CFS*, Proc of the 18th ACM Symposium on Operating System Principles, 2001.

[Kato 2002] D. Kato.: *GISP: Global Information Sharing Protocol - A Distributed Index for Peer-to-Peer Systems*, In Proceedings of 2nd International Conference on Peer-to-Peer Computing, Sweden, 2002.

[Kistler 1992] Kistler, J., Satyanarayanan, M.: *Disconnected Operation in the Coda File System*, ACM Transactions on Computer Systems, 1992.

[Kubiatowicz 2003] Kubiatowicz, J.: *Extracting Guarantees from Chaos*, Communications of the ACM, Vol 46, No 2, Feb 2003.

[Kurose 2000] Kurose, J., Ross, K.: *Computer Networking – A Top-Down Approach Featuring the Internet*, Addison-Wesley, 1996-2000.

[Lamport 1978] Lamport, L.: *Time, clocks, and the ordering of events in distributed systems*, Communications of the ACM 21(7):558--565, July 1978.

[Lamport 1982] Lamport, L., Shostak, R., Pease, M.: The Byzantine Generals Problem, ACM Transactions on Programming Languages and Systems, 4 (3), pp.382-401, July 1982.

[Lawrence 1999] Lawrence, S., Giles, C.: *Accessibility and Distribution of Information on the Web*, Nature 400(6740): 107-109, July 8, 1999.

[Lynch 1992] Lynch, N.: *Distributed Algorithms – Lecture Notes*, MIT, 1992.

[LTSC 2002] IEEE Learning Technology Standards Committee: *Learning Object Metadata Standard*, http://ltsc.ieee.org/wg12/index.html, 2002.

[Martin 1996] Martin, R.: *The Dependency Inversion Principle*, Object Mentor C++ Report, 1996.

[Minoura 1982] Minoura, T., Wiederhold, G.: *Resilient Extended True-Copy Token Scheme for a Distributed Database System*, IEEE Transactions on Software Engineering, 1982.

[Mukherjee 2002] Mukherjee A., Esfandiari B., Arthonrne N.: *U-P2P: A Peer-to-Peer System for Description and Discovery of Resource-sharing Communities*, 2002.

[Nejdl 2002] Nejdl W., Wolf B., Staab S., Tane J.: *Edutella: Searching and Annotating Resources within an RDF-based P2P Network*, Semantic Web Workshop, 2002.

[Ng 2002] Ng W., Ooi B., Tan K.: *BestPeer: A self-configurable Peer-to-Peer system*. Proceedings of the 18th International Conference on Data Engineering, page 272, San Jose, CA, April 2002.

[Ooi 2003] Ooi B., Tan K., Zhou A., Goh C., Li Y., *et al* : *PeerDB: Peering into Personal Databases*, SIGMOD 2003, June 9-12, 2003, San Diego, CA.

[Ooi, Shu 2003] Ooi B., Shu Y., Tan K.: *DB-Enabled Peers for Managing Distributed Data*, APWeb 2003.

[Oszu 1999] Oszu, T., Valduriez, P.: *Distributed Databases*, Prentice Hall, 1999.

[Ratnasamy 2002] Ratnasamy, S.: A Scalable Content-Addressable Network, Ph.D. Thesis, October 2002.

[Rhea 2003] Rhea, S., Eaton, P., Geels, D., *et al*: *Pond: the OceanStore Prototype*, Appears in Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST '03), March 2003.

[Rivest 1992] Rivest, R.: *RFC 1321 – The MD5 Message-Digest Algorithm*, MIT Laboratory for Computer Science and RSA Data Security Inc., April 1992.

[Rodrig 2003] Rodrig, M., LaMarca, A.: *Decentralized Weighted Voting for P2P Data Management*, MobiDE'03, September 2003, San Diego, CA, USA.

[Roussopoulos 2004] Roussopoulos, M., Baker, M., Rosenthal, D., *et al*: *2 P2P or Not 2 P2P*, IPTPS, February 2004.

[Roth 1997] Roth M., Schwarz P.: *Don't Scrap It, Wrap It! A Wrapper Architecture for Legacy Data Sources*, Proceedings of the Conference on Very Large Data Bases (VLDB), August 1997.

[Silberschatz 1999] Silberschatz, A.: *Sistema de Banco de Dados*, Editora Makron Books, 1999.

[Skeen 1982] Skeen, D.: *A Quorum-Based Commit Protocol*, Berkeley Workshop, 1982.

[Stoica 2001] Stoica, I., Morris, R., Karger, D., Kaashoek, M., Balakrishnan, H.: *Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications*. In Proc. SIGCOMM, San Diego, CA, Aug. 2001. ACM.

[Tang 1990] Tang, J.: *Voting Class – An Aproach to Achieving High Availability for Replicated Databases*, Proc 2nd Int. Symp. on Databases in Parallel and Distributed Systems, Dublin Ireland, 1990.

[Tanenbaum 2001] Tanenbaum, A.: Modern Operating Systems, Prentice Hall, 2nd Edition, 2001.

[Tanenbaum 2002] Tanenbaum, A., Steen, M.: *Distributed Systems: Principles and Paradigms*, Prentice Hall, 2002.

[Tarjan 1972] Tarjan, R.: *Depth-first search and linear graph algorithms*, SIAM Journal on Computing, 1:146--160, 1972.

[Thomas 1979] Thomas, R.: A majority consensus approach to concurrency control, ACM Transactions on Database Systems, 1979.

[Traversat 2003] Traversat, B., Arora, A., *et al*: *Project JXTA 2.0 Super-Peer Virtual Network*, Project JXTA, Sun Microsystems, May 2003.

[Terry 1995] Terry, D., *et al*: *Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System*, Proc. 15th ACM Symp on Operating Systems Principles, 1995.

[Walkerdine 2002] Walkerdine, J., Melville, L., Sommerville, Ian: *Dependability Properties of P2P Architectures*, Published in the proceedings of P2P2002.

[Walkerdine 2004] Walkerdine, J., Melville, L., Sommerville, Ian: *The P2P Application Framework*, Technical Report, Lancaster University, Lancaster, UK, April 2004.

[Waterhouse 2001] Waterhouse S.: *JXTA Search: Distributed Search for Distributed Networks*, Sun Microsystems Inc., 2001.

[Waterhouse 2002] Waterhouse S., Doolin D., Kan G., Faybishenko Y.: *Distributed Search in Peer-to-Peer Networks*, IEEE Internet Computing, 2002.

[Xiang 2003] Xiang X., Shi Y., Guo L.: *Rich Metadata Searches Using the JXTA Content Manager Service*, 2003.

[XQuery 2003] Boag S., Chamberlin D., Fernandez M., Florescu D., *et al*: *XQuery 1.0: An XML Query Language,* Technical report, World Wide Web Consortium (W3C), August 2003.

[Zhao 2001] Zhao, B., Kubiatowicz, J., Joseph, A.: *Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing.* Technical Report CSD-01-1141, Computer Science Division, University of California, Berkeley, Apr. 2001.