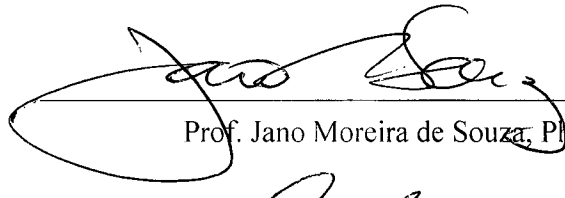
GARANTIA DE DISPONIBILIDADE EM AMBIENTE PEER-TO-PEER UTILIZANDO REPLICAÇÃO COORDENADA

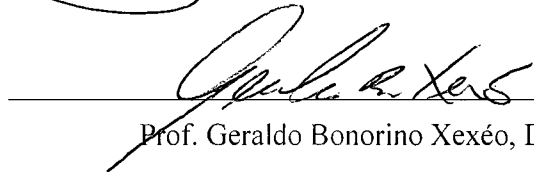
José Nogueira D' Almeida Junior

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.

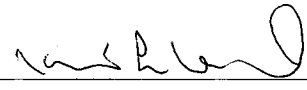
Aprovada por:



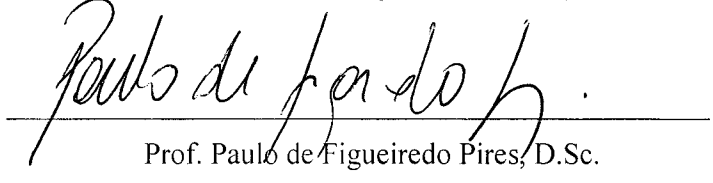
Prof. Jano Moreira de Souza, Ph.D.



Prof. Geraldo Bonorino Xexéo, D.Sc.



Profa. Marta Lima de Queirós Mattoso, D.Sc.



Prof. Paulo de Figueiredo Pires, D.Sc.

RIO DE JANEIRO, RJ – BRASIL

NOVEMBRO DE 2005

D' ALMEIDA JUNIOR, JOSÉ NOGUEIRA

Garantia de Disponibilidade em Ambiente Peer-to-Peer Utilizando Replicação Coordenada [Rio de Janeiro] 2005

XI, 113p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 2005)

Dissertação - Universidade Federal do Rio de Janeiro, COPPE

1. Banco de Dados Distribuídos
2. Sistemas Peer-to-Peer

I. COPPE/UFRJ II. Título (série)

Aos meus pais que tudo fizeram para eu chegar até aqui.

À minha namorada, Luciana, que acompanhou cada passo meu desta etapa.

À minha irmã, Marcelle, que está presente comigo aonde vou.

Agradecimentos

Aproveito este espaço para escrever meus agradecimentos a todos aqueles que estiveram presentes na minha vida até então. Para começar, agradeço a meus pais, que sempre se importaram em indicar os melhores caminhos a percorrer, do berço até hoje. Minha irmã, que me viu iniciar este trabalho, infelizmente não poderá me ver concluí-lo, o que não me impede de ter a certeza de que ela estaria muito feliz por esta conquista. Tenho plena consciência de que esta conquista não é apenas minha, mas um trabalho resultante de todo o empenho das pessoas que me cercam, da sociedade e suas boas escolas, do governo e suas instituições de qualidade que ainda existem. Meu namoro, ao contrário do que acontece em outros casos, foi crescendo à medida que esse trabalho foi evoluindo, o que me torna plenamente certo de ter encontrado a pessoa que me completa. Agradeço ao meu orientador, que além de me orientar academicamente, orientou minha carreira e, em tudo aquilo que fui procurar, obtive uma resposta ou um caminho. Cada professor do mestrado, meus colegas de mestrado, de projeto Star One, projeto na Marinha, todos tiveram um pedaço de participação nesta tese. Agradeço a Deus, ao criador, aquele que não vemos, que não entendemos, mas está presente, de alguma forma, escrevendo as linhas de código do universo.

José Nogueira D' Almeida Junior.

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

GARANTIA DE DISPONIBILIDADE EM AMBIENTE PEER-TO-PEER UTILIZANDO REPLICAÇÃO COORDENADA

José Nogueira D' Almeida Junior

Novembro/2005

Orientadores: Jano Moreira de Souza
Geraldo Bonorino Xexéo

Programa: Engenharia de Sistemas e Computação

As redes peer-to-peer, famosas por suas aplicações de troca de arquivos, são uma nova forma de computação distribuída. Uma de suas características é a volatilidade de seus membros. Criar mecanismos que tornem mais confiável tal ambiente é um desafio que pode ser auxiliado com a área de conhecimento de banco de dados distribuídos. Este trabalho apresenta uma abordagem de tolerância à falhas dos membros de uma rede peer-to-peer a fim de manter disponíveis os objetos, independentemente das oscilações que venham a ocorrer. O algoritmo proposto utiliza a replicação para gerar a redundância e a coordenação para ordenar os movimentos dos peers e seus objetos. A técnica utiliza eleições para escolher os membros mais aptos a realizarem a coordenação. Os resultados experimentais mostraram a importância de coordenar a replicação e listaram os principais critérios para realizar as eleições.

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

AVAILABILITY GUARANTEED IN PEER-TO-PEER ENVIRONMENT THROUGH COORDINATED REPLICATION

José Nogueira D' Almeida Junior

November/2005

Advisors: Jano Moreira de Souza
Geraldo Bonorino Xexéo

Department: Systems and Computing Engineering

Peer-to-peer networks, which are famous because its file-sharing applications, are a new way to make distributed computing. One of its features is the volatility of its members. The creation of new mechanisms to make more reliable this kind of environment is a challenge, which can be aided by distributed database knowledge area. This thesis presents one tolerant-failure approach to guarantee the availability of the objects in a peer-to-peer network. The proposed algorithm uses replication to generate redundancy and coordination to order the peers' actions and its objects. The technique uses elections to choose the best members, which will be the coordinators. The experimental results showed the importance of the coordination to replication and listed the main criteria to make the elections.

Sumário

Índice de Figuras

Índice de Tabelas

Capítulo 1 – Introdução

1.1 – Motivação

O cenário atual mostra que com os avanços recentes da computação e das telecomunicações, estamos diante de uma revolução tecnológica. A sociedade que outrora era industrial passa a ser a sociedade da informação, onde o ativo mais importante é o conhecimento. Os computadores pessoais são fortes ferramentas para a explicitação do conhecimento. Com a crescente popularização dos PCs, a quantidade de bytes/ano gerados em todo o globo tende a aumentar cada vez mais. Segundo estudos da consultoria especializada em armazenamento de dados Horison Information Strategies[1], em 2000, havia sete exabytes (sete vezes 2^60 bytes) de conteúdo digital no mundo. A previsão é que, em 2005, esse número chegue a 99,5 exabytes. Gerenciar todo este volume de informação não é tarefa trivial, o que abre um espaço para que a gestão do conhecimento conquistar a atenção do cenário atual.

O primeiro paradigma que ganhou espaço na busca e recuperação de informação foi o modelo centralizado (cliente/servidor). Podemos citar que o Google[2], um dos mais renomados sites para consulta na web, possui como missão "...organizar o enorme montante de informação disponível na web e no mundo". Segundo o próprio Google, seu índice contém mais de 1 bilhão de URLs, o que representa uma pequena parcela do conteúdo existente na web. De fato, é exigida uma crescente estrutura computacional para acompanhar a curva de geração de informação.

Dividir para conquistar é uma abordagem comum em computação. Indexar a enorme quantidade de informação no modelo centralizado concorre com novas propostas de indexação descentralizada. Conseqüentemente, um modelo descentralizado que vem atraindo a atenção da comunidade científica, assim como o interesse do meio

[1] http://www.horison.com/

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

corporativo, é a comunicação Peer-to-Peer. Segundo a empresa de pesquisa de mercado Frost & Sullivan, estima-se que o número de clientes Peer-to-Peer (P2P) corporativos nos EUA cresceu de maneira constante em 2005. A partir de então, o crescimento será maior, onde existe a previsão de cerca de 1.900.000 clientes P2P corporativos. Esta pesquisa ainda demonstra um salto no rendimento de vendas de U$ 42,8 milhões em 2004 para U$ 4,53 bilhões em 2007.

Li Gong, um diretor de engenharia da Sun Microsystems disse: "O termo peer-to-peer é aplicado a um amplo leque de tecnologias que aumentam bastante a utilização da informação, largura de banda e recursos computacionais na Internet. Freqüentemente, P2P adotam um modelo computacional que nem exclui nem cria dependências em pontos de controles centralizados".

A trajetória dos sistemas P2P foi iniciada com programas de troca de arquivos, com destaque para MP3[3]. A popularidade alcançada foi tamanha que este tipo de sistema de troca de músicas é apontado como um dos grandes responsáveis pela popularização de sistemas Peer-to-Peer.

É importante notar que houve o surgimento de diversos sistemas P2P, mas não houve uma padronização dos mesmos, havendo uma grande restrição em relação à interoperabilidade dos sistemas. Além disso, as aplicações P2P desenvolvidas não possuem o propósito de serem ampliadas a novas funcionalidades.

A Sun Microsystems, em abril de 2001, apresentou a proposta do JXTA[4]. O JXTA, acrônimo de *Juxtapose*, é um conjunto de protocolos abertos que possibilitam a troca de mensagens entre diferentes Peers. O conceito Peer do JXTA vem do mapeamento que acontece entre a rede física com seus diferentes dispositivos (PCs, laptops, palmtops, celulares, etc) e a rede virtual do JXTA, onde cada dispositivo é visto como um Peer. Desta maneira, a camada de protocolos JXTA é justaposta às demais camadas de transporte, rede ou até mesmo enlace Ethernet.

---

[3] O MP3 é um padrão para arquivos de músicas com alta taxa de compactação.

[4] http://www.jxta.org

Apesar de ser um projeto da Sun Microsystems, o JXTA não se restringe a implementação em Java. Atualmente existem implementações da API para C, Perl, Python, tal como está em andamento uma versão para .NET da Microsoft. Assim, o JXTA segue aquilo a que se propõe: ser um padrão para sistemas peer-to-peer, escalável, interoperável, ubíquo.

A especificação JXTA vem adquirindo cada vez mais adeptos. Atualmente são mais de 100 projetos listados pelo site oficial, o que não contabiliza outros projetos que existam em outros meios.

Um projeto baseado em JXTA que em breve entrará no site oficial da especificação é o COPPEER, o projeto Peer-to-Peer da COPPE. Seu principal propósito é servir como framework de desenvolvimento de aplicações P2P.

O COPPEER é desenvolvido em Java e estende as funcionalidades do JXTA. O framework da COPPE possui serviços que podem ser usados por seus plugins. Desta maneira, conseguimos atender alguns dos requisitos aqui colocados até então: padronização e interoperabilidade.

No entanto, a ubiqüidade[5], um outro requisito importante, não foi atendida ainda. O problema vai além do COPPEER e do JXTA. É um problema típico de redes P2P, descentralizadas. As redes P2P possuem como comportamento natural um padrão transiente de comunicação (KUBIATOWICZ, 2003). A escalabilidade do JXTA, comprovadamente melhorada em sua versão 2.0 (HALEPOVIC, DETERS, 2003), caracteriza um cenário onde há o tempo todo troca de mensagens, entrada e saída constante na rede de Peers, troca de recursos, falhas na comunicação, falhas nos Peers, entre outros.

Extrair garantias tais como disponibilidade (KUBIATOWICZ, 2003) de uma arquitetura peer-to-peer torna-se uma área de pesquisa interessante, pois controlar a natureza desse tipo de sistema requer abordagens elaboradas, complementares e

[5] O termo ubiqüidade se refere à propriedade de quem está a todo tempo em toda parte.

desenvolvimento de técnicas para contornar o problema. Os Bancos de Dados Distribuídos possuem problemas comuns a sistemas peer-to-peer (GRIBBLE et al. 2001) e têm sido fontes de consultas dos pesquisadores da área para aplicar suas técnicas na arquitetura P2P .

Um tipo de garantia a somar em uma arquitetura P2P é a garantia de disponibilidade dos objetos da rede, recursos ou dos dados do sistema. No caso do COPPEER, objetos podem ser arquivos, serviços, classes ou anúncios (*jxta advertisements*). Além dos Peers, objetos podem também estar associados a grupos de Peers. Portanto, dependendo do propósito de um grupo de Peers, é fundalmental existir garantia de disponibilidade de objetos.

É importante condicionar a garantia de disponibilidade a um grupo de Peers, de forma a existir grupos com garantias e grupos sem garantias (o COPPEER provê grupos com determinados serviços agregados), tendo em vista que garantir disponibilidade envolve custos computacionais e de rede.

Um fato desejável é que os peers se agrupem com uma finalidade comum, p.e., utilizar uma aplicação de edição colaborativa de ontologias, como COE (XEXÉO et al., 2004). O COE é um dos motivadores deste trabalho, pois certas ontologias são imprescindíveis ao grupo e precisam ser altamente disponíveis, o que requer um serviço especializado.

Esta dissertação se propõe a apresentar uma proposta de garantia de disponibilidade de objetos no COPPEER, o que é na verdade uma abordagem para tornar sistemas P2P mais confiáveis.

1.2 – Organização da Tese

Este trabalho é uma proposta de ferramenta computacional para apoiar outras aplicações desenvolvidas para o COPPEER. Para basear conceitualmente este trabalho, tornou-se necessário mostrar conceitos importantes nos capítulos 2, sobre banco de dados distribuídos e 3, sobre redes peer-to-peer. No capítulo 2, focamos ainda em

confiabilidade de bancos de dados distribuídos, envolvendo os conceitos de disponibilidade e replicação, assim como assuntos relacionando a consistência de réplicas. No capítulo 3 temos uma visão geral das redes peer-to-peer, numerando dezenas de sistemas existentes, classificando-os e entramos em estudo profundo sobre a plataforma que tende a padronizar o desenvolvimento para área. O capítulo 4 apresenta o COPPEER e a proposta de um Serviço de Garantia de Disponibilidade para aplicações desenvolvidas para o framework. Neste capítulo são mostradas as minúcias do modelo adotado, assim como os resultados encontrados em cada uma das quatro etapas computadas através do simulador COPS: uma fase exploratória, uma buscando o menor esforço, uma utilizando análise de regressão linear e última utilizando algoritmos genéticos. No capítulo seguinte, uma conclusão sobre o trabalho é feita bem como são mencionados os passos futuros desta pesquisa.

Capítulo 2 – Banco de Dados Distribuídos

2.1 – Conceitos Básicos

A concepção de Banco de Dados Distribuídos vem da união de duas áreas da computação até então isoladas: banco de dados e redes de computadores. Com a vertiginosa evolução das redes de computadores nos últimos tempos, em especial com a popularização da internet, os bancos de dados distribuídos vêm atingido cada vez mais nichos e aplicações distintas.

Os bancos de dados surgiram como uma forma de isolar os dados das aplicações, pois estas haviam que manipular fisicamente seus dados e grande parte do código era despendido para este fim. Desta forma, os bancos de dados centralizavam o controle dos dados de quaisquer aplicações que precisassem utilizar tais informações, caracterizando a *independência dos dados*. Os dados deixaram de ficar a critério da manipulação das aplicações, que por muitas vezes havia a redundância de dados nos diversos arquivos desconexos.

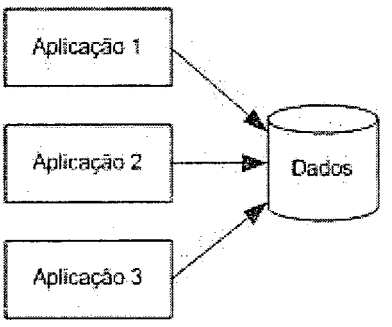


Figura 1: Aplicações com independência dos dados

As redes de computadores vêm, de maneira oposta aos bancos de dados tradicionais, descentralizar o processamento. Os bancos de dados somados as redes deixam de lado a centralização e passam a levar em conta a integração, que é o objetivo de qualquer banco de dados distribuído.

2.1.1 – O Processamento Distribuído

É comum encontrar o termo processamento distribuído na literatura de banco de dados distribuídos, assim como uma definição particular para o mesmo ou vários nomes para um mesmo conceito: computação distribuída, função distribuída, processamento de backend, etc. Em ÖZSU, VALDURIEZ (1999), é definido o termo sistema de computação distribuída como aquele que consiste em diversos elementos autônomos de processamento conectados por uma rede onde haja cooperação entre os membros (não necessariamente homogêneos) para realização de uma determinada tarefa.

Importante notar que não apenas a lógica de processamento pode estar sendo feita distribuída. A distribuição pode ser feita de acordo com a função, i.e., alguns membros podem receber tarefas distintas na realização de determinada atividade. Existe a distribuição por dados, onde determinados membros da rede possuem dados específicos. Por último, temos a distribuição feita pelo controle de execução das tarefas.

Uma questão fundamental é a motivação de distribuir. De uma maneira geral, nós humanos seguimos a filosofia de René Descartes[6] em sua regra da Análise, dividimos as grandes tarefas em pequenas tarefas para podermos trabalhar em cada um delas para termos o resultado final. Uma justificativa econômica para distribuir a computação é conseguir um maior poder computacional através do uso de vários elementos simultaneamente, de maneira coordenada e otimizada.

2.1.2 – O que é um Banco de Dados Distribuídos

Um banco de dados distribuídos pode ser definido com um conjunto de banco de dados interligados por uma rede (vide Figura 1). Um SGDB Distribuído promove ao usuário transparência no acesso aos dados, i.e., o usuário não precisa se preocupar em que membro da rede seus dados estão ao realizar uma determinada consulta. O termo

[6] René Descartes (1595-1650) foi um famoso matemático francês, cientista e filósofo, conhecido como "pai da filosofia moderna". Suas idéias, expostas principalmente em o "Discurso sobre o Método", são aplicadas até hoje. Autor da célebre frase: "Penso, logo existo".

Sistema de Banco de Dados Distribuído (SBDD) geralmente referencia o conjunto banco de dados distribuído e o SGDB distribuído.

A questão da localização dos dados de um banco de dados distribuídos é muito importante, pois vai impactar no desempenho do sistema como um todo e será tratada adiante. Quando um dado se localiza exclusivamente em um nó da rede há o que se denomina *fragmentação*. Quando um dado se encontra em mais de um nó existe a *replicação*.

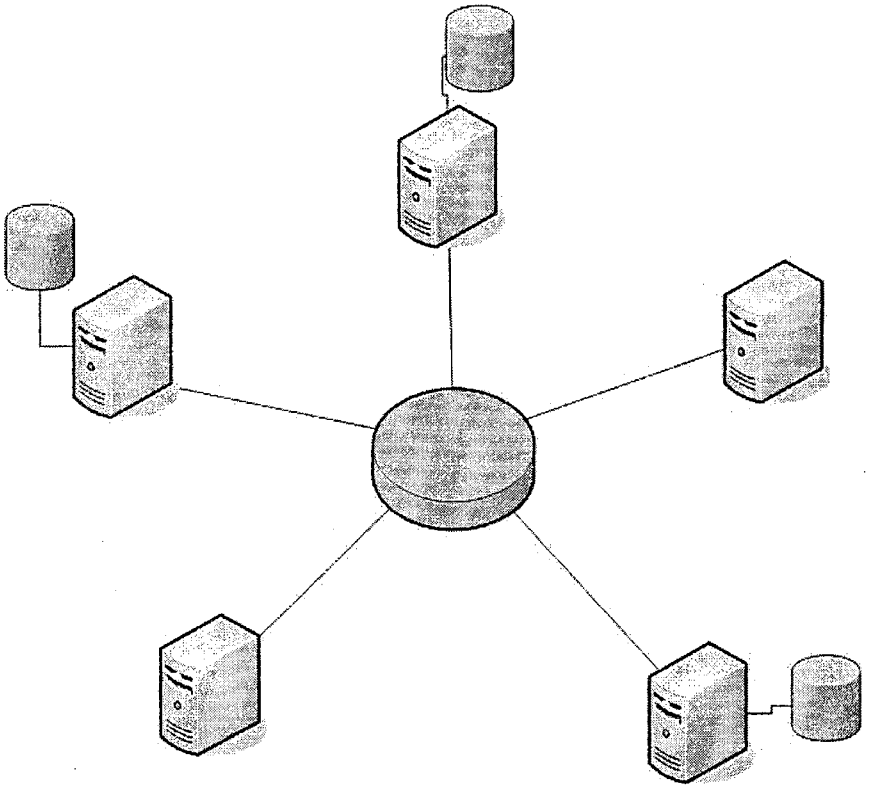


Figura 2: Exemplo de Banco de Dados Distribuído

2.1.3 – Objetivos de um SBDD

2.1.3.1 – Camadas de Transparência

O conceito de transparência é muito utilizado em computação e pode ser entendido como prover a separação de uma semântica de alto nível dos detalhes de implementação (ÖZSU, VALDURIEZ, 1999). Estes ficam ocultados do usuário.

Primeiramente, um SBDD possui como objetivo prover a *independência dos dados*, que ocorrerá quando o usuário tiver independência lógica e física dos dados. A independência lógica se refere à definição do esquema dos dados. Por exemplo: as colunas de uma tabela que são acessadas por uma aplicação devem permanecer inalteradas se houver uma alteração no esquema da tabela como o acréscimo de uma nova coluna. A independência física dos dados se refere ao fato do usuário não ter que manipular estruturas internas de dados que são implementadas pelo banco. Se o banco usa uma árvore B+ ou um vetor para armazenar deve ficar transparente para o usuário.

Um outro objetivo fundamental para um SBDD é a *transparência de rede*. O usuário deve estar livre de detalhes do funcionamento da rede e seus protocolos durante a utilização do sistema gerenciador de banco de dados distribuídos, que deve prover um resultado (exceto pelos delays que podem ocorrer da rede) similar a um SGDB.

A replicação torna-se necessária para atingir melhores taxas de desempenho, confiabilidade e disponibilidade. Uma vez que os dados estão replicados, deve ser analisada a questão da *transparência da replicação*. Uma vez que a replicação é transparente para o usuário, o sistema gerenciador de transações do SBDD torna-se muito mais complexo, ao contrário de quando a replicação fica a critério do usuário. No entanto, a transparência da replicação é a opção padrão e que se refere apenas a existência de réplicas e não sua localização, que é do domínio da transparência de rede.

O último tipo de transparência que deve existir é a *transparência de fragmentação*. É interessante que um SBDD transforme algumas de suas relações em fragmentos menores, aumentando assim parâmetros como desempenho, confiabilidade e disponibilidade. Existem dois tipos básicos de fragmentação: a *fragmentação horizontal*, onde uma relação original é divida em conjuntos de tuplas distintas; a *fragmentação vertical*, onde cada sub-relação é um subconjunto dos atributos da relação original. A transparência de fragmentação vai atuar de maneira que uma consulta a relação original será convertida (usando técnicas que fogem ao escopo deste trabalho) em *consultas de fragmentos* e montará posteriormente o resultado de cada fragmento.

É importante salientar que existe um inevitável compromisso entre a facilidade de uso e a dificuldade e overhead de oferecer elevados níveis de transparência.

2.1.4 – Desafios de um SBDD

Um SBDD implica na adição de problemas aos já existentes de um SGDB. A localização dos dados é um desafio que pode ser decomposto em três subproblemas relacionados:

1 – a escolha de uma das cópias quando for necessária uma consulta àquele dado.

2 – garantir que a atualização de uma das cópias se refletirá as demais;

3 – sincronizar as transações que ocorrem nos diferentes nós da rede.

Em comparação a um SGDB, podemos citar que um SBDD é mais complexo, é mais custoso (p.e. a manutenção de diferentes nós da rede para que o sistema como um todo funcione), é mais difícil de coordenar o controle que é distribuído e é potencialmente mais inseguro.

A questão da segurança em SBDDs herda das redes de computadores toda sua complexidade e desafios. Em um SGDB centralizado temos uma autenticação local. Em um SBDD, entram questões como localidade da autenticação, tráfego de informações sigilosas (como a senha do usuário), entre outros.

Segundo ÖZSU, VALDURIEZ (1999), para atingir todos os benefícios de se implantar um SBDD é preciso que alguns dos seguintes problemas sejam resolvidos:

1. Projeto de Banco de Dados Distribuídos – durante o projeto do SBDD, é preciso definir a posição dos dados, que basicamente pode estar particionados ou replicados. Caso seja feita a replicação, ela pode ser total (duplicação) ou parcial. A distribuição ótima dos fragmentos é em geral um problema NP completo e as soluções para o problema são

baseadas em heurísticas, o que também ocorre em uma rede peer-to-peer (KANGASHARJU et al., 2001).

2. Processamento distribuído de consulta
3. Gerenciamento de diretório distribuído
4. Controle distribuído de concorrência
5. Gerenciamento distribuído de impasse
6. Confiabilidade de SBDD
7. Suporte do sistema operacional
8. Banco de Dados heterogêneos

2.2 – Confiabilidade de SGDBs Distribuídos

A Confiabilidade, a Disponibilidade e a Replicação são termos inter-relacionados que precisam estar bem definidos. Além disso, existe a necessidade da definição de protocolos que irão tornar um Sistema Gerenciador de Banco de Dados Distribuídos mais confiável, mais disponível, explorando a replicação e a distribuição dos dados.

Diz-se que um Sistema Gerenciador de Banco de Dados Distribuídos é confiável quando este, mesmo quando um sistema subjacente não é confiável, torna possível processar solicitações (ÖZSU, VALDURIEZ, 1999). Em outras palavras, existe confiabilidade de um sistema quando é garantido o uso de um determinado serviço mesmo se um subsistema falha.

Uma discussão isolada sobre a confiabilidade de banco de dados poderia ser dada. No entanto, o SGDB é parte da computação distribuída, que envolve componentes de harwdware, software e rede, que constituem o ambiente.

2.2.1 – Defeito, erro e falha

Os termos confiabilidade e disponibilidade são aplicados amplamente na literatura, apesar de não haver um consenso sobre a definição precisa dos termos. A confiabilidade de um sistema depende da confiabilidade de seus subsistemas, ou componentes. Os componentes de um sistema envolvem diferentes níveis de complexidade e abstração: sinais elétricos, hardware, software, camadas de rede, aplicação, além da comunicação entre os diferentes componentes.

Um sistema recebe estímulos externos e reage a esses estímulos conforme um padrão determinado especificado em seu projeto. Uma resposta diferente da esperada é considerada um erro, proveniente de um defeito que resultado em uma falha, conforme mostra a figura abaixo:

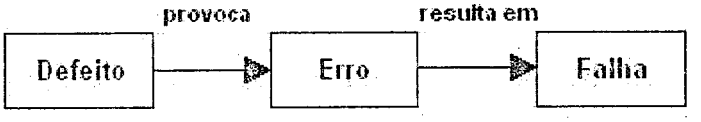


Figura 3: Eventos que levam a uma falha do sistema

Na maioria dos casos, localizar defeitos não é tarefa trivial. O tipo de falha pode servir de dica para detecção do defeito, que pode ser reparado de maneira automática ou por intervenção humana. Na literatura não se faz distinção entre defeitos permanentes e efeitos transientes (intermitentes). Defeitos de projetos encadeiam falhas de projetos. Este tipo de defeito constitui parte significativa das falhas dos sistemas.

2.2.2 – Confiabilidade e Disponibilidade

A confiabilidade é o termo referente à probabilidade de um sistema não apresentar falhas em um dado intervalo de tempo (ÖZSU, VALDURIEZ, 1999). De maneira geral, a confiabilidade é aplicada a sistemas que não podem ser reparados (como as sondas espaciais) ou a operação do sistema é fundamental de maneira que se torna intolerável a inatividade para tempo de reparo.

A confiabilidade pode ser descrita formalmente como:

$$R(t) = P(nenhuma.falha.em.[0,t] \mid 0\,falhas.em.t = 0)$$

Equação 1

Supondo que as falhas seguem uma Distribuição de Poisson, temos:

$$P(k.falhas.em.[0,t]) = \frac{e^{-v(t)}[v(t)]^k}{k!}$$

Equação 2

$$\text{onde } v(t) = \int_0^t u(x)dx$$

Equação 3

A função u(t) é chamada de função de risco, que descreve a taxa de falhas individual de cada componente do sistema. Logo, a confiabilidade para não haver nenhuma falha seguindo uma distribuição de Poisson é:

$$R(t) = e^{-v(t)}$$

Equação 4

A fim de calcular a confiabilidade do sistema todo, considerando que as falhas dos *n* componentes são independentes, basta calcular o produtório abaixo:

$$R_{Sistema}(t) = \prod_{i=1}^{n} R_i(t)$$

Equação 5

Uma vez definida e formalizada a confiabilidade, a disponibilidade pode ser definida como a probabilidade de que o sistema esteja operacional em um dado instante de tempo (ÖZSU, VALDURIEZ, 1999). A disponibilidade, ao contrário da

confiabilidade, se refere a sistemas que podem ser reparados. De uma maneira geral, é mais fácil se construir altamente disponíveis do que altamente confiáveis.

A fim de formalizarmos a disponibilidade, supondo que a chegada de falhas (λ) segue uma distribuição de Poisson e o reparo (μ) é descrito por uma distribuição exponencial, a disponibilidade de um sistema pode ser calculada por:

$$A = \frac{\mu}{\mu + \lambda}$$

Equação 6

Para efeito de simplificações no cálculo da disponibilidade e confiabilidade, são amplamente empregadas duas medidas: o tempo médio entre falhas (MTBF – mean time between failures) e o tempo médio para reparos (MTTR – mean time to repair). Na literatura também é citado o tempo médio até falhar (MTTF – mean time to fail).

2.2.3 – Falhas

As falhas são grandes responsáveis por motivar este trabalho. Um sistema, seja ele centralizado ou distribuído, está propenso à falhas. No caso de uma rede de peers, a todo momento um peer pode sofrer uma falha e ficar inoperável. Portanto, um sistema que se propõe a garantir a disponibilidade dos objetos distribuídos deve recorrer a técnicas de tolerância à falhas.

As falhas podem ser classificadas como brandas e severas (ÖZSU, VALDURIEZ, 1999). As falhas brandas, desde o início da computação, apresentam-se em maior quantidade que a severas. Cerca de 90% das falhas de hardware são brandas. Além disso, a maior parte das falhas de software é transiente, e por conseqüência, branda. Empiricamente, em muitos casos não há a necessidade de reparar o software, bastante reiniciá-lo para torná-lo estável novamente.

As duas abordagens fundamentais para lidar com as falhas em sistemas são a tolerância à falhas e a prevenção de falhas. A tolerância sabe que as falhas ocorrerão e

sabe como tratá-las. Já a prevenção tem como objetivo assegurar que o sistema não possuirá falhas.

A detecção de uma falha nem sempre é imediata, ou seja, existe um certo tempo entre a ocorrência da falha e sua detecção, portanto temos uma *falha latente*. Esse período é chamado de latência de erro. Existe também a métrica do tempo médio para detectar erro (MTTD – mean time to detect).

O princípio básico empregado em todos os projetos de sistemas tolerantes a falhas é a existência de componentes redundantes no sistema. Em uma rede peer-to-peer, os objetos se tornam tolerantes as falhas dos peers através da replicação dos mesmos.

2.2.4 – Falhas em Banco de Dados Distribuídos

Um Sistema Gerenciador de Banco de Dados Distribuídos deve ser capaz de tolerar falhas dos diferentes nós da rede. Para isso, existe o gerenciador de recuperação de banco de dados, que lida com quatro tipos de falhas (ÖZSU, VALDURIEZ, 1999): falhas de transação, falhas de sites (ou nós), falhas de disco e falhas no link.

Nas falhas de transação, a abordagem usual quando ocorre uma falha é abortar a transação. Desta maneira, a base de dados voltará ao estado anterior. Nas falhas de sites (o que num contexto P2P seria o peer), pode haver uma falha total (onde todos os sites falham) ou parcial. Ocorrida a falha, é impossível acessar o site. As falhas de disco ou mídia dizem respeito às falhas nos dispositivos de armazenamento que guarda o banco de dados.

Os três tipos de falhas citadas no último parágrafo são comuns tanto a SGDBs centralizados quanto distribuídos. Já a falha no link é peculiar aos SGDBs distribuídos, que dependem de um canal para se comunicarem. Um assunto interessante neste tipo de falha é o Particionamento da Rede (DAVIDSON et al., 1985), onde cada partição pode continuar a operar de maneira independente. O problema é a manutenção da consistência do banco de dados distribuído entre as diferentes partições.

2.3 – Conclusões

Neste capítulo tivemos uma revisão de Banco de Dados Distribuídos, definimos formalmente confiabilidade, disponibilidade e, sempre que possível, observamos a aplicação dos conceitos estudados a redes peer-to-peer. Vimos como é importante a replicação para gerar a redundância necessária a fim de transpor uma característica inerente das redes peer-to-peer que são as falhas de seus nós, links, etc. Foi mostrado aqui também o problema de manter as réplicas consistentes. No capítulo seguinte veremos mais detalhes sobre as redes peer-to-peer.

Capítulo 3 – Redes Peer-to-Peer

3.1 – Conceitos Básicos

O termo peer-to-peer, amplamente difundido devido à popularidade de softwares como Napster[7] e KaZaA[8], muita vezes é confundido como sistema de troca de arquivos. Segundo ORAM (2001), a Internet concebida no final de 1960 era um sistema peer-to-peer, pois o objetivo final da ARPANET[9] era compartilhar recursos de computação de todo o território americano.

Uma forma de esclarecer em que consiste o paradigma peer-to-peer é através de uma comparação de definições com o a arquitetura cliente-servidor. Vejamos a definição elaborada por KELLERER (1998): *"Uma arquitetura de rede distribuída pode ser chamada Peer-to-Peer (P-to-P, P2P, etc.) se os participantes compartilharem parte de seus próprios recursos de hardware (poder de processamento, capacidade de armazenamento, banda de rede, impressoras, etc.). Esses recursos são necessários para prover os serviços e conteúdo oferecidos pela rede (por exemplo, compartilhamento de arquivos ou espaços de trabalho para colaboração mútua). Os serviços e recursos são acessíveis por todos os pares sem necessidade de passar por nenhuma entidade intermediária. Os participantes dessa rede são tanto provedores de recursos (serviços e conteúdo) como demandadores desses mesmos recursos."*

Para a arquitetura de cliente-servidor, KELLERER (1998) a definiu da seguinte forma: *"Uma rede cliente-servidor possui uma arquitetura distribuída com um sistema de alto desempenho, o servidor, e vários clientes, de menor performace. O servidor é uma unidade central de registro e também o único provedor de serviço e conteúdo. Um*

[7] O site do Napster é http://www.napster.com. Hoje em dia o Napster passou a ser pago.

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

[9] Progenitor da Internet foi a rede ARPANET (Advanced Research Projects Agency Network), proveniente da agência pertencente ao Departamento de Defesa Americano.

cliente somente faz requisições de conteúdo ou execução de serviços ao servidor, sem compartilhar nenhum de seus próprios recursos.".

Portanto uma arquitetura peer-to-peer se refere a um conjunto peers (ou nós) em uma rede de computadores que compartilham recursos entre si através de seus serviços, onde um mesmo peer atua como servidor e como cliente. Neste capítulo veremos algumas classificações para peer-to-peer, alguns exemplos de sistemas com diferentes objetivos e em seguida estudaremos a plataforma JXTA, escolhida como ferramenta para viabilizar este trabalho.

3.1.1 – Classificações

Dentre uma gama de opções de sistemas peer-to-peers existente, é importante classificá-los para seu melhor entendimento. Segundo ROSS, RUBENSTEIN (2004), existem distintas categorias de peer-to-peer, conforme ilustra a Tabela 1.

Tabela 1: Categorias de Sistemas Peer-to-Peer

Categorias	**Exemplos**
Compartilhamento de Arquivos	Napster, Gnutella, KaZaA, eDonkey[10]
Comunicações	Instant Messaging, Voz sobre IP (Skype[11])
Computação	seti@home[12]
Distributed Hash Tables (DHTs) e suas aplicações	Chord (STOICA et al., 2001), CAN (KANGASHARJU et al., 2001), Pastry (ROWSTRON et al. 2001), Tapestry (ZHAO et al., 2001)
Aplicações em overlays emergentes	PlanetLab (PETERSON et al. 2002)

Em ROSS, RUBENSTEIN (2004) são citadas ainda outras classificações para sistemas Peer-to-Peer: centralizados e descentralizados. A idéia de um sistema peer-to-

[10] http://www.edonkey2000.com

[11] http://www.skype.com

[12] Projeto de computação distribuída que pesquisa existência de vida extra-terrestre.

peer centralizado (p.e. Napster) é usar um servidor central para prover a comunicação entre os nós da rede. A comunicação é sempre feita entre o solicitante e o servidor (contém os índices de recursos e peers) para localizar o recurso desejado. Uma vez localizado, é estabelecida a conexão ponto-a-ponto entre o solicitante e o portador do recurso. Esta abordagem implica na falha de toda a rede caso o servidor apresente problemas, o que torna esta abordagem não escalável. Na rede descentralizada, não existe um servidor central: tanto a localização de recursos quanto a transferência dos mesmos é feita ponto-a-ponto.

Quanto sua estrutura, em ROSS, RUBENSTEIN (2004) sistemas peer-to-peer são classificados da seguinte maneira: entende-se por estruturadas as redes descentralizadas que apresentam algum mecanismo que determina a localização obrigatória de um recurso em um dado nó da rede. Portanto, o local onde fica armazenado um recurso é precisamente determinado, como por exemplo, através do uso de Distributed Hash Tables[13], como em RATNASAMY et al. (2001). Existem ainda as redes descentralizadas fracamente estruturadas, que são menos rígidas na localização de um recurso na rede, pois usam uma dica (uma informação não precisa) para encontrar o objeto procurado, tal como ocorre no Freenet[14].

As redes descentralizadas e não estruturadas são aquelas que não possuem servidor central para indexar recursos e nem controle preciso da localização dos mesmos. Podemos citar o Gnutella[15] como exemplo, que usa um mecanismo de busca baseado em *flooding*[16] dos nós vizinhos, o que limita a escalabilidade deste modelo devido ao grande overhead de mensagens trocadas na rede.

---

[13] As DHT ou Tabela de Hash Distribuída possuem um papel importante aplicados a sistemas peer-to-peer no problema de alocação de recursos. A partir do cálculo do hash de um recurso, é determinada sua localização na rede.

[14] http://freenet.sourceforge.net

[15] http://www.gnutella.com/

[16] Técnica de envio de diversas requisições com a mesma consulta para os peers vizinhos. Estes, não sabendo a resposta da consulta, reenviam requisições para seus vizinhos, provocando um efeito bola-de-neve ou flooding da rede.

3.1.2 – Sistemas Peer-to-Peer existentes

3.1.2.1 – Compartilhamento de Arquivos

O compartilhamento de arquivos, em especial o compartilhamento de músicas, é apontado como um dos responsáveis pela disseminação de sistemas Peer-to-Peer. Softwares como o Napster tornaram-se famosos em sua ampla utilização por pessoas do mundo inteiro. A repercussão deste tipo de software Peer-to-Peer tornou-se ainda maior quando a RIAA (Recording Industry Association of America) resolveu abrir processo legal contra o Napster, assim como o fez contra Audiogalaxy, Morpheus, Kazaa, Grokster, Madster e MP3Board. A RIAA sentiu-se ameaçada pelo intercâmbio de músicas que é viabilizado pelos programas, alegando pirataria na justiça americana.

A fim de ilustrar a dimensão da quantidade de sistemas P2P que tem como funcionalidade o compartilhamento de arquivos, podemos citar os seguintes nomes[17]: Acquisition, Aimster (Madster), Ares, Ares Lite, Audiogalaxy, BearShare, BitTorrent, Blubster, Direct Connect, eDonkey, FreeWire, Gnotella, Gnucleus, Grokster, GTK-Gnutella, iMesh, Kazaa Lite, Kazaa Lite K++, Kazaa Media Desktop, LimeWire, LordofSearch, Mactella, Morpheus, Napster, NeoNapster, OneMX, Phex, Piolet, Qtella, Shareaza, SoulSeek, SwapNut, TrustyFiles, Warez P2P, WinMX, XoLoX

3.1.2.2 – Freenet

O projeto Freenet (CLARKE et al., 2004) desenvolveu um software livre peer-to-peer agregando como principal característica o anonimato dos peers. O Freenet se auto-define como um sistema de armazenamento distribuído de informações com o propósito de manter a privacidade alheia. Freenet emprega uma arquitetura totalmente

[17] http://security.uchicago.edu/peer-to-peer/no_fileshare.shtml

descentralizada, empregando parte do disco rígido em desuso de cada peer para formar um sistema de arquivos virtual.

Enquanto que P2P existentes como o Napster e Gnutella provêem o serviço de compartilhamento de arquivos, o Freenet provê o serviço de compartilhamento de armazenamento, isto é, o objetivo do sistema não é trocar recursos em si, mas sim permitir que os integrantes da rede guardem seus dados no conjunto união dos dispositivos de armazenamento existentes. O Freenet implementa um algoritmo probabilístico de adaptação ao padrão de uso de recursos, o que gera replicações e deleções automáticas de arquivos armazenados no sistema de arquivos virtual, de forma a tornar o espaço utilizado mais útil. A arquitetura do software possui dois tipos de chaves: o content-hash key (CHK), usado para armazenamento primário, e o signed-subspace key (SSK), criado para o uso humano, ambos usando hash seguro SHA-1 (EASTLAKE, JONES, 2001). O projeto Freenet realizou simulações com cerca de 200.000 nós e o sistema provou escalabilidade e tolerância à falhas.

3.1.2.3 – EDUTella

O projeto Edutella (NEJDL et al., 2002) consiste em um infra-estrutura open source peer-to-peer tendo como linguagem de descrição dos dados o RDF (BECKETT, MCBRIDE, 2003) e usando o JXTA para utilizar abstrações de rede P2P. O projeto possui uma linguagem própria para consultas distribuídas na rede peer-to-peer Edutella, denominada RDL-QEL (query exchange language), definida em cinco níveis diferentes. O primeiro foco de aplicação da infra-estrutura do Edutella é um ambiente distribuído de anotações educacionais entre universidades alemãs (Hannover, Braunschweig e Karlsuhe), universidades suecas (Stockholm e Uppsala), a universidade de Stanford, entre outras.

Os primeiros serviços projetados para o Edutella foram:

1. Serviço de Consulta: padronização da busca e recuperação de meta-dados escritos em RDF
2. Serviço de Replicação: prover persistência, disponibilidade dos dados, balanceamento de carga, integridade dos dados e consistência dos dados.

3. Serviço de Mapeamento: permitir traduções entre diferentes vocabulários de meta-dados permitindo a interoperabilidade de diferentes peers.
4. Serviço de Anotação: possibilita anotações em qualquer material armazenada na rede edutella.

3.2 – A Plataforma JXTA

3.2.1 – Introdução

O Projeto JXTA é uma iniciativa open-source da Sun Microsystems Inc.[18], originalmente concebido e desenvolvido por um pequeno grupo de especialistas, acadêmicos e corporações, que vem crescendo a cada dia o número de integrantes. Atualmente conta com mais de 100 projetos e mais de um milhão de downloads da tecnologia. Segundo o site do projeto[19], empresas, governos, corporações estão planejando e desenvolvendo soluções baseadas em JXTA.

O JXTA (do inglês *juxtapose* ou justaposição) especifica um conjunto de protocolos que criam uma rede virtual auto-organizável sobre uma topologia de rede real, seja ela qual for, permitindo a comunicação entre diferentes peers. A Figura 4 mostra a idéia da rede virtual criada pelo JXTA:

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

[19] http://www.jxta.org/

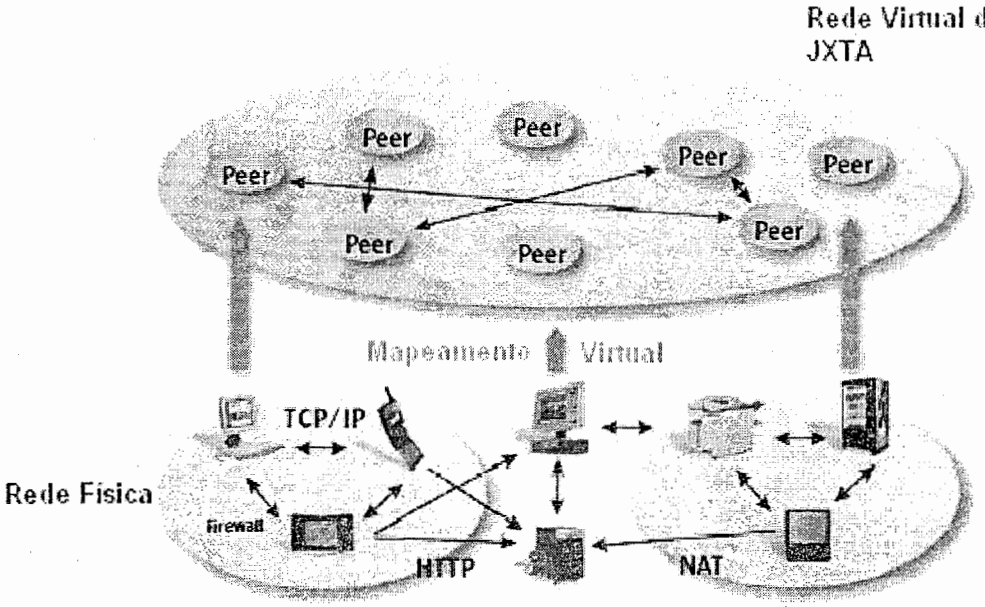


Figura 4: Mapeamento entre a rede física e a rede virtual do JXTA[20]

O JXTA permite que, além dos administradores de rede, desenvolvedores de aplicações possam definir suas topologias de rede, atingindo assim a melhor configuração para cada necessidade específica. Para exemplificar, múltiplas redes *ad hoc*[21] podem ser criadas e dinamicamente mapeadas em apenas uma rede física. O JXTA foi projetado para ser independente de linguagens de programação (atualmente existem implementações em C, Perl, Python e Java) , plataformas (tal como Microsoft Windows e Unix), definições de serviços (RMI[22], WSDL[23]) e protocolos de rede (TCP/IP, Bluetooth). Desta maneira, o JXTA abrange PCs, laptops, PDAs, sensores, eletrônicos, roteadores, servidores de data-centers e sistemas de armazenamento.

---

[20] Figura extraída do site http://www.sun.com/aboutsun/media/presskits/jxta/jxta_slides.html

[21] O termo *ad hoc* vem do Latim e significa "para uma proposta específica". Redes *Ad Hocs* são aquelas que não possuem uma infra-estrutura pré-definida. Elas se organizam conforme a necessidade de resolver uma tarefa específica.

[22] RMI (Remote Method Invocation) é um conjunto de protocolos que permite a objetos Java se comunicarem remotamente.

[23] WSDL (Web Services Description Language) é um protocolo baseado em XML para a troca de informções em um ambiente distribuído.

A versão 1.0 do JXTA foi lançada em Abril de 2001, porém aqui será descrita a versão 2.0 da especificação, que possui diferenças significativas em relação a anterior. Tais diferenças vão desde novos conceitos introduzidos, como *rendezvous peer view* (RPV) a otimizações no controle de threads, filas e tratamento de buffers.

3.2.2 – Conceitos

3.2.2.1 – Arquitetura

A arquitetura do JXTA é composta por três camadas idealmente definidas (veja a Figura 5):

- Camada da Plataforma (*JXTA Core*) – encapsula o mínimo e as primitivas essenciais para o funcionamento da rede P2P.
- Camada de Serviços – é a camada que contém os serviços que são comuns e desejáveis a aplicações P2P.
- Camada de Aplicação – utiliza as demais camadas para criação de aplicações P2P.

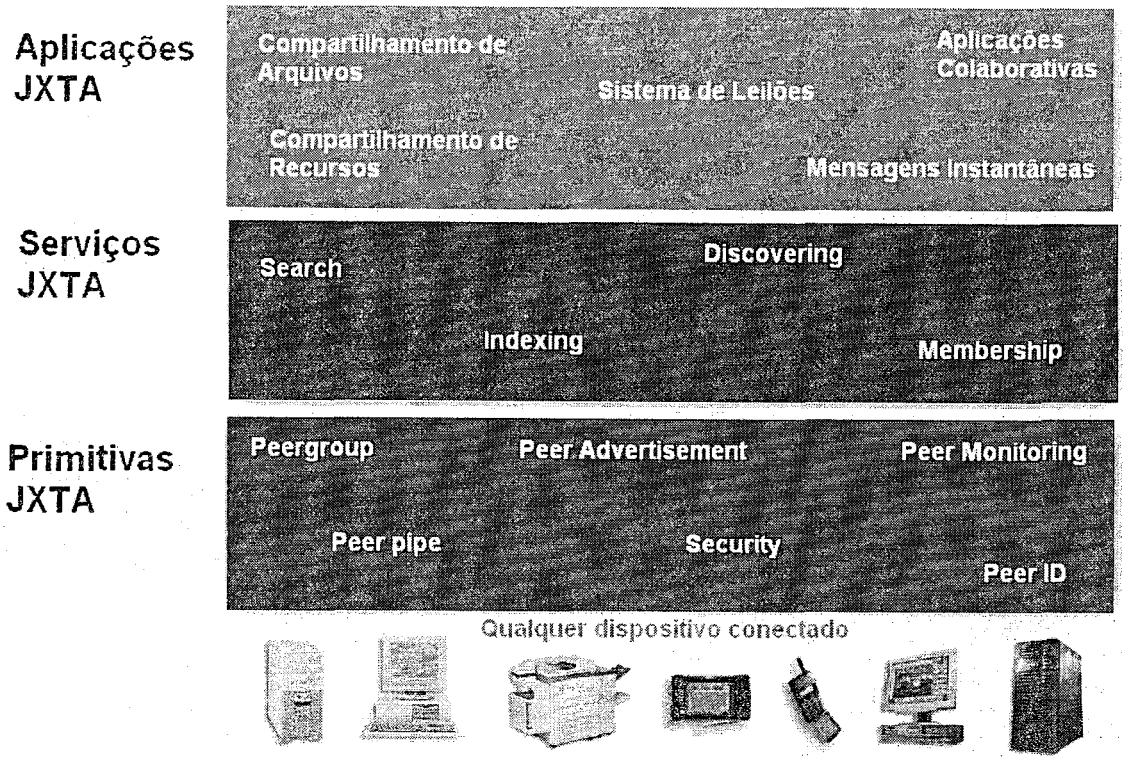


Figura 5: Camadas do JXTA[24]

A fronteira entre a camada de serviços e aplicação não é rígida. Como exemplo, uma aplicação para um cliente pode ser vista como um serviço para outro cliente.

3.2.2.2 – Segurança

Uma rede P2P dinâmica como a rede JXTA necessita suportar diferentes níveis de acesso a recursos. Além disso, cinco requisitos básicos de segurança devem ser providos:

1. Confidencialidade – garantia de que o conteúdo de uma mensagem não será aberto por indivíduos não autorizados.
2. Autenticação – garantia que um remetente é o indivíduo que ele diz ser.
3. Autorização – garantia que um remetente é autorizado a enviar uma mensagem.

[24] Figura adaptada do site http://www.sun.com/aboutsun/media/presskits/jxta/jxta_slides.html

4. Integridade dos Dados – garantia que uma mensagem não foi modificada acidentalmente ou deliberadamente em seu percurso.
5. Refutabilidade – garantia de que a mensagem foi transmitida por um remetente identificado pelo remetente e que não é uma mensagem anterior repetida.

O JXTA conta com um modelo de segurança distribuído, o que evita custos de gerenciamento e implantação de uma infra-estrutura centralizada de Autoridades Certificadoras[25]. O modelo de segurança do JXTA especifica que peers podem ser as Autoridades Certificadoras. Para este fim, fortes algoritmos de criptografia são usados. O Internet Engineering Task Force's (IETF) Transport Layer Security (TLS) (DIERKS, ALLEN, 1999), continuação do SSL versão 3 (FRIER et al., 1996), é implementado como transporte virtual entre peers, onde a cifra default é o RSA (RIVEST et al., 1978).

Além do TLS, faz parte da visão do JXTA ter seus protocolos compatíveis com os mecanismos de segurança na camada de transporte que são amplamente utilizados, tal como o SSL.

3.2.2.3 – JXTA IDs

Cada recurso (peer, peergroup, endpoint, service, content) da rede JXTA possui um identificador único, um número aleatório de 128 bits denominado JXTA ID. Através dessa abstração, um recurso pode ser referenciado independentemente de sua localização física. Por exemplo, um laptop que recebe via DHCP um IP diferente a cada período de tempo, possui o mesmo Peer ID.

[25]Autoridades Certificadoras são instituições responsáveis por emitirem Certificados Digitais que permitem a comunicação segura entre quaisquer entidades.

3.2.2.4 – Peer Endpoint

Um *peer endpoint* é associado a um *peer ID* contendo todos os endereços físicos de rede daquele *peer*. A analogia que é feita para o conceito do *peer endpoint* é o cartão de visitas, que contém todas as formas de se entrar em contato com uma determinada pessoa.

Quando um *peer* recebe um *advertisement* (veremos em seguida) de um *peer endpoint*, é escolhida a forma mais eficiente de se estabelecer comunicação entre ambos, selecionando um dos endereços físicos listados no *peer endpoint* (usando TCP/IP diretamente sempre que possível ou HTTP sobre TCP/IP se existe um firewall).

3.2.2.5 – Advertisements

Todos os recursos da rede JXTA (*peers, peergroups, pipes, services, metering, route, content, rendezvous, peer endpoint, transport*) possuem *advertisements*, que são descritores (meta-dados) dos recursos e são representados por documentos XML.

Interessante salientar que os *advertisements* são extensíveis, o que permite ao desenvolvedor criar seus próprios tipos de metadados. *Advertisements* podem descrever virtualmente qualquer coisa: código-fontes, scripts, binários, classes, J2EE containers, etc. Todos *advertisements* são publicados com um tempo de vida (default é de 15 minutos). As Figuras de 6 a 10 mostram alguns exemplos de *advertisements.*

```
<?xml version="1.0" encoding="UTF-8"?>
<jxta:PeerAdvertisement>
        <Name> name of the peer</Name>
        <Keywords>search keywords </Keywords>
        <Pid> Peer Id </Pid>
        <Services>
                <jxta:ServiceAdvertisement>
                        ...
                </jxta:ServiceAdvertisement>
        </Services>
        <Endpoints>
                <jxta:EndpointAdvertisement>
                        ...
                </jxta:EndpointAdvertisement>
        </Endpoints>
        <InitialApp>
                <jxta:ServiceAdvertisement>
                        ...
                </jxta:ServiceAdvertisement>
        </InitialApp>
</jxta:PeerAdvertisement>
```

Figura 6: Peer Advertisement

```
<?xml version="1.0" encoding="UTF-8"?>
<jxta:ServiceAdvertisement>
        <Name> name of the Service</Name>
        <Version> Version Id </Version>
        <Keywords>search keywords </Keywords>
        <PipeService> Pipe advertisement</PipeService>
        <Params> service configuration params </Params>
                ...
        <Params> service configuration params </Params>
        <Uri> service provider location</URI>
        <Provider> Service Provider</Provider>
        <Security> TBD </Security>
        <Code> class name </Code>
</jxta:ServiceAdvertisement>
```

Figura 7: Service Advertisement

```
<?xml version="1.0" encoding="UTF-8"?>
<jxta:PipeAdvertisement>
        <name> name of the pipe</name>
        <id> Pipe Id </id>
</jxta:PipeAdvertisement>
```

Figura 8: Pipe Advertisement

```
<?xml version="1.0" encoding="UTF-8"?>
<jxta:PeerGroupAdvertisement>
        <Name> name of the peer group</Name>
  <PeerName> name of the peer</PeerName>
        <Keywords>search keywords </Keywords>
  <Pid> Peer Id </Pid>
        <Gid> Peer group Id </Gid>
  <isRendezvous>true or false</isRendezvous>
        <Services>
                <jxta:ServiceAdvertisement>
                        ...
                </jxta:ServiceAdvertisement>
        </Services>
   <Endpoints>
                <jxta:EndpointAdvertisement>
                        ...
                </jxta:EndpointAdvertisement>
        </Endpoints>
        <InitialApp>
                <jxta:ServiceAdvertisement>
          ...
    </jxta:ServiceAdvertisement>
  </InitialApp>
</jxta:PeerGroupAdvertisement>
```

Figura 9: Peer Group Advertisement

```
<?xml version="1.0" encoding="UTF-8"?>
<jxta:EndpointAdvertisement>
        <Name> name of the endpoint</Name>
        <Keywords> search string </Keywords>
        <Address> endpoint logical address </Address>
  <Transport> <jxta:TCPTransportAdvertisement/> <Transport>
  <Transport> <jxta:HTTPTransportAdvertisement/> <Transport>
</jxta:EndpointAdvertisement>
```

Figura 10: Endpoint Advertisement

3.2.2.6 – PeerGroups

Os *peers* da rede JXTA que em geral possui um interesse comum se organizam em *peergroups*, um grupo de peers. A especificação diz como um *peergroup* deve ser criado, anunciado e descoberto. Quanto ao momento criação, fica a critério da aplicação, desenvolvedor e/ou administrador da rede.

Existem três motivações básicas para a existência de *peergroups*:

1. Criar domínios seguros para a troca de conteúdo protegido.
2. Criar um ambiente com escopo delimitado. Algumas mensagens ficam restritas ao grupo.
3. Criar um ambiente de monitoramento, p.e., inspeção de tráfego.

Os grupos do JXTA são criados de maneira hierárquica. Todo *peer* faz parte do grupo *NetPeerGroup*, o pai de todos os grupos (nó raiz). *Peergroups* tipicamente publicam um conjunto de serviços chamado *peergroup services*. Por default, os *peergroups services* de um grupo são: *discovery service*, *resolver*, *pipe*, *peer info* e *rendezvous*. *Peergroup services* são compostos de uma coleção de instâncias (réplicas) de um serviço rodando em múltiplos *peers* do grupo. Desta maneira, se um *peer* falha, o serviço não é afetado. A Figura 11 ilustra a abstração do peergroup.

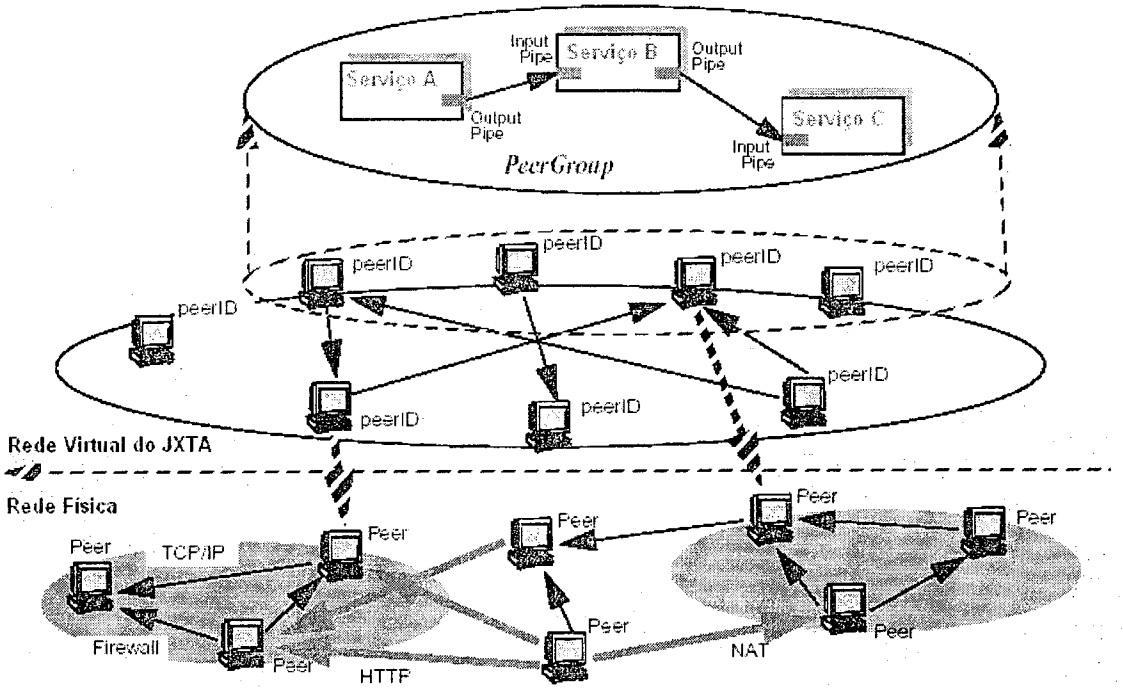


Figura 11: Comunicação entre peers dentro de um Peergroup, adaptada de TRAVERSAT et al. (2003)

3.2.2.7 – Pipes

Os *pipes* possibilitam efetivamente a comunicação entre *peers*. Por exemplo, para enviar uma mensagem, um *peer* A abre um *output pipe* (pipe de envio) para um *peer* B, que possui um *input pipe* (pipe de recebimento). Os *pipes*, assim como demais recursos, possuem *advertisements*, que publicam sua existência.

Existem dois tipos básicos de comunicação por *pipes*:

- *Point-to-point pipe* – conecta exatamente dois *pipes* em um canal unidirecional e assíncrono, sem suporte a *acknowledgments* ou *replies*. O *payload* da mensagem pode conter um *pipe advertisement* para abertura de um novo *pipe* para resposta ao remetente inicial.

- *Propagate pipe* – conecta um *output pipe* a múltiplos inputs *pipes* (interno ao *peergroup*).

Pipes bidirecionais, confiáveis e seguros são *pipes services* implementados em cima dos dois tipos básicos descritos.

3.2.2.8 – Network Services

Peers podem invocar *network services*, que são serviços implementados por um ou mais *peers*. Peers descobrem network services atráves do Peer Discovery Protocol (veremos em 3.2.4.4).

Existem dois tipos de *network services*:

1. Peer Services – acessível apenas em um *peer* que publica o serviço. Se o *peer* sair da rede, o serviço fica indisponível. Múltiplas instâncias do serviço podem rodar em diferentes *peers*, mas cada instância com seu próprio *advertisement*.
2. Peer Group Services – é composto por uma coleção de instâncias (potencialmente cooperando umas com as outras) de um mesmo serviço rodando em múltiplos membros de um *peergroup*. Se um *peer* sair da rede, o serviço continua disponível (assumindo que o serviço está presente em outros *peers* do grupo). Peer Group Services são publicados como parte de um *peer group advertisement*.

Serviços podem ser pré-instalados em um *peer* ou carregados dinamicamente conforme a necessidade. Neste último caso, o peer deve localizar uma implementação compatível com o serviço. Este processo de localização, download e instalação do serviço é similar ao processo de instalação de um plug-in em um web browser.

3.2.2.9 – Modules

Módulos JXTA são abstrações para representar qualquer pedaço de código, seja ele uma classe java, um .jar, uma biblioteca dinâmica DLL, mensagens XML, scripts, etc. É interessante este tipo de abstração para rede JXTA porque permite que um mesmo

network service possua distintas implementações em diferentes plataformas, tais como Windows e Unix.

Módulos provêem uma abstração genérica que permite um *peer* instanciar um novo comportamento, p.e., quando um *peer* entra em um *peergroup*. Neste caso, um serviço de busca específico do grupo deve ser instanciado pelo *peer*. O framework de módulos do JXTA permite a representação e o *advertisement* dos comportamentos de forma independente da plataforma. Desta maneira, o peer poderá instanciar qualquer tipo de implementação de comportamento, seja uma classe Java ou programa em C, por exemplo.

Descrever e publicar de forma independente da plataforma é essencial para suportar múltiplos grupos compostos de *peers* heterogêneos. Para permitir essa variedade de peers, a abstração dos módulos é dividida em três níveis:

1. Module Class – é primariamente usado para anunciar a existência de um comportamento. A definição da classe representa um comportamento esperado e é identificado por um ID único (ModuleClassID).
2. Module Specification – é basicamente usado para acessar módulos. Contém toda informação necessária para acessar e invocar um módulo. Por exemplo, no caso de um serviço, o *module specification* pode conter um *pipe advertisement* a ser usado para a realização da comunicação com o serviço. Um *module specification* é uma abordagem de prover a funcionalidade que aquele *module class* define. Pode haver múltiplos *module specifications* para um mesmo *module class*. Cada *module specification* é identificado por um ID único (ModuleSpecID). Todas implementações de um dado *module specification* precisam usar os mesmos protocolos e são compatíveis, embora possam ser escritas em diferentes linguagens de programação.

3. Module Implementation – é uma implementação de um dado *module specification*. Pode haver múltiplas *module implementations* para um dado *module specification*. Cada *module implementation* contém seu

ModuleSpecID associado, refereciando qual especificação é implementada.

Pode ser citado como exemplo de uso de módulos o JXTA Discovery Service. Este possui apenas um ModuleClassID, identificando-o como um serviço de busca, sua função abstrata. Pode haver múltiplas especificações do *discovery service*, cada uma possivelmente incompatível com outra. Cada uma pode usar diferentes estratégias em relação ao tamanho do *peergroup*, sua dispersão na rede, ou outras abordagens. Cada especificação possui um ModuleSpecID, que referencia o ModuleClassID do *discovery service*. Além disso, para cada especificação, múltiplas implementações podem existir, apontando para um dado ModuleSpecID.

3.2.3 – Arquitetura da Rede

3.2.3.1 – Organização

A rede JXTA é *ad hoc*. Conexões podem ser transientes e o roteamento entre *peers* não é determinístico. Peers entram e saem da rede constantemente e as rotas entre eles mudam com freqüência.

A organização da rede JXTA mostrada a seguir não é obrigatória, mas na prática existem quatro tipos de *peers*:

1. Minimal Edge Peer – envia e recebe mensagens, mas não armazena *advertisements* ou mensagens de rotas de outros *peers*. Geralmente são *peers* de dispositivos com recursos limitados, por exemplo, PDAs e telefones celulares.
2. Full-featured Edge Peer – envia e recebe mensagens. Tipicamente armazena (*cache*) *advertisements*. Este tipo de *peer* responde a consultas (*discovery requests*) com as informações contidas em seu *cache*, mas não encaminha *discovery requests*. A grande maioria dos *peers* em uma rede JXTA é deste tipo.

3. Rendezvous Peer – como outros *peers*, um *rendezvous* armazena *advertisements*. Entretanto, *rendezvous* também encaminha *discovery requests* para auxiliar a descoberta de recursos para outros *peers*. Quando um *peer* entra para um grupo, ele automaticamente procura por um *rendezvous*.
4. Relay Peer – mantém informações sobre rotas para outros *peers* e *route messages* para *peers*. Um *edge peer* tipicamente olha primeiro em seu *cache* para informações sobre rotas. Caso não encontre, o *peer* envia uma consulta para *relay peers*. Além disso, *relays* também encaminham mensagens de *peers* que ficam impossibilitados fisicamente de efetuar uma comunicação direta.

As peculiaridades dos *rendezvous* e dos *relays* serão analisadas adiante. Eles são conhecidos também como os super-peers da rede JXTA por acumularem funções além dos *edge peers*.

3.2.3.2 – Rendezvous Super-Peers

Sistemas distribuídos tradicionalmente possuem algum mecanismo de resolução, onde abstrações são mapeadas em objetos concretos. No JXTA, esse mecanismo é chamado de *resolver*. O *resolver* do JXTA é um protocolo genérico e é utilizado por padrão para resolução, mas nada impede que outras abordagens de resolução sejam implementadas, pois os protocolos do projeto JXTA não especificam como a busca dos *advertisements* deve ser realizada. Assim, desenvolvedores ficam livres para criar suas implementações de resolução, usando abordagens descentralizadas, centralizadas ou híbridas, conforme for a necessidade do domínio da aplicação.

O JXTA tem como política *default* de resolução o uso de *super-peers* denominados Rendezvous. Estes são peers que agregam a função de armazenar apenas índices que referenciam *advertisements*. É fundamental perceber que a infra-estrutura de Rendezvous provê um mecanismo de busca de baixo-nível, assim como deixa o "gancho" para novos serviços de busca de alto-nível.

A criação dos índices que são armazenados nos Rendezvous é feita pelo *Shared Resource Distributed Index* (SRDI) *service* onde cada peer é responsável pelo envio de seus índices, que pode acontecer de forma síncrona com a criação de um *advertisements* ou não (um *daemon* poderia enviar índices em certos períodos). Através do SRDI do JXTA 2.0, três melhorias em relação JXTA 1.0 podem ser citadas: a escalabilidade se torna maior quando um Rendezvous não precisa armazenar *advertisements*; existe a redução do problema de *advertisements* desatualizados; consultas a *advertisements* percorrem apenas Rendezvous enquanto não encontrarem o índice correspondente ao *peer* que contém o alvo da busca, minimizando tráfego na rede.

JXTA propõe o uso de uma Distributed Hash Table (DHT) para mapeamento dos índices distribuídos. Quando um índice se encontra desatualizado, entra em cena o mecanismo denominado *Limited-Range Walker*. Através deste, é percorrido um conjunto de Rendezvous até encontrar o índice buscado, seguindo um subconjunto da lista de Rendezvous vizinhos. Esta lista é ordenada pelo *peerID* de outros Rendezvous no mesmo peerGroup e é denominada *Rendezvous Peer View* (RPV). A fim de se atualizar com rede, Rendezvous periodicamente enviam a outros Rendezvous (selecionados aleatoriamente de suas RPVs) um subconjunto aleatório de Rendezvous conhecidos, assim como também enviam uma mensagem estilo "*I'm alive*" a seus vizinhos mais próximos (o vizinho +1 e o -1 da RPV, onde 0 aponta o próprio Rendezvous).

É interessante ressaltar que pode haver um particionamento da rede, i.e., subconjuntos que a princípio não possuem interseção de *peers*, o mesmo problema que aflinge os Sistemas de Banco de Dados Distribuídos. Entretanto, havendo pelo menos um Rendezvous pertecente as duas partições, o mecanismo de troca de RPVs entre Rendezvous irá realizar a união das partições.

Quanto à conexão de um peer a um Rendezvous, ela acontece da seguinte maneira:

1. *Peer* dispara 5 tentativas de conexões por vez com os Rendezvous referenciados em seus advertisements armazenados
2. Caso não tenha sucesso na conexão (apenas uma conexão é permitida e necessária) em um tempo ajustável (30 segundos por default), o *peer*

dispara uma busca por Rendezvous via *propagate request* ou via *propagate pipe* do *peerGroup* o qual faz parte.

3. Caso não haja uma resposta de nenhum Rendezvous em 30 segundos, o *peer* envia uma consulta aos *seeding rendezvous* - uma lista de rendezvous pré-definidos (qualquer peer de um grupo pode se tornar um)
4. Se os *seeding rendezvous* não responderam em um período (5 minutos por default), o peer irá tentar se tornar um Rendezvous (o *peer* pode voltar a condição de *edge peer*, i.e., não *super-peer*, quando encontrar um Rendezvous) e continuará a busca em background

Os Rendezvous são criados dinamicamente, seguindo a política default do JXTA. Dependendo do cenário empregado, é interessante estruturar a rede, alocando Rendezvous fixos, como o caso de uma organização com servidores 24 por 7[26] ou manter o ambiente *ad hoc*, como por exemplo, um cenário de uma estação de metrô onde PDAs e celulares entram e saem da rede constantemente. Neste último caso, Rendezvous tendem a ser menos estáveis.

3.2.3.3 – Relay Super-Peers

Os *relays* existem para permitirem a comunicação de forma transparente entre *peers* quando a realidade física não a permitiria, tal quando existem NAT e/ou Firewalls. Assim, um *peer* tentará 'falar' com outro *peer* diretamente. Caso não seja possível, tentará via um (ou mais) *relay(s)*. E talvez em uma outra mensagem, por outro(s) *relay(s)*. Isto demonstra a natureza não-determinística das rotas, que podem mudar rapidamente. Assim como os Rendezvous, *relays* mantém uma lista de *relays* conhecidos e as enviam de maneira análoga. Existem também os *seeding relays*, análogos aos *seeding rendezvous*.

[26] Termo empregado para expressar que um sistema fica ligado todo tempo, ou seja, 24 horas por dia e 7 dias por semana.

As rotas, assim como os demais recursos (*peers*, grupos, etc.), possuem *advertisements*. As rotas são construídas a partir do *peer* que envia uma mensagem. Os *edge peers* armazenam *route advertisements*, que descreve como chegar a um determinado *peer* seguindo uma seqüência de *hops* (pulos), tipicamente *relays*. Cada mensagem enviada armazena informações sobre a rota. O mecanismo de roteamento é chamado então de *Adaptive Source-based Routing*.

Cada mensagem enviada possui uma *credential*, que quando presente no corpo da mensagem, é usado para identificar o remetente e seus direitos de envio daquela mensagem. Cada implementação de uma *credential* é específica, possibilitando a coexistência de múltiplas técnicas de autenticação na mesma rede. Mensagens também podem ser criptografadas e assinadas para garantir irrefutabilidade, confidencialidade e integridade. Neste sentido, protocolos de transporte seguros como Transport Layer Security (TLS), Internet Protocol Security (KENT, ATKINSON, 1998) e Secure Sockets Layer (SSL) são amplamente aceitos no JXTA.

3.2.4 – Protocolos

O JXTA é composto por seis protocolos, divididos em duas categorias: *core specification protocols* e *standard services protocols*. O mínimo necessário para uma implementação poder ser considerada complacente com a especificação do JXTA é implementar o *core specification protocols*. A Figura 12 mostra os protocolos do JXTA.

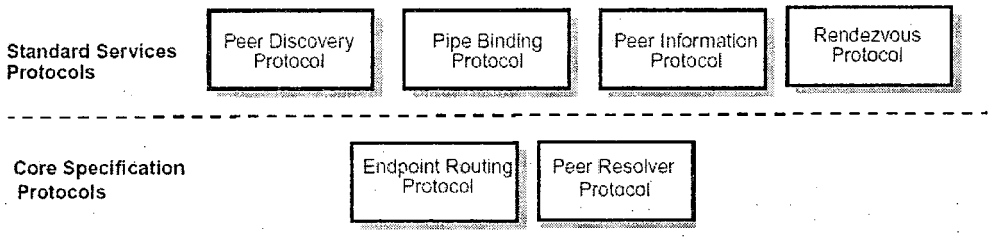


Figura 12: Protocolos do JXTA, extraída de TRAVERSAT et al. (2003)

O *core specification protocols* define dois protocolos:

1. Endpoint Routing Protocol (ERP) – protocolo o qual um peer deve usar para encontra uma rota para outro peer.
2. Peer Resolver Protocol (PRP) – através dele, um peer lança um consulta genérica para um ou mais peers, e recebe uma ou mais respostas a consulta.

O *standard service protocols* define quatro protocolos:

1. Rendezvous Protocol (RVP) – protocolo usado por um peer para se inscrever/desinscrever no serviço de propagação de mensagens.
2. Peer Discovery Protocol (PDP) – o protocolo pelo qual um peer publica seus advertisements e descobre advertisements de outros peers.
3. Peer Information Protocol (PIP) – através dele, um peer pode obter informações sobre o estado de outros peers.
4. Pipe Binding Protocol (PBP) – usado por peers para atar (bind) dois ou mais pipes de uma conexão mapeados em um peer endpoint físico.

3.2.4.1 – Endpoint Routing Protocol

O Endpoint Routing Protocol define um conjunto de mensagens de busca usadas para encontrar informações sobre rotas. Este tipo de informação é necessário para o envio de uma mensagem de um *peer* a outro. Quando um *peer* é precisa enviar uma mensagem a um *peer endpoint address*, o *peer* primeiramente checa seu *cache* para verificar se possui a rota desejada. Caso não a encontre, ele enviará uma *route resolver query* para seus *relays*. Quando um *relay* rececbe uma consulta deste tipo, checa se possui a rota. Caso afirmativo, o peer retorna a informação sobre rota na forma de uma enumeração de *hops*.

Informações sobre rotas incluem *peer ID* do remetente, *peer ID* do destinatário, um tempo de expiração ou *time-to-live* (TTL) para a rota e uma seqüência ordenada de

peers IDs que serviram de *gateway*. Esta última pode não estar completa, mas deve conter pelo menos o primeiro *relay*.

3.2.4.2 – Peer Resolver Protocol

O Peer Resolver Protocol (PRP) permite que *peers* enviem uma consulta genérica a outros *peers*. Consultas podem ser enviadas a um *peer* específico, ou podem ser propagadas se enviadas via um *rendezvous service* dentro de um *peergroup*.

O PRP serve de base para outros protocolos do JXTA, tais como o PIP (usado para extrair informações sobre um *peer*) e o PDP (descobrir recursos de um *peer*). O PRP pode ser usado pode ser usado, por ser genérico, por qualquer aplicação desenvolvida para JXTA.

Uma *resolver query message* é usada para enviar um consulta a um serviço de um outro membro do grupo. A *resolver query message* contém a credencial do remetente, um único ID para a consulta, um *service handler* específico (quem irá tratar a mensagem) e a consulta propriamente dita. Cada serviço pode registrar um *handler* no *peer group resolver service* para processar as consultas e gerar respostas. Para as respostas, existe resolver *response message* que contém a credencial do remetente, um ID único para a consulta, um *service handler* específico e a resposta. Um peer pode receber zero ou mais respostas para uma consulta.

3.2.4.3 – Rendezvous Protocol

O Rendezvous Protocol (RVP) é responsável por propagar mensagens dentro de um *peergroup*. Enquanto diferentes *peergroups* podem ter diferentes maneiras de propagar suas mensagens, o RVP é um protocolo simples que permite:

- Peers conectem a serviços (através das consultas e respostas propagadas)
- Controle de propagação de mensagens (detecção de *loopback*, TTL, etc.)

O RVP é usado por dois outros protocolos: o Peer Resolver Protocol (PRP) e pelo Pipe Binding Protocol (PBP).

3.2.4.4 – Peer Discovery Protocol

O Peer Discovery Protocol (PDP) é usado para descobrir e publicar recursos, onde recursos podem ser *peers*, *peergroups*, *pipes*, *services*, etc. Cada recurso possui um *advertisement* associado.

O PDP permite que um peer encontre *advertisements* em outros peers. O PDP é o protocolo default de busca para todos os usuários de um dado grupo, assim como do *peergroup* default *NetPeerGroup*. Serviços de buscas podem ser usados para melhorar o PDP. Se um *peergroup* não possui seu próprio *discovery service*, então o PDP é usado.

3.2.4.5 – Peer Information Protocol

O Peer Information Protocol (PIP) extrai informações sobre o estado de um dado *peer*. Esta informação pode ser usada para *deployment* comercial ou interno. Como exemplo, em *deployments* comerciais, as informações podem ser usadas para determinar o uso de um *peer service* e geração da conta para os consumidores deste serviço. Exemplo de *deployment* interno: um departamento monitorando um peer para melhorar o roteamento e a desempenho da rede como um todo.

Além disso, o PIP possui a mensagem de ping para checar se um *peer* está online. A mensagem de ping especifica que tipo de resposta deve ser retornada: um simples *acknowledge* ou resposta completa (*advertisement* do *peer* requisitado).

3.2.4.6 – Pipe Binding Protocol

O Pipe Binding Protocol é usado por membros de um *peergroup* para atar (bind) um *pipe advertisement* a um *pipe endpoint*. Assim é criado o link na rede virtual do JXTA que é mapeado na rede física pela camada de transporte, tipicamente TCP/IP.

A mensagem de consulta PBP é enviada por um *peer pipe endpoint* para encontrar um *pipe endpoint* já atado ao mesmo *pipe advertisement* desejado. A mensagem de consulta pode conter um opcional *peer ID*, significando que apenas o dado *peer* será respondido.

A mensagem de resposta do PBP é enviada ao peer que enviou a consulta por cada *peer* com um *pipe* atado. Esta mensagem contém o *pipe ID*, o *peer* onde um *input pipe* foi criado e um valor booleano indicando se o *input pipe* existe no *peer* especificado.

3.2.5 – Implementação do JXTA

Por ser tratar um projeto criado pela Sun Microsystems, é natural que Java seja pioneiro na implementação do JXTA, apesar de, como já foi mencionado, existem implementações em outras linguagens de programação.

Para visualização da arquitetura implementada do JXTA, a Figura 13 apresenta uma visão geral do JXTA 2.0 implementado em J2SE (Java 2 Standard Edition).

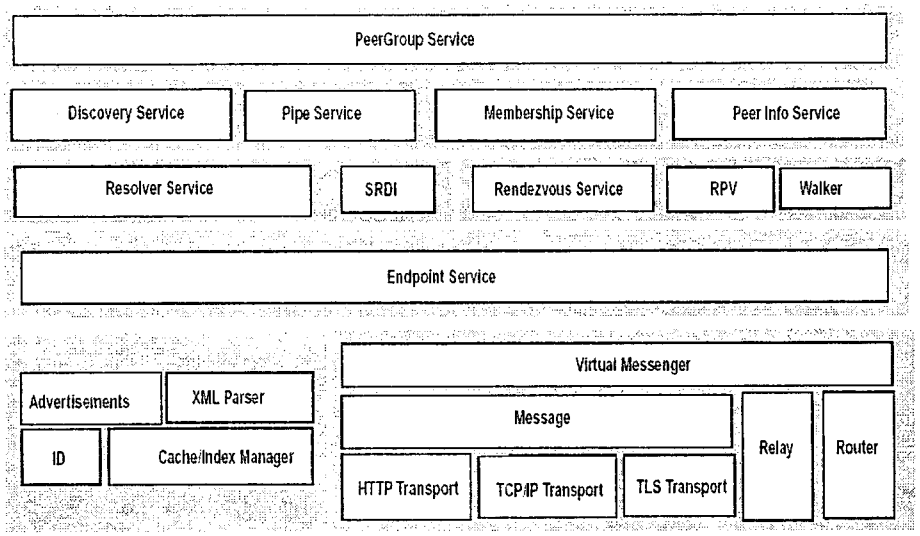


Figura 13: Arquitetura do JXTA implementada em Java

3.2.6 – Projetos usando JXTA

O JXTA, desde seu lançamento em 2000, vem atraindo a atenção de muitos grupos de pesquisa e o mercado em geral. Vejamos alguns projetos desenvolvidos usando a tecnologia.

3.2.6.1 – AIsland's

O Projeto AIsland's[27] consiste em um framework para construir e distribuir agentes, onde cada peer consiste em um AIsland, onde agentes móveis viajam pelas ilhas (ou peers). Se um agente requisitar um módulo desconhecido em determinada ilha, a plataforma solicita a permissão do usuário para efetuar o carregamento do código necessário proveniente de uma outra ilha.

O projeto inclui um editor de agentes (que são escritos em JavaScript), certificação dos módulos por assinatura digital, carregamento automático de módulos, interface gráfico em Swing (mostrada na Figura 14), entre outros.

[27] http://aisland.jxta.org/

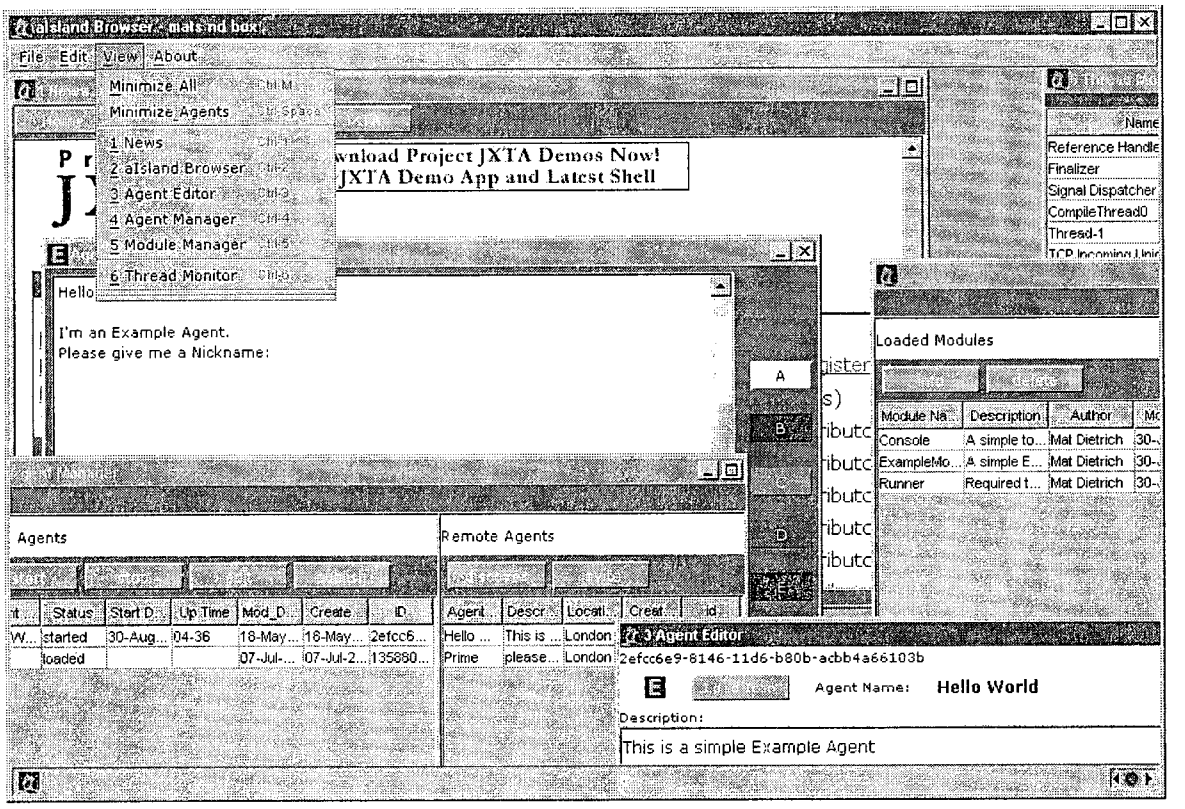


Figura 14: GUI do AISland

3.2.6.2 – MyJXTA2

O MyJXTA[28] versão Enterprise 2 é uma aplicação que foi projetada para ser fácil de usar, estender e implantar. As suas principais características dessa ferramenta são:

- Chat em grupo
- Chat privado seguro
- Facilidades para Criação de grupos e credenciais
- Compartilhamento de Conteúdo (proveniente do projeto CMS como veremos adiante)
- Busca por recursos na rede JXTA
- Text-to-Speech (FreeTTS)

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

Por ser um dos mais completos projetos desenvolvidos em JXTA pela licença da Sun, o MyJXTA2 serviu de base para implementação do COPPEER (veremos em 4.1). A Figura 15 mostra a interface gráfica do MyJXTA2.

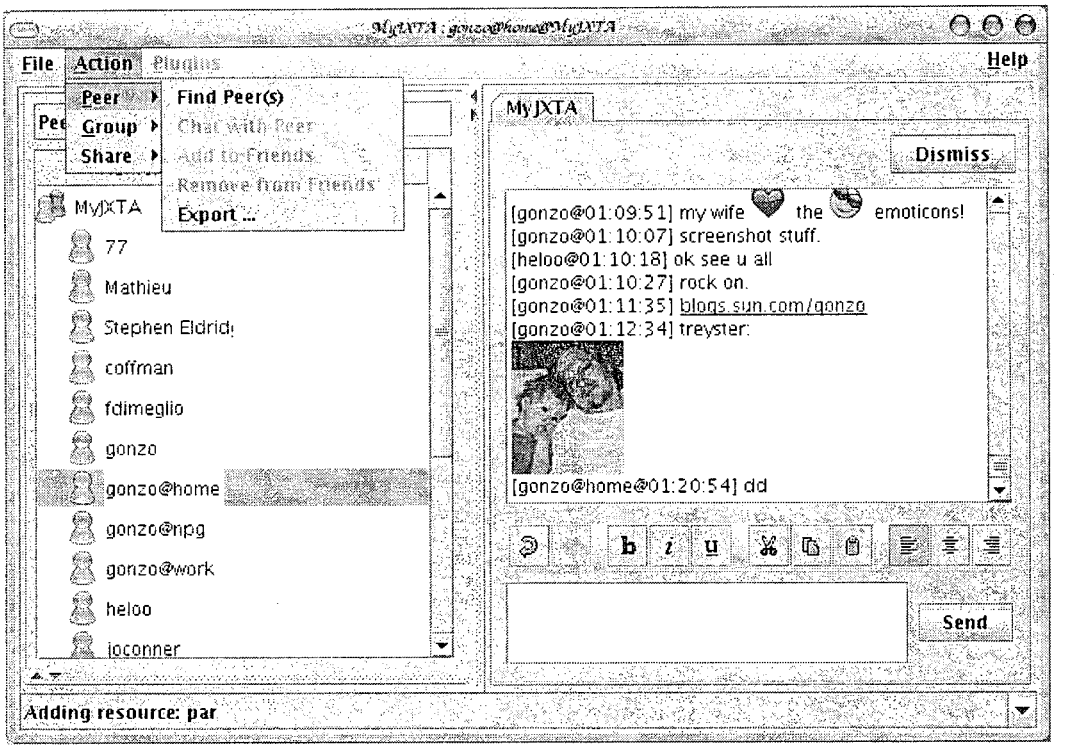


Figura 15: MyJXTA2 e sua interface gráfica, mostrando um chat entre peers

3.2.6.3 – CMS

O Content Manager Service[29] (CMS) é um serviço baseado em JXTA que permite o compartilhamento de arquivos dentro de um grupo de peers. Cada arquivo compartilhado possui um ID único, gerado a partir do conteúdo do arquivo usando um hash MD5 (RIVEST, 1992). Todo arquivo também possui um *advertisement* que contém meta-dados como nome, descrição, tamanho e mime-type. O controle de transferência dos arquivos é feito através de pipes. A seguir um exemplo de *advertisement* do CMS.

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

```
<?xml version="1.0">
    <!doctype jxta:contentAdvertisement>
    <jxta:contentAdvertisement>
      <name>index.html</name>
      <cid>md5:1a8baf7ab82c8fee8fe2a2d9e7ecb7a83</cid>
      <type>text/html</type>
      <length>23983</length>
      <description>Web site index</description>
    </jxta:contentAdvertisement>
```

Figura 16: Exemplo de Advertisement do CMS

3.3 – Conclusões

Este capítulo mostrou o estado da arte dos sistemas peer-to-peer e focou especialmente em JXTA.

Ao longo do capítulo, percebemos também que problemas como alocação de arquivos, integridade dos dados, replicação e segurança permanecem comuns a sistemas peer-to-peer e banco de dados distribuídos.

Alguns dos projetos aqui ilustrados mostraram a versatilidade de aplicações JXTA. Um deles, o projeto MyJXTA2, foi utilizado como base para o desenvolvimento da plataforma COPPEER, como veremos no capítulo seguinte. Também veremos como a Garantia de Disponibilidade se introduz neste contexto e como o COPPEER atua para resolver os desafios comuns de banco de dados distribuídos e sistemas peer-to-peers.

Capítulo 4 – Garantia de Disponibilidade

Neste capítulo veremos como a garantia de disponibilidade pode auxiliar um sistema peer-to-peer a oferecer uma premissa dos bancos de dados: a completude da consulta. Veremos como funciona o COPPEER (BRAGA et al., 2004), framework peer-to-peer da COPPE e em seguida é descrito o modelo implementado para garantia de disponibilidade.

4.1 – COPPEER

4.1.1 – O Conceito

Como vimos nos capítulos anteriores, existe uma série de sistemas peer-to-peer que apresentam funcionalidades interessantes, que vão desde a voz sobre IP (Skype[30]) até troca de mensagens em tempo-real (MSN Messenger[31]). Dentro deste mar de opções, um grande empecilho é que não existe interoperabilidade entre a maioria deles. No caso dos messengers, existem até alguns aplicativos que focam em tornar interoperáveis demais sistemas peer-to-peer, como é o caso do Trillian[32].

Programar aplicações diretamente em JXTA pode ser uma tarefa complexa, dependendo da experiência do desenvolvedor com a tecnologia. Existem algumas abordagens que tentam minimizar este impacto que é programar em JXTA, tal como o EZEL/JAL[33], que abstraem alguns detalhes de implementação e tornam mais fácil o desenvolvimento.

---

[30] http://www.skype.com

[31] http://messenger.msn.com/

[32] http://www.trillian-messenger.de/

[33] JAL (JXTA Abstract Layer) e EZEL (Easy Entry Library for JXTA) são APIs que buscam facilitar o desenvolvimento de aplicações peer-to-peer usando JXTA.

Além da interoperabilidade e facilidade de desenvolvimento, há entre os distintos sistemas peer-to-peer características comuns, como por exemplo, a troca de mensagens entre os peers, implementado pelo JXTA. Se nos aprofundarmos e pensarmos em uma aplicação colaborativa tal como o COE (XEXÉO et al., 2004), perceber a presença dos demais peers da rede é uma característica importante, assim como a consistência dos dados e a completude da busca por ontologias. Essas características podem ser pensadas como serviços comuns a aplicações diversas que fazem uso da comunicação ponto-a-ponto.

Para preencher este nicho foi projetado o COPPEER (BRAGA et al., 2004), um framework de aplicações peer-to-peer. Uma aplicação pode ser plugada ao container estruturado com um microkernel (veremos em 4.1.2) que por sua vez oferece as facilidades de:

- Uso dos serviços do COPPEER (Controle de Transação (DE SOUZA et al., 2004), Garantia de Disponibilidade, Memória Compartilhada, etc.)
- Maior nível de abstração para o desenvolvimento das aplicações em JXTA
- Interface Gráfica Padronizada para as Aplicações
- Suportar aplicações colaborativas

O COPPEER é um framework que facilita o desenvolvimento de aplicações colaborativas peer-to-peer. A Figura 17 mostra a GUI do COPPEER executando o COE, a aplicação de edição colaborativa de ontologias.

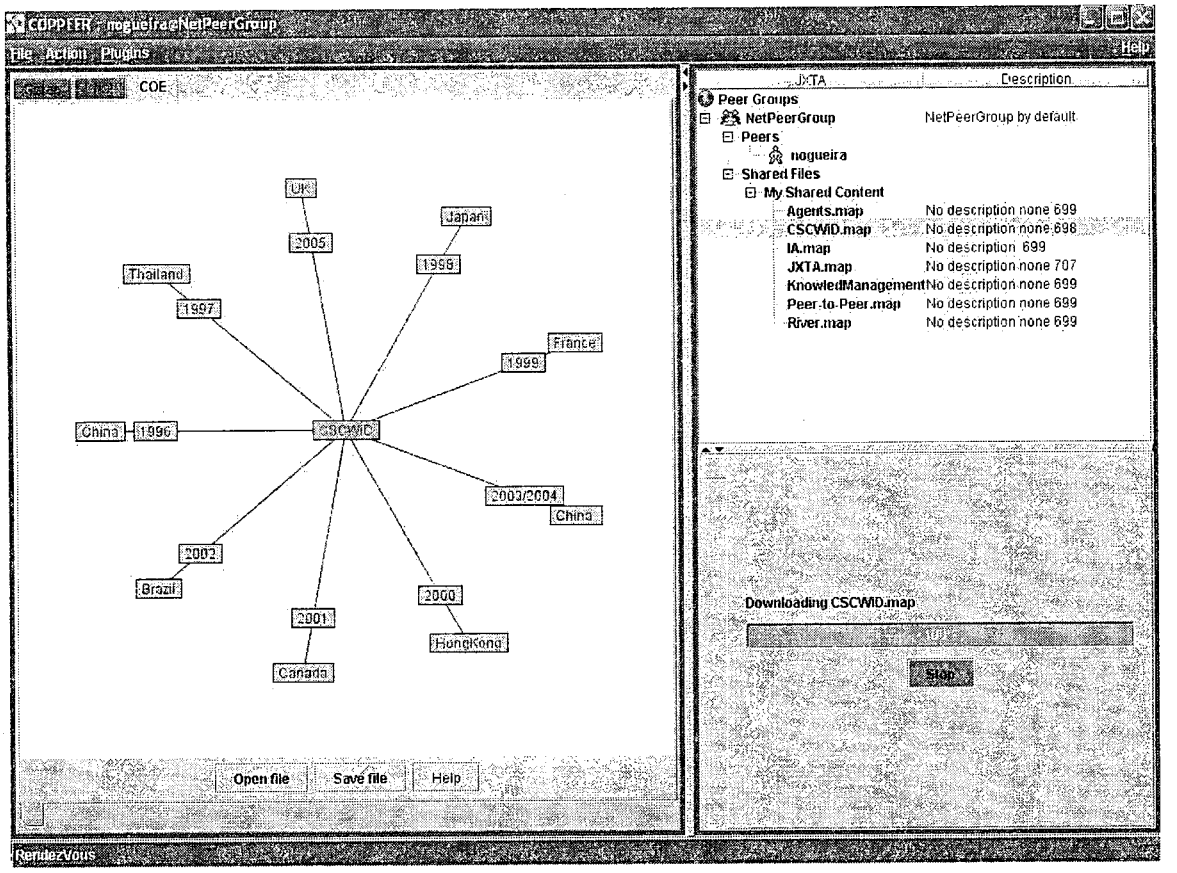


Figura 17: COE – editando e trocando ontologias entre peers pelo COPPEER

4.1.2 – Microkernel COPPEER

O microkernel é a estrutura responsável por agregar novas aplicações peer-to-peer ao COPPEER, de forma a agrupar aplicações desacopladas em uma estrutura coesa. Através da utilização das interfaces do COPPEER, uma aplicação pode ser desenvolvida e incorporada ao container.

É importante notar que a aplicação a ser desenvolvida e adicionada ao container não invoca diretamente métodos do COPPEER. A aplicação, além de implementar as interfaces, necessita de um arquivo de configuração em XML para ser implantada (deploy) no container, um funcionamento similar ao de aplicações EJB (THOMAS, 1998).

Do ponto de vista do microkernel, aplicações são componentes que devem ser gerenciados. O COPPEER utiliza o framework Spring[34], que implementa a estrutura de microkernel necessária usando o padrão Inversão de Controle (IoC) ou mais precisamente *Dependency Injection*[35]. O *Dependency Injection* é nome genérico dado a três outros sub-padrões: *Interface Injection* (1 *IoC*), *Setter Injection* (2 *IoC*) e *Constructor Injection* (3 *IoC*). No COPPEER, é utilizado o *Setter Injection*, apesar do framework Spring suportar 2 *IoC* e 3 *IoC*.

O padrão Inversão do Controle é que vai promover o suporte ao desenvolvedor da aplicação abstrair chamadas a componentes. A aplicação não invocará classes concretas e sim interfaces, onde o microkernel irá assinalar a interface à implementação do componente.

Atualmente existem outros containers que implementam o Dependency Injection, tais como: PicoContainer[36], que possui Setter Injection e Constructor Injection; Avalon[37], um dos containers mais completos.

Além do *Dependency Injection*, outro pattern, denominado *Service Locator,* é também considerado Inversão de Controle. Dependendo do tipo de aplicação, deve ser ponderado o uso do *Service Locator* ao invés do *Dependency Injection.*

---

[34] http://www.springframework.org/documentation.html

[35] http://www.martinfowler.com/articles/injection.html

[36] http://www.picocontainer.org/

[37] http://avalon.apache.org/. Atualmente o Avalon foi descontinuado.

4.1.3 – Serviços do COPPEER

Um serviço do COPPEER é uma aplicação que possui como finalidade auxiliar a execução de uma outra aplicação. Esta comunicação é feita por intermédio do microkernel.

O COPPEER possui atualmente dois serviços: o controle de transação e a garantia de disponibilidade. O controle de transação pode ser sintetizado em manter a consistência dos dados das aplicações peer-to-peer. Já a garantia de disponibilidade se resume a prover a uma estabilidade dos dados que não é própria de um ambiente peer-to-peer. Este tema será abordado em detalhes a seguir.

4.2 – Serviço de Garantia de Disponibilidade

Vimos que os Bancos de Dados Distribuídos ampliam as funcionalidades dos bancos de dados centralizados ao mesmo tempo em que trazem novos desafios. Um dos problemas estudados em SBDD é a alocação de arquivos, uma vez que é precisa definir quais dados estão disponibilizados em determinados locais da rede. A alocação ótima é um problema NP - difícil, sendo viabilizado através do uso de heurísticas.

Na arquitetura peer-to-peer JXTA do COPPEER, seus objetos permanecem *online* enquanto o peer que os contém está na rede. Quando o peer se desconecta, por algum motivo como por exemplo, a queda da conexão do link ou o simples desligar da máquina por parte do usuário, seus objetos deixam de estar disponíveis para os demais peers da rede.

Através do mecanismo de garantia de disponibilidade, fica a critério do usuário ou da aplicação colaborativa desenvolvida para o COPPEER (plugin) decidir se um determinado objeto deve continuar *online* ou não, independentemente do estado do peer. Cada objeto é tratado de uma maneira diferente, i.e., possui um parâmetro que

determina seu grau de disponibilidade (varia de zero a um). Por exemplo, uma aplicação como o COE (Editor Colaborativo de Ontologias) pode determinar que certo conjunto de ontologias básicas deva estar altamente disponível, ou seja, possui um grau de disponibilidade próximo a 1 (100% disponível), enquanto que demais ontologias derivadas possam ser menos importantes e, portanto é aceitável que sejam definidas a priori como menos disponíveis.

Grandes clusters de servidores, dentro da arquitetura cliente/servidor, se encarregam de manter sites online a maior parte do tempo. Para mantê-los altamente disponíveis, é desenvolvido um projeto que quantifica, por exemplo, o número de máquinas, HDs, RAID[38], múltiplos links, enfim, toda a redundância envolvida para este fim.

A arquitetura peer-to-peer permite envolver uma enorme quantidade de máquinas para fins distintos. Dentro de um grupo COPPEER onde haja o interesse de manter disponíveis os objetos isto é feito através da ativação do serviço de garantia de disponibilidade do mesmo. Apesar de não haver necessariamente grandes clusters de servidores 24 por 7 mantendo os objetos online, a quantidade de máquinas presente no grupo da rede peer-to-peer e a redundância gerada pelo serviço de garantia de disponibilidade tendem a surtir o mesmo efeito de uma maneira ad hoc.

4.3 – Modelo de Replicação

Em sistemas distribuídos a replicação é um reconhecido mecanismo de aumento da disponibilidade. Através dela, é atingida a redundância para que, no caso de falha de um dos componentes do sistema, haja outro que possa suprir a necessidade momentânea, i.e., é provido um mecanismo de tolerância à falhas.

Em BORGHOFF, SCHLICHTER (2000) é citado que a principal razão para adoção de um modelo distribuído e replicado é alcançar uma maior disponibilidade dos

[38] Redundant Array of Independent (ou Inexpensive) Disks – conjunto de HDs que promovem mais disponibilidade através de suas redundâncias.

dados do groupware. Vale lembrar que a replicação pode ser acionada para outros fins, como o aumento do desempenho da busca, como em COHEN, SHENKER (2002). Nosso objetivo é replicar para aumentar a disponibilidade. A seguir foram elaboradas questões para descrever nosso mecanismo de replicação.

4.3.1 – O que replicar

Alguns sistemas peer-to-peer como e-Donkey[39] e BitTorrent[40] particionam seus arquivos em pedaços menores e os compartilha. Esta técnica de fragmentação, segundo KUBIATOWICZ et al. (2000), aumenta a confiabilidade das réplicas.

Em GRIBBLE et al. (2001), é definido o conceito de Granularidade de Dados como uma dimensão do problema de alocação de arquivos. Desta forma, para atingir uma alocação ótima, conjuntos de arquivos podem ser formados ou, de maneira contrária, arquivos podem ser quebrados em pedaços, gerando uma hierarquia de granularidade. Esta hierarquia descreve como os objetos se inter-relacionam, i.e., qual parte forma o todo.

No nosso modelo de replicação, assim como a aplicação colaborativa define qual o grau de disponibilidade é desejado para um determinado objeto, a Granularidade de Dados fica a critério da mesma. Também fica a critério da aplicação a associação dos objetos replicados, pois o serviço não prevê um controle sobre relacionamentos. Desta forma a aplicação ganha flexibilidade na escolha do que replicar. Grandes objetos podem ser divididos em pequenas partes de tamanho fixo e tratadas como objetos independentes. Vejamos um exemplo: suponha que um objeto X de 500 MBytes com grau de disponibilidade (threshold) igual a 99% seja particionado pela aplicação em objetos de 50 MBytes. A aplicação ficará encarregada de definir para cada uma das 10 partes o threshold de 99% e montar uma estrutura de meta-dados (uma ontologia, por exemplo) que define as ligações entre fragmentos de objetos. A replicação das partes fica a cargo do serviço de disponibilidade, inclusive da estrutura que define a ligação

[39] http://www.edonkey2000.com/

[40] http://bittorrent.com/documentation.html/

entre os objetos. Quando um usuário da aplicação em outro peer desejar acessar o objeto X, este deverá procurar primeiro pela estrutura que define a ligação entre os fragmentos (meta-dados) e posteriormente terá as informações para procurar pelas partes. Este processo de busca pode ser automatizado por um serviço de busca agregado ao COPPEER.

4.3.2 – Quem inicia o processo de replicação

O processo de replicação deve ser iniciado de alguma forma dentro da rede de peers. O processo pode ser definido de duas formas: autônomo (cooperação assíncrona) ou coordenado (cooperação síncrona).

Um processo de replicação autônomo viabiliza a geração de réplicas pelos peers de uma maneira independente, i.e., um peer da rede que sinta a necessidade de criação de novas réplicas o fará sem trocar informações ou delegar atos aos demais peers. Esta forma autônoma possui vantagens como:

1. Rapidez no processo decisório de replicação, pois o consenso e eleições serão descartados;
2. Menor volume de mensagens trafegadas na rede.

Apesar de sua independência e isolamento, trata-se de uma forma de cooperação – pois um peer que aciona a replicação está o fazendo para aumentar a disponibilidade de um ou mais objetos para o grupo – e é assíncrona, pois não envolve a sincronização dos peers no processo. A desvantagem desta abordagem é que, como os peers não se comunicam para relatar suas ações, o trabalho pode ser replicado também, ou seja, uma replicação de um objeto X pode ser realizada por um ou mais peers simultaneamente, o que irá gerar um dispêndio maior de banda e espaço de armazenamento (excesso de réplicas de X na rede). Este esforço poderia ter sido evitado com a comunicação entre os peers.

O processo coordenado envolve a comunicação nas decisões de replicar. Esta abordagem evita esforço repetido por replicações simultâneas como acontece no caso da replicação autônoma.

Uma forma de realizar a coordenação é através da divisão de papéis entre os peers: o peer coordenador, aqui chamado de Master, e os demais peers que serão os receptores de réplicas.

A existência do Master evita replicação excessiva (no caso de peers replicando simultaneamente no processo assíncrono) e consequentemente diminui custos de armazenamento, banda e monitoramento, que veremos adiante.

O peer Master é responsável pela geração e monitoramento das réplicas. Cada objeto é controlado por um Master, que checa periodicamente seus peers coordenados. A Figura 18 mostra dois objetos coordenados por diferentes Masters em um mesmo grupo, onde a linha representa que um Master (M) checa se o peer coordenado (C) e sua réplica estão ainda online.

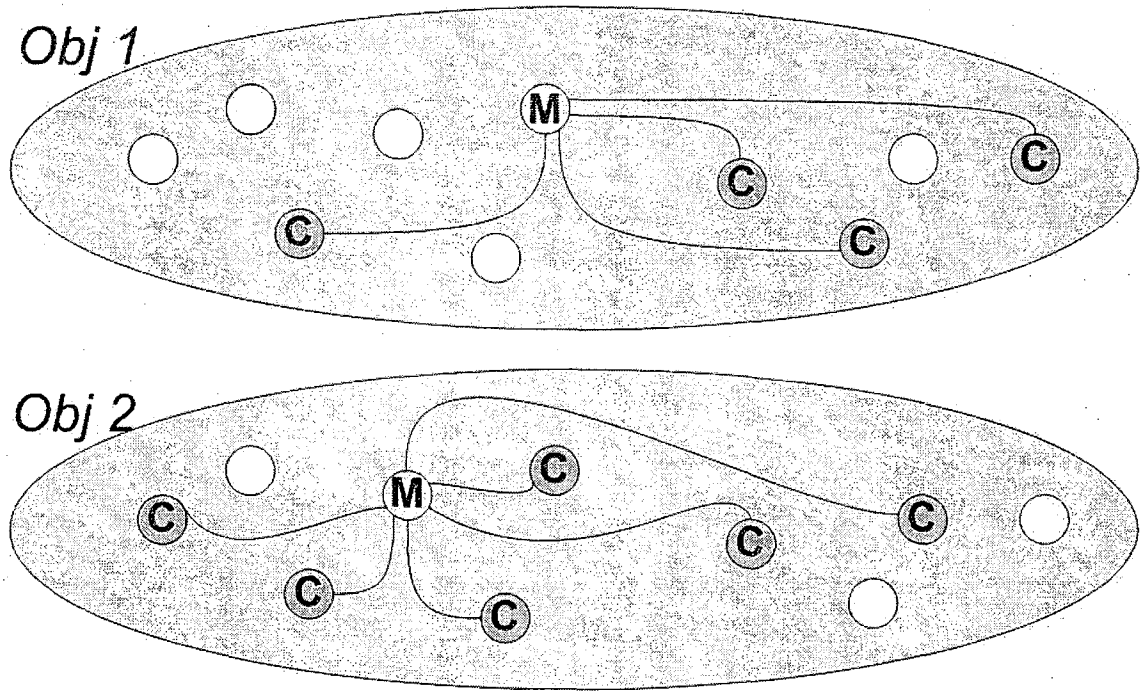


Figura 18: Diferentes camadas de coordenação envolvendo o mesmo grupo de peers

Para se chegar a um nó Master, é realizada uma eleição por algum peer (ou o peer criador do objeto ou algum peer que já contém aquele objeto e detectou que não existe um nó Master) segundo algum critério, que no nosso caso está baseado seguintes parâmetros normalizados:

1. Disponibilidade Absoluta (*A*) – mede a razão entre o tempo que o peer fica online sobre o tempo de vida do peer – desde a primeira entrada do peer na rede. Quanto mais tempo um peer permanece online, maiores as chances de ele ser eleito nó Master. Como estamos trabalhando no nível de abstração de um grupo JXTA, percebemos a rede por este ângulo. Portanto se um link físico cai, determinados peers dependentes deste link cairão também, excluindo-os do grupo e deixando de contabilizar tempo online. O tempo online é um parâmetro relativo e portanto estaremos relacionando-o com uma máquina *seeding rendezvous* do JXTA, i.e., se um peer for capaz de alcaçá-la, estará online. Assim, fatores externos como a queda de um link ou o congestionamento de uma rota podem ser percebidos por esta métrica de disponibilidade.

2. Número de Coordenações (*M*) – mede a razão entre o número de objetos coordenados pelo peer sobre o número total de objetos armazenados (coordenados e não-coordenados por este). Se um nó já possui muitas responsabilidades de coordenação, ele tende a ser menos cotado para o papel de Master. Este critério é fundamental para evitar a concentração de coordenação em alguns poucos nós. Para exemplificar, quando existem super-peers na rede, i.e., nós que contêm alta disponibilidade, grande capacidade de armazenamento e largura de banda suficiente para suas necessidades, tenderiam a acumular as coordenações. A idéia deste parâmetro é diminuir a sobrecarga dos super-peers da rede.

3. Espaço de Armazenamento para o Objeto (*S*) – mede o percentual de utilização de espaço para armazenamento para novos objetos no peer. Fica à critério da aplicação configurar as partições dos peers: a partição de objetos – destinada para armazenamento de objetos criados ou coordenados pelo peer – e a partição de réplicas – designada para armazenar réplicas de objetos

coordenados por outros peers do grupo. A Figura 19 ilustra um exemplo de partições configuradas com o mesmo tamanho, diferenciados pelo uso.

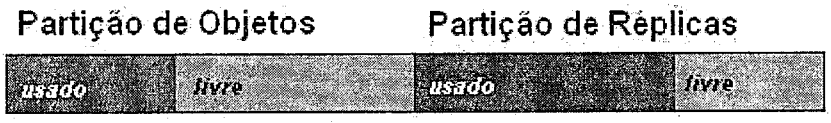


Figura 19: Exemplo de Particionamento adotado pela aplicação colaborativa

Na eleição é computada, para cada peer, a seguinte equação:

$$C(peer_i) = w_1 * A(peer_i) + w_2(peer_i) * M + w_3 * S(peer_i)$$

Equação 7

O peer que possui o maior C computado é eleito nó Master. O ajuste feito aos pesos w1, w2 e w3 será analisado na seção 4.4.4.

Na nossa simulação, o peer criador de um objeto atribui seu grau de disponibilidade e aciona os demais peers para tomar decisão sobre a replicação: a eleição do nó Master. Definido o coordenador, ele será responsável por gerar efetivamente as réplicas do objeto e monitorá-las no tempo.

4.3.3 – Quando acontece a replicação

A replicação precisa ser iniciada quando há um número de réplicas abaixo do necessário para manter o grau de disponibilidade definido pela aplicação. Podemos enumerar os dois eventos em que há a necessidade da replicação:

1. Um novo nó Master foi eleito
2. O nó Master detectou que uma ou mais réplicas não estão mais online

O 2° evento mostra que a tarefa de monitoramento é essencial para preservar a disponibilidade de um objeto de acordo com um predeterminado threshold. O nó Master tem a responsabilidade de, para cada um de seus peers coordenados, checar periodicamente se um peer está *up* e se sua réplica continua armazenada nele (p.e., uma política de remoção de réplicas poderia tê-la apagado).

Na simulação deste trabalho, o período entre checagem é feito baseado no histórico do peer, mais precisamente no tempo médio que dura o estado *up* de um peer. Portanto se um peer apresenta uma disponibilidade alta será requisitado menos vezes que um peer com menor disponibilidade.

A disponibilidade pode ser avaliada de formas distintas. No e-Donkey, a disponibilidade de algum recurso é computada como o número de peers online que o possuem. Em ÖZSU, VALDURIEZ (1999), a disponibilidade de um sistema (availability) tem sua definição baseada nas métricas Tempo Médio entre Falhas (MTBF – Medium Time Between Failures) e Tempo Médio entre Reparos (MTTR – Medium Time To Repair), definida como:

$$Availability = \frac{MTBF}{MTBF+MTTR}$$

Equação 8

Em SCHMITT et al. (2003), a partir da definição básica de disponibilidade mostrada acima, os autores refinaram a fórmula. Considere as seguintes definições:

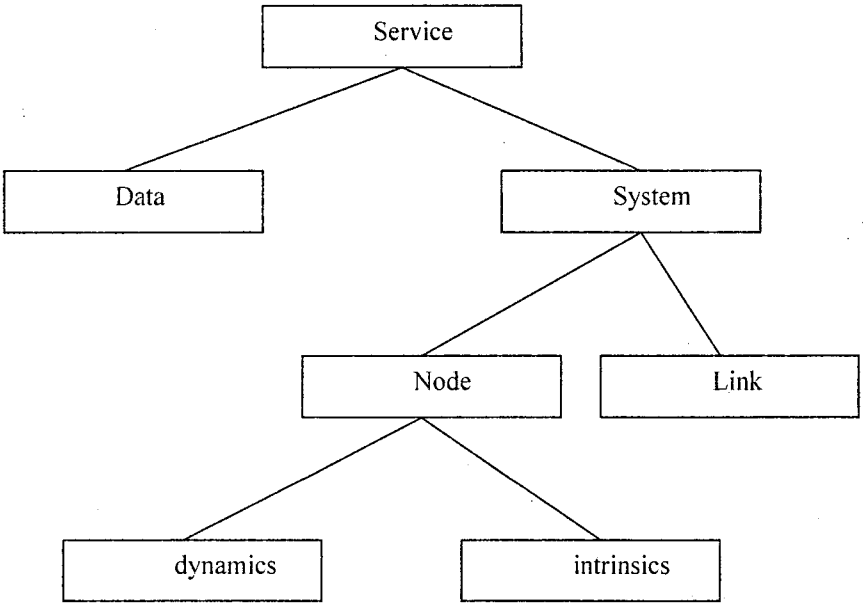
- Um serviço é disponível quando tanto os dados como o sistema no qual o serviço executa está disponível
- Um dado é disponível quando é acessível em um dado tempo
- Um sistema é disponível quando tanto os nós quanto os links de comunicação estão disponíveis
- Um link é disponível quando ele não falha e há recursos suficientes que podem ser alocados para transmitir dados requisitados por uma aplicação

- Um nó está disponível quando está UP, i.e., não está desconectado da rede (medida por *dynamics*). Ciclos de CPU, espaço para armazenamento e memória são medidas por *intrinsics*.

Baseados nestas definições, os autores do referido trabalho definiram três tipos de disponibilidade:

- $Availability_{Service} = Availability_{Data} * Availability_{System}$
- $Availability_{System} = Availability_{Node} * Availability_{Link}$
- $Availability_{Node} = Availability_{NodeDynamics} * Availability_{NodeIntrtinsics}$

De uma forma hierárquica, podemos visualizar as definições:



Existem outras formas de computar a disponibilidade. Em BHAGWAN et al. (2003), é analisada a disponibilidade também em função da hora do dia. Podemos computá-la, por exemplo, como:

1. Tempo que o Peer ficou efetivamente Online sobre o Tempo que o Peer desejava estar Online
2. Tempo Online sobre o Tempo desde o primeiro estado UP do peer no grupo da rede JXTA

3. Tempo Online sobre o Tempo desde que o grupo da rede JXTA foi criado

Para efeitos de simulação e considerando a abstração de grupo fornecida pelo JXTA, adotamos como Disponibilidade Absoluta a 2ª. métrica definida acima.

4.3.4 – Onde replicar

Definir o destino das réplicas é essencial. A abordagem de COOPER, GARCIA-MOLINA (2002) consiste basicamente em um esquema de troca de compartilhamentos entre peers de uma mesma comunidade. Em RANGANATHAN et al. (2002), os peers mais aptos são escolhidos para receberem as réplicas, considerando dois critérios para a tomada desta decisão:

1. O Custo de Armazenamento da réplica
2. O Custo de Transferência pela Rede

No modelo proposto por este trabalho, o peer Master é responsável por definir quais peers do grupo são mais indicados para receber as réplicas de seu objeto e é abstraído o custo da transferência pela rede, já que se situa no nível de um grupo JXTA de uma aplicação COPPEER. No entanto, o custo da rede vai ser ponderado de maneira indireta, pois um dos nossos critérios é a quantidade de coordenações (papel de Master) que o peer host possui. Por exemplo, se um peer possui muitas coordenações, que envolve um custo de rede para monitoramento das réplicas, ele será menos interessante para armazenar um objeto.

Adicionamos ainda um outro critério para a tomada de decisão: a disponibilidade absoluta (definida na seção anterior) do peer que é candidato a receber a réplica. Quanto mais disponível, mais apto a ser host.

Portanto, podemos enumerar os critérios da eleição do peer que recebe réplica:

1. Disponibilidade Absoluta (A)
2. Número de Coordenações (M)
3. Espaço de Armazenamento para a Réplica (S)

Note que são os mesmos critérios da eleição de um nó Master. No entanto, os pesos dados a estes critérios distinguem as duas eleições. A Equação 7 é aplicada neste caso também.

4.3.5 – Quantas réplicas serão criadas

O número de réplicas a ser gerado deve ser basicamente um compromisso entre custo / benefício. O custo, que será discutido mais detalhadamente adiante, envolve armazenamento, rede, cpu, memória, i.e., recursos computacionais. O benefício é a garantia de haver ao longo do tempo a acessibilidade desejada a um determinado objeto.

Os dois casos extremos seriam:

- Não replicar – implica em uma disponibilidade do objeto igual a do peer que o contém. Se o peer sai da rede, o objeto se torna inacessível.
- Replicar Tudo em Todos – implica em gerar réplicas de todos os objetos de cada peer em todos os peers. Envolve custos altíssimos e é impraticável.

Portanto, para determinar o número ideal de réplicas que um Master precisa gerar, iremos fazê-lo baseados na idéia de RANGANATHAN et al. (2002), com o refinamento de que cada peer possui uma disponibilidade distinta.

Suponha que cada peer contendo uma réplica tem certa disponibilidade $A(peer_i)$. A probabilidade de que todas as réplicas r estejam indisponíveis é dada por:

$$\prod_{i=1}^{r}(1-A(peer_i))$$

Equação 9

Conseqüentemente, a probabilidade de que haja pelo menos uma réplica disponível é computada da seguinte forma:

$$1-\prod_{i=1}^{r}(1-A(peer_i))$$

Equação 10

Considerando que as réplicas são encontradas por um serviço de busca, o grau de disponibilidade de um objeto pode ser delimitado pelo threshold da seguinte inequação:

$$1-\prod_{i=1}^{r}(1-A(peer_i)) \geq Threshold$$

Equação 11

Seguindo Equação 11, podemos computar o número de réplicas do serviço de garantia de disponibilidade conforme o algoritmo abaixo:

```
Function ComputeNumberOfReplicas
   numberOfReplicas = 0
   product = 1
      for each node in bestNodesInTheGroup
         if node is UP then
            product= product * (1 - Availability(node))
            numberOfReplicas= numberOfReplicas + 1
         end if
         availability = 1 - product;
         if availability >= threshold then
            break
         end if
      end for
   return numberOfReplicas
End of Function
```

Note que o número de réplicas depende dos melhores nós escolhidos no grupo, segundo o critério de eleição de peers definido anteriormente. O número de réplicas depende apenas da qualidade dos nós existentes e, quanto mais nós existirem, maior a probabilidade de haver nós com boa qualidade, i.e., que atendam melhor o critério da eleição de peers.

Vejamos um exemplo: suponha uma rede contendo 100 peers que possuem disponibilidade igual a 50% cada. Nossa aplicação Editor Colaborativo de Ontologias plugada ao COPPEER requer que o recurso OntologiaGeral tenha grau de disponibilidade igual 90% .

Da Equação 11, temos:

$$(1-(1-0.5)^r) \geq 0.9$$

$$= (1-(0.5)^r) \geq 0.9$$

Iteração	Resultado
$r = 1 \Rightarrow 0.5 \geq 0.9$	Falso
$r = 2 \Rightarrow 0.75 \geq 0.9$	Falso
$r = 3 \Rightarrow 0.875 \geq 0.9$	Falso
$r = 4 \Rightarrow 0.9375 \geq 0.9$	Verdadeiro

Portanto, serão necessárias 4 réplicas para garantir a disponibilidade do recurso OntologiaGeral.

4.3.6 – Monitoramento das Réplicas

Em uma rede peer-to-peer, mudanças de estados ocorrem constantemente em seus nós, i.e., os peers entram e saem da rede a qualquer tempo. Este cenário pode ser visto como um sistema falível, onde cada componente que falha deve ser recuperado e durante a recuperação um componente redundante deve assumir a função do anterior. Para que um sistema seja resistente à falhas é necessário que ele as detecte, conforme abordado na seção 2.2.3. O quanto antes detectá-las, melhor será a resposta ao problema.

Não podemos nos restringir e assumir que a falha é voluntária, i.e., quando um peer sair da rede ele irá enviar uma mensagem comunicando o evento. De fato, ocorrem muitas falhas involuntárias, como queda da conexão, problemas em algum nível da pilha de protocolos da rede, etc.

A fim de detectar efetivamente quedas de peers existe a técnica de *pooling*, que consiste basicamente em enviar periodicamente mensagens entre pares de peers para checar seu estado na rede. Uma questão que surge diante da aplicação do *pooling* é quais pares de redes serão arranjados para um monitoramento ótimo, aquele que minimiza o tempo de detecção de uma falha no sistema.

No modelo simulado, agregamos ao nó Master a responsabilidade de monitorar as suas réplicas. Conforme mostrado na Figura 18, para cada objeto são montados elos entre os peers que armazenam as réplicas e o nó Master, que os monitora através do envio de mensagens periódicas pela rede. A periodicidade da transmissão de mensagens é determinada pela média de duração de um estado *up* de cada peer, ou seja, quanto maior for esta média de um peer X, menos mensagens um nó Master irá enviar para X. A Figura 20 ilustra o processo de monitoramento.

Um nó Master aguarda uma resposta do peer coordenado durante certo tempo (*timeout*). Quando este tempo expira, o coordenador considera a falha como uma perda da réplica e reinicia o processo de replicação do objeto para estabilizar novamente o grau de disponibilidade.

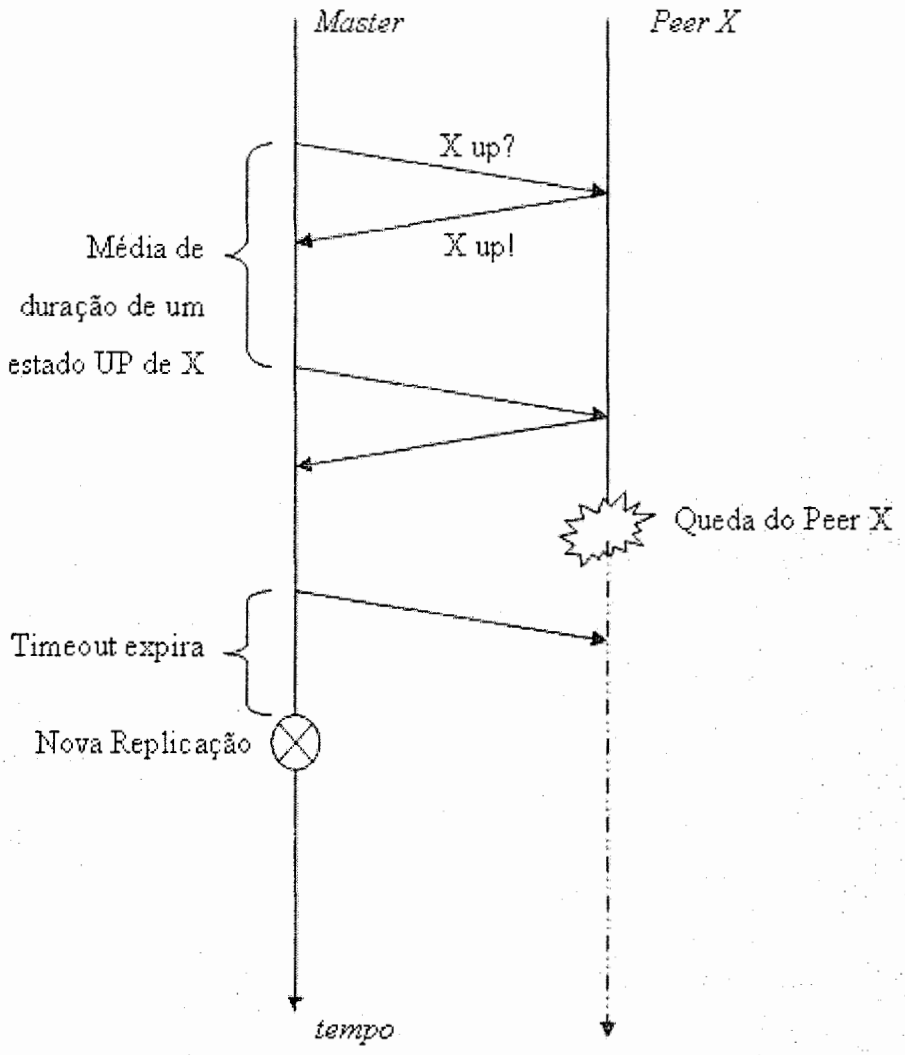


Figura 20: Monitoramento ao longo do tempo

4.3.7 – Falhas no Coordenador (Master)

Um nó coordenador em geral é um peer que possui uma disponibilidade maior que os demais membros que mantêm suas réplicas. No entanto, o nó coordenador também está suscetível à falhas.

Quando um nó Master falha e, portanto fica inacessível, ele perde este papel de coordenação na rede. Os peers que contém réplicas recebem normalmente uma mensagem de checagem de seus estados pelo nó coordenador. Quando esta mensagem

não chega, o peer busca pelo nó Master. Caso se confirme a suspeita da ausência do coordenador, o peer inicia uma nova eleição para coordenação.

4.3.8 – Custos da Replicação

A replicação envolve custos computacionais a serem ponderados quando se decidi que determinado recurso deve possuir um grau de disponibilidade específico. Os custos estão relacionados a consumo de banda na rede, espaço usado para armazenamento de objetos e réplicas e, em menor escala, ao tempo processamento da CPU e memória principal alocada para o serviço.

Através da Tabela 2 mostrada a seguir são discriminados os custos das principais atividades do modelo proposto por este trabalho.

Tabela 2: Custos da Replicação

Atividade	Requisito	Rede	Armazenamento
Eleição de Master	Troca de Mensagens	Sim	Não
Eleição de Peers Hosts	Troca de Mensagens	Sim	Não
Monitoramento	Troca de Mensagens	Sim	Não
Replicação	Transferência de Réplicas	Sim	Sim

4.4 – Avaliação do Modelo

O Modelo proposto neste trabalho passou por quatro etapas de avaliação, cada uma servindo de insumo para a etapa posterior. As duas primeiras etapas foram computadas através de simulação, a 3ª. etapa usou análise estatística e a última a técnica de Algoritmos Genéticos.

4.4.1 – O Simulador

Identificado o problema da disponibilidade dos recursos e definida uma proposta de resolução através do modelo de replicação descrito anteriormente, à simulação foi delegada a tarefa de relatar o comportamento da rede de peers usando o serviço de garantia de disponibilidade (SGD). O simulador tem o propósito de avaliar efetivamente o SGD proposto ao COPPEER.

O COPS (COPPEER Simulator) foi desenvolvido em Java, utilizando a *engine* de simulação de eventos discretos denominada Jist[41] - Java in Simulation Time. Essa tecnologia é o coração do COPS, pois ela simplifica bastante a manipulação do tempo simulado. A Figura 21 mostra a GUI do COPS simulando uma rede com 1 recurso e 10 peers, onde o SGD elegeu 1 nó Master e 6 nós peer hosts.

[41] http://jist.ece.cornell.edu/

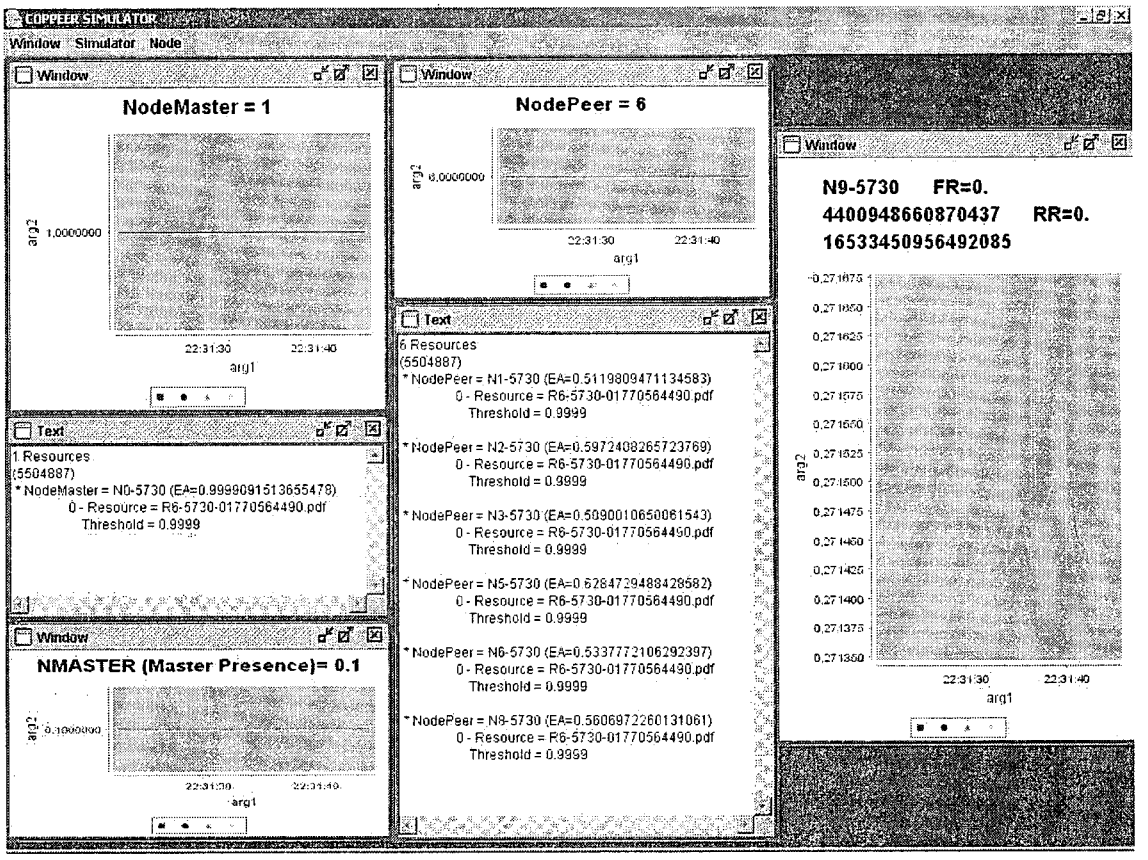


Figura 21: GUI do COPPEER Simulator

Outras tecnologias foram agregadas ao COPS, como:

- JFreeChart[42] – usado aqui para geração de gráficos de acompanhamento e análise de resultados
- XMLBeans[43] – usado para mapear arquivos XML em classes Java e vice-versa, facilitando sua manipulação

Outras configurações são necessárias ao COPS para ajustar a máquina virtual Java para um melhor desempenho da aplicação, tal como o uso da tecnologia HotSpot (veremos adiante) em modo servidor e alteração nas partições de memória da Java Virtual Machine.

[42] http://www.jfree.org/jfreechart/

[43] http://xmlbeans.apache.org/

4.4.1.1 – Parâmetros de Entrada

O funcionamento do COPS consiste receber como entrada um cenário de simulação, calcular o comportamento da rede durante o tempo especificado no cenário e gerar um log de resultados como saída. Uma classe Analisador de Log compila e armazena os resultados em um SGBD relacional (MySQL[44]), facilitando a manipulação dos dados.

Na Figura 22 há o esquema dos cenários do COPS seguido de um exemplo na Figura 23. Cada nó da árvore XML é um parâmetro analisado pelo COPS durante a simulação, cujas funções são:

- Volatility – este parâmetro especifica a volatilidade da rede simulada. Cada nó da rede possui uma Taxa de Falhas (probabilidade de falhar/sair da rede) e uma Taxa de Recuperação (probabilidade de se recuperar/voltar à rede). Essas taxas são distribuídas segundo uma lei de potências [45], usando a volatilidade como expoente.
- NumberOfNodes – este parâmetro determina quantos nós/peers existem na rede simulada.
- NumberOfSuperNodes – determina o número de super-peers haverá na rede. No COPS, um super-peer é um peer que possui as Taxas de Recuperação e Falhas pré-definidas pelo usuário e geralmente são taxas que permitem disponibilidade absoluta próxima de um.
- Failure Rate – define a taxa de falhas do supernode.
- Repair Rate – define a taxa de recuperação do supernode.
- InitialNumberOfResources – define o número inicial de recursos na rede, ou seja, quantos objetos terão que ser trabalhados para garantir suas disponibilidades.

---

[44] http://www.mysql.com/

[45] Uma distribuição é dita como uma distribuição de lei de potência (power law distribution), se P(k) ~ k^-y, onde y >0 é chamado de expoente da distribuição.

- ResourceCreationProbability – determina a probabilidade de criação de recursos em cada peer. Um recurso, quando criado, passa a necessitar do serviço de garantia de disponibilidade, segundo seu threshold (adiante).
- ResourceThreshold – determina qual é o valor do threshold, i.e., o grau de disponibilidade dos recursos criados na rede. Pode ser aleatório (valor=RAND).
- UnlimitedSpace – determina se os peers simulados não possuem restrição de armazenamento de objetos, i.e., se o espaço é ilimitado. Nas simulações estudadas, consideramos sempre este parâmetro como falso (false).
- ObjectPartitionSize – define o tamanho da partição dos objetos dos peers. Esta partição armazena objetos coordenados e/ou criados pelo peer.
- ReplicaPartitionSize – define o tamanho da partição de réplicas dos peers. Armazena as réplicas no peer enviadas por um Master.
- MasterElectionWeights – grupo de nós XML que contém os pesos normalizados da eleição Master.
- ReplicationElectionWeights – grupo de nós XML que contém os pesos normalizados da eleição de peers hosts de réplicas.
- AbsoluteAvailability – este peso determina quão importante para decisão na eleição da Disponibilidade Absoluta dos peers.
- MasterRoles – este peso se refere a quantos papéis de Master um peer possui. Quanto maior o número de papéis Master, menos interessante o peer para ser eleito. Normalmente este peso possui valor negativo (entre -1 e 0).
- ObjectSpaceUsageCost – este peso determina o quão importante será para a eleição a quantidade de espaço disponível na partição de objetos de um peer.
- ReplicaSpaceUsageCost – este peso determina o quão importante será para a eleição a quantidade de espaço disponível na partição de réplicas de um peer.
- SimulationTime – tempo de simulação que o cenário deve levar. O tempo de simulação é controlado pela *engine* do Jist e tratado dentro do COPS.

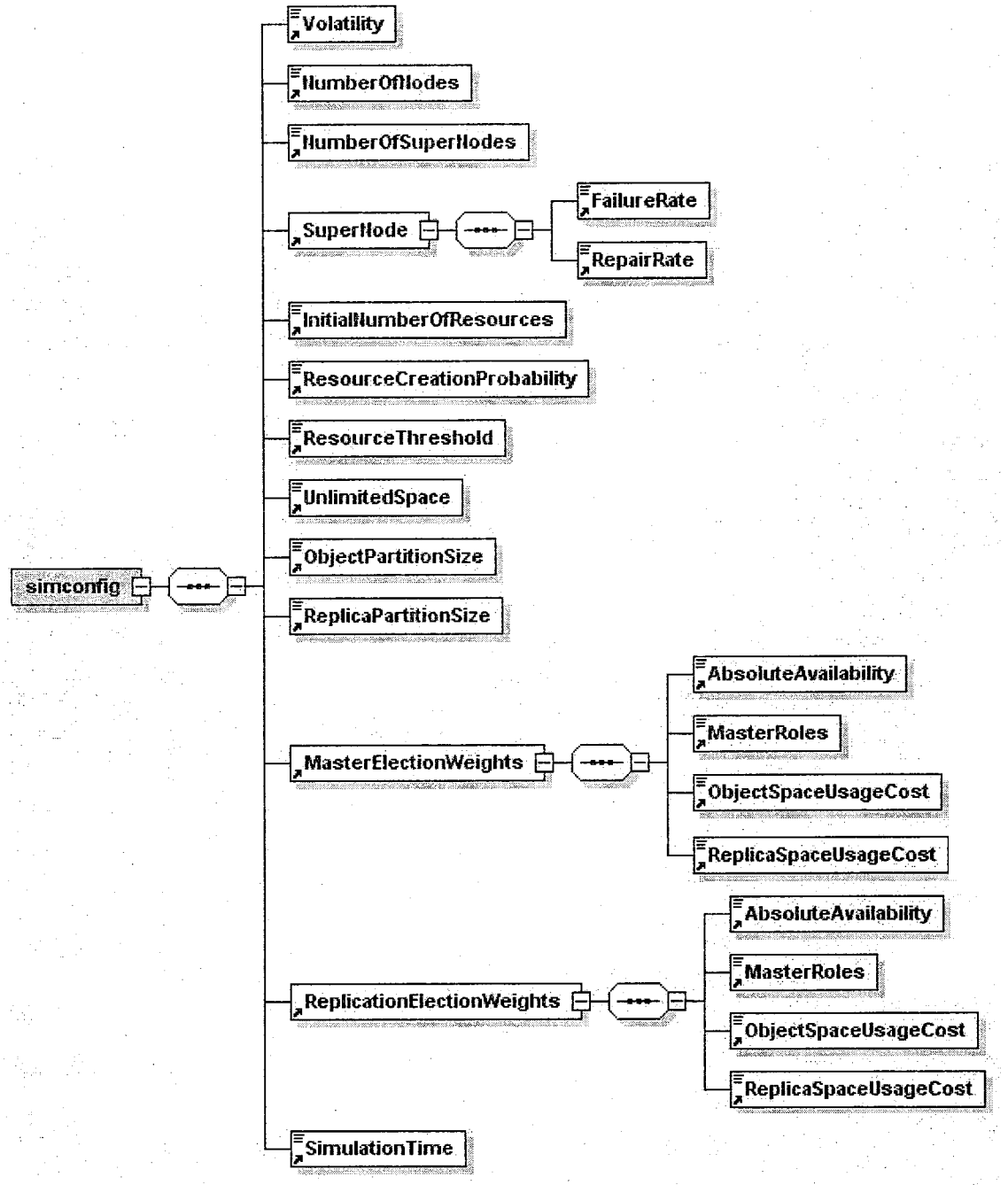


Figura 22: Esquema da Configuração do COPPEER Simulator

```
<?xml version="1.0" encoding="UTF-8"?>
<simconfig targetNamespace="http://coppeer.cos.ufrj.br"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://coppeer.cos.ufrj.br">

<!-- Heuristic created by ScenarioCreator -->

<Volatility>0.5</Volatility>
<NumberOfNodes>256</NumberOfNodes>
<NumberOfSuperNodes>0</NumberOfSuperNodes>
<SuperNode>
<FailureRate>0.001</FailureRate>
<RepairRate>0.9999</RepairRate>
</SuperNode>
<InitialNumberOfResources>5</InitialNumberOfResources>
<ResourceCreationProbability>0.00000</ResourceCreationProba
bility>
<ResourceThreshold>0.95</ResourceThreshold>
<UnlimitedSpace>false</UnlimitedSpace>
<ObjectPartitionSize>999999999</ObjectPartitionSize>
<ReplicaPartitionSize>2000000</ReplicaPartitionSize>
<MasterElectionWeights>
<AbsoluteAvailability>0.9</AbsoluteAvailability>
<MasterRoles>-0.1</MasterRoles>
<ObjectSpaceUsageCost>0.2</ObjectSpaceUsageCost>
<ReplicaSpaceUsageCost>0.0</ReplicaSpaceUsageCost>
</MasterElectionWeights>
<ReplicationElectionWeights>
<AbsoluteAvailability>0.8</AbsoluteAvailability>
<MasterRoles>-0.6</MasterRoles>
<ObjectSpaceUsageCost>0.0</ObjectSpaceUsageCost>
<ReplicaSpaceUsageCost>0.6</ReplicaSpaceUsageCost>
</ReplicationElectionWeights>
<SimulationTime>2139999999</SimulationTime>
</simconfig>
```

Figura 23: Exemplo de Configuração do COPS

4.4.1.2 – Parâmetros de Saída

Os parâmetros de saída são armazenados nos logs da simulação à medida que são gerados. Esses parâmetros buscam caracterizar a rede de forma a possibilitar uma analise comparativa entre diferentes configurações de entrada. Cada um deles varia no tempo de simulação e, portanto, a saída apresenta uma média aritmética dos parâmetros. São eles:

1. Eficiência – medida do desempenho do algoritmo de replicação. É definida em função do Threshold Real, i.e., quão disponível se tornou um objeto e do Threshold Desejado – o grau de disponibilidade definido a priori pela aplicação. É realizada a soma dos threholds para cada objeto (n objetos) a fim de permitir uma análise global do resultado.
2. $$Eficiência = \frac{\sum_{i=1}^{n} ThresholdReal(O_i)}{\sum_{i=1}^{n} ThresholdDesejado(O_i)}$$

Equação 12

3. Esforço – se refere à quantidade de réplicas que foram necessárias para atingir o objetivo de manter o grau de disponibilidade requerido. É definido em função do número de réplicas geradas por objetos e pelo número de objetos originais (não-réplicas) da rede. O somatório é realizado de maneira análoga ao da Eficiência.

$$Esforço = \frac{\sum_{i=1}^{n} NúmeroDeRéplicas(O_i)}{n}$$

Equação 13

4. Espalhamento – medida inversa ao agrupamento de coordenações. É definida em função do número de nós que possuem alguma coordenação,

i.e., são Masters de algum objeto e do número total de nós (inclusive supernós) na rede.

$$Espalhamento = \frac{NúmeroDeNósComCoordenação}{NúmeroTotalDeNós}$$

Equação 14

4.4.1.3 – Desempenho do Simulador

O desempenho da simulação foi assunto estudado durante este trabalho. Existem muitos simuladores de eventos discretos, sendo um dos mais citados o NS - Network Simulator[46]. No nosso caso foi adotada a *engine* de simulação do Jist, trazendo consigo as seguintes vantagens:

- Escrever a simulação na mesma Linguagem de Programação que o COPPEER: Java
- Bom desempenho (veja o quadro comparativo[47] com demais tecnologias)
- Simplicidade de uso da API

Uma vez definida a tecnologia de simulação e escrito o simulador, foi necessário ajustar a máquina virtual a fim de adequar configurações de alocação de memória e garbage collection. Foi fundamental o uso da tecnologia HotSpot da JVM.

A Tecnologia HotSpot é responsável pela melhora no desempenho da máquina virtual Java (JVM). Incorporada a partir do Java 2 Standard Edition (J2SE) versão 1.4.1, o HotSpot se insere nas fundações da JVM. As mais distintas aplicações de Java, que

[46] http://www.isi.edu/nsnam/ns/

[47] Quadro comparativo de Jist com outras tecnologias em http://jist.ece.cornell.edu/

variam de celulares a mainframes, são beneficiadas com mais desempenho e escalabilidade.

Seu impacto alcança até a reusabilidade de código, pois se dá mais liberdade ao programador desenvolver métodos virtuais e abstrações que o otimizador será capaz de gerar um bom desempenho.

4.4.2 – Escolha das Configurações

As configurações para o COPS permitem um estudo abrangente do comportamento de uma rede JXTA usando os mecanismos de eleição citados anteriormente. Compondo diferentes parâmetros de entrada, o simulador permite a análise detalhada de resultados bastante diversificados.

Conforme foi visto na seção 4.4.1.1, existem muitos parâmetros, o que implica em um número grande de configurações possíveis. É importante notar que, muitos dos atributos de configuração são características da rede, ou seja, o ambiente o qual o Serviço de Garantia de Disponibilidade estaria atuando.

Para efeitos de demonstração do modelo aqui proposto e considerando o custo computacional da realização de sucessivas simulações, dentro do grande número de configurações possíveis, foram escolhidas aquelas que são importantes para cada etapa, mantendo os demais parâmetros fixos.

4.4.3 – Primeira Etapa

A primeira etapa teve caráter exploratório, buscando pelas variáveis que poderiam ser determinantes para o funcionamento do modelo. Foram geradas configurações para cenários com:

- 64, 256 e 1024 peers
- Thresholds iguais a 25%, 50% e 90%

- Definidas 5 heurísticas distintas para os pesos das eleições de Master e Peers, conforme especificado abaixo:
    - Heurística 1 : Aleatória
    - Heurística 2 : Critério único é a Disponibilidade dos peers
    - Heurística 3 : Critério único é o Nº. de Coordenações dos peers
    - Heurística 4 : Critério único é o Espaço Disponível dos peers
    - Heurística 5 : Critério que prioriza a Disponibilidade dos Peers mas considera também o Espaço Disponível e o Nº. de Coordenações

Temos um total de 45 configurações computadas nesta etapa. Em todas elas, os peers tinham baixa probabilidade (0,01%) de criação de novos recursos, o que justifica em alguns resultados a queda do esforço.

4.4.3.1 – Resultados

Os resultados desta etapa descrevem eficiências muito distintas entre as configurações. Na verdade, a eficiência calculada nesta etapa foi computada como a razão entre a Disponibilidade Real do Recurso ao fim da simulação sobre o Threshold Desejado (Equação 15). A Disponibilidade Real foi definida como o produto das disponibilidades dos peers que contém um dado recurso (Equação 16).

$$Eficiência_1 = \frac{Disponibilidade\,\mathrm{Re}\,al(recurso)}{ThresholdDesejado}$$

Equação 15

$$Disponibilidade\,\mathrm{Re}\,al(recurso) = \prod_{i=1}^{r} Disponibilidade(peer_i)$$

Equação 16

A definição da Eficiência em função da Disponibilidade Real do recurso gerou resultados muito próximos do parâmetro de saída Esforço, de forma inversa. À medida que o Esforço aumenta, i.e., mais replicas são necessárias, *r* aumenta, o produto da disponibilidade dos peers decresce e, portanto, a Disponibilidade Real diminui. Logo, neste caso, a eficiência é inversamente proporcional ao esforço. Nos gráficos do Anexo B – Resultados da Etapa pode ser observada esta relação. Portanto a análise da eficiência desta etapa é equivalente a análise do esforço, feita a seguir. Na segunda etapa a eficiência recebeu uma redefinição mais apropriada (Equação 12) para análise final do trabalho.

O esforço variou conforme o número de nós, o threshold dos recursos e os pesos pré-definidos das eleições. Para as Heurísticas 2 e 5, a variação do número de nós e do threshold teve pouco impacto nos resultados. As Heurísticas 1,3 e 5 apresentaram a necessidades de maiores esforços e atingiram um esforço de até 25 vezes maior que os das Heurísticas 2 e 5.

O espalhamento apresentou variações conforme o número de nós, o threshold dos recursos e os pesos pré-definidos das eleições. À medida que o número de nós cresce, o espalhamento apresentou decrescimento, para as 5 Heurísticas. O espalhamento cresce conforme o threshold. A Heurística que apresentou maior espalhamento foi a 3, seguida pelas 1 e 4. O desempenho das Heurísticas 2 e 5 foi pior para este parâmetro de saída.

4.4.4 – Segunda Etapa

Conforme os resultados apresentados na primeira etapa, pode ser notado que os pesos das eleições são fundamentais para um bom desempenho do Serviço de Garantia de Disponibilidade. Nesta etapa, os experimentos foram preparados, através da criação de configurações específicas, variando apenas os pesos das eleições. Os demais parâmetros de configuração foram especificados idealizando o comportamento de uma rede qualquer.

Conforme a definição de configuração do simulador (vide Figura 22), nesta etapa temos os seguintes valores fixados:

- Volatility = 0.5
- NumberOfNodes = 256
- NumberOfSuperNodes = 0
- InitialNumberOfResources = 5
- ResourceCreationProbability = 0
- ResourceThreshold = 90%
- ObjectPartitionSize = 999999999
- ReplicaPartitionSize = 2000000
- SimulationTime = 100000

Quanto à variação dos pesos das eleições, devemos considerar que:

1. Os pesos foram normalizados e, portanto cada peso possui um valor absoluto entre zero e um

2. A fim de reduzir o número de combinações possíveis, cada peso pode assumir os valores absolutos: 0.0, 0.5 e 1.0
3. Aqueles pesos que são fatores multiplicativos das disponibilidades em cada eleição, são sempre maiores ou iguais aos outros dois pesos da eleição (número de coordenações e espaço de armazenamento) – vide Equação 7. Este critério foi estabelecido já que sabemos a priori que o Threshold (Equação 11), nosso maior objetivo, é calculado baseado na Disponibilidade e portanto, deve ser privilegiado.
4. Durante a eleição do Master, dentre os custos de armazenamento (ObjectSpaceUsageCost e ReplicaSpaceUsageCost), devemos considerar apenas o custo de armazenamento de objetos, já que o objeto coordenado será depositado na partição de objetos. Este fato exclui uma variável do problema.
5. De maneira análoga ao do item anterior, durante a eleição dos peers hosts, devemos considerar apenas o custo de armazenamento de réplicas, já que a réplica será depositada na partição de réplicas. Exclui outra variável do problema.

Feitas as devidas considerações, temos seis variáveis correspondentes aos pesos (três para cada eleição), cada uma podendo assumir três valores distintos. Desta forma, teríamos um total de $3^6 = 729$ configurações possíveis a serem simuladas (sem aplicar a 3ª. consideração). Acionando a terceira consideração, onde é privilegiada a disponibilidade, temos um total de 195 configurações possíveis.

Cada uma das 195 configurações foi simulada três vezes (3 *rounds*) e computada uma média aritmética como resultado final.

4.4.4.1 – Resultados

Conforme pode ser visto no Anexo C – Resultados da Etapa 2, os experimentos mostraram que, para as 195 configurações simuladas, a eficiência apresentou pequenas variações, assim como o espalhamento. A variável que apresentou um desvio padrão

significativo foi o Esforço de Replicação e esta foi o alvo da análise de regressão realizada na etapa posterior.

4.4.5 – Terceira Etapa

A terceira etapa consistiu na análise dos resultados usando um pacote estatístico com método de Regressão Linear Múltipla. A regressão foi realizada sobre os dados da segunda etapa.

Através da regressão é obtida uma fórmula matemática que descreve cada parâmetro de saída da simulação (eficiência, esforço e espalhamento) em função das entradas do algoritmo (os pesos das eleições). A partir dos modelos gerados é possível compreender melhor a relação entre as variáveis do simulador.

O pacote estatístico usado foi o Analyze-It[48] versão 1.68, da empresa homônima ao software. Esta ferramenta é um add-in do Microsoft® Excel que permite, dentre outras funcionalidades, a análise de regressão de dados em uma planilha.

4.4.5.1 – Análise de Regressão

Em um estudo da relação entre as alturas dos pais e filhos, Francis Galton (1822-1911) usou pela primeira vez o termo **regressão** para análise estatística. Desde então, os modelos de regressão vêm sendo amplamente aplicadas em todas as áreas do conhecimento: computação, administração, engenharias, biologia, agronomia, saúde, sociologia, etc.

Os modelos de regressão utilizam variáveis, contínuas ou discretas que são o objeto de estudo (target) e a partir delas tentamos aproximar uma função que descreve o fenômeno observado. A função descreve aproximadamente o relacionamento entre variáveis preditoras e variáveis de resposta.

[48] http://www.analyse-it.com/

4.4.5.1.1 – Regressão Linear Simples

Existem fenômenos que apresentam relações entre variáveis que os descrevem. Podemos pensar alguns fenômenos como: quanto mais um político gasta em propaganda, mais chances ele tem de vencer; quanto mais se estuda, melhor nota é obtida; quanto mais dorme, menos sono há. A expressão de causa e efeito pode ser descrita através de uma função matemática que relaciona variáveis dependentes com as independentes. Por exemplo, os gastos do político são independentes, enquanto que a chance dele vencer depende de seus gastos. Através da observação dos fenômenos são formuladas hipóteses dadas por funções. Uma ferramenta que provê uma hipótese ou teoria a ser testada é a Regressão Linear. A Regressão Linear Simples fornece um modelo dado por:

$$Y = \beta_0 + \beta_1 X + \varepsilon$$

Equação 17

O parâmetro ε é o erro pertinente ao modelo, chamado de erro residual.

Suponha que se observa o fenômeno do consumo relacionado à renda familiar. Os dados coletados estão descritos conforme a tabela a seguir:

Tabela 3: Dados coletados relacionando Consumo e Renda Familiar

Consumo (R$)	Renda Familiar (R$)
723	8246
780	8742
990	9048
1634	10584
1189	10626
1295	10984
1025	11822
1792	12532

1328	12952
780	13220
1366	13386
2950	13746
1273	13946
1953	14206
866	14388
2125	14622
2372	15032
2477	15172

Estes dados da Tabela 3 plotados em um plano cartesiano são mostrados na Figura 24.

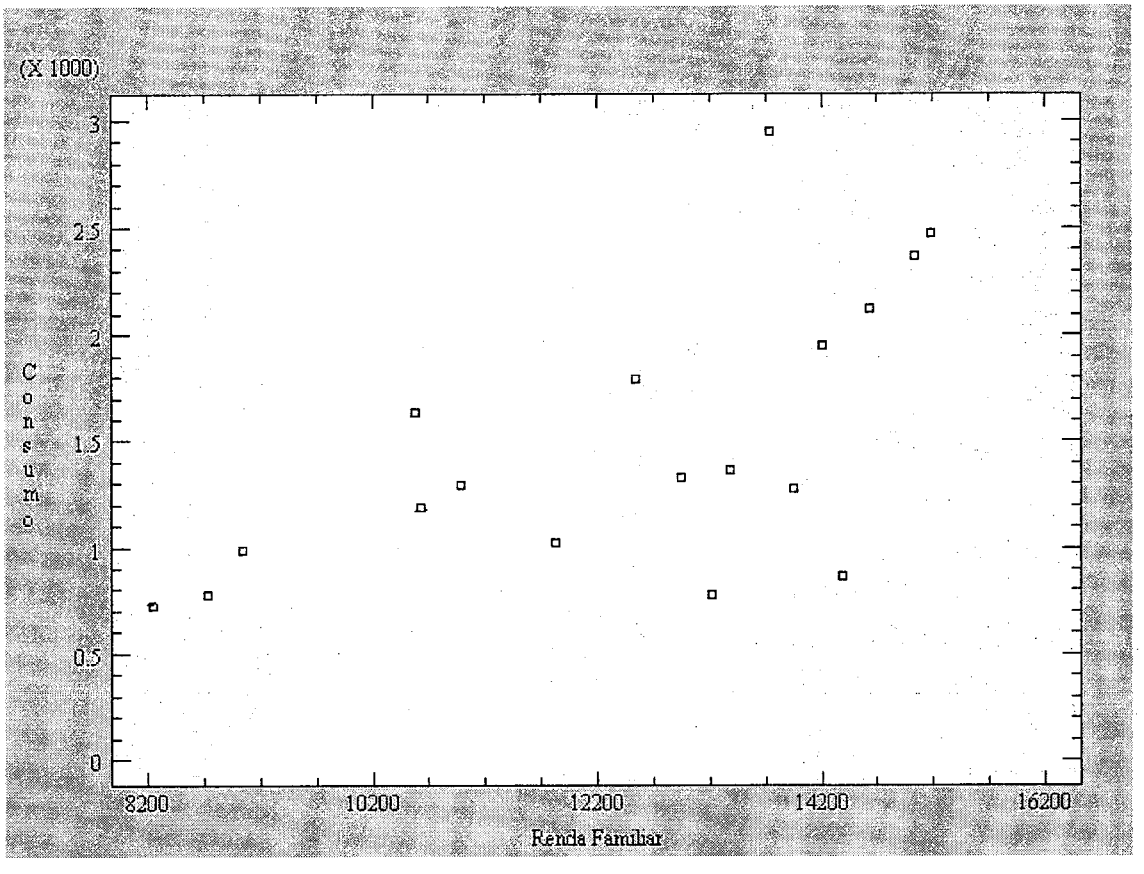


Figura 24: Gráfico Renda Familiar x Consumo

A hipótese formulada para este fenômeno é dada por uma equação do tipo $Y = aX + b$, o que nos fornece uma reta. Intuitivamente, é desejável que a reta passe próximo ao maior número possível de pontos do gráfico. Um dos métodos de regressão linear mais usados para este fim é o Método dos Mínimos Quadrados (MMQ) (SHROEDER et al., 1986), que minimiza a distância vertical entre a reta e os pontos do gráfico simultaneamente. No exemplo dado da renda familiar, a reta fornecida pelo MMQ é dada pela Equação 18 e plotada na Figura 25:

$$Consumo = 0.182037 * RendaFamiliar - 762.363$$

Equação 18

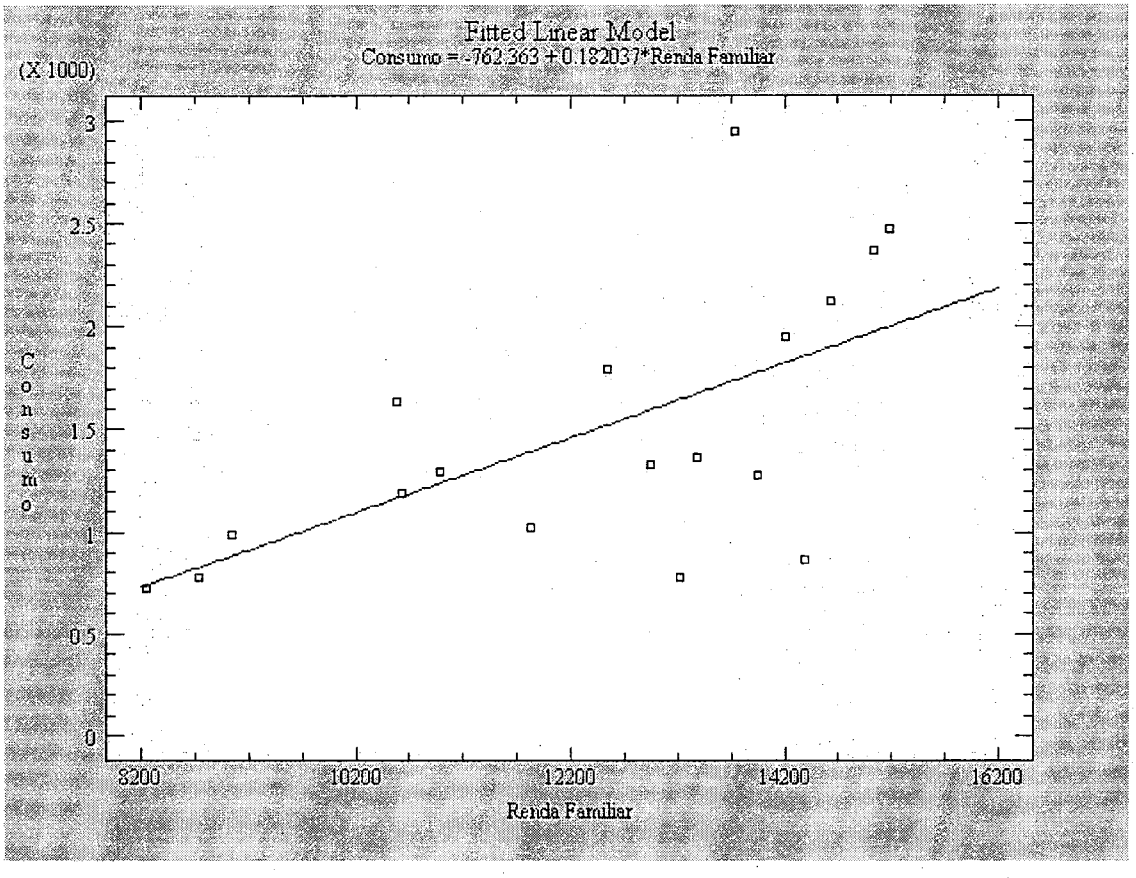


Figura 25: Reta gerada a partir do MMQ que aproxima o fenômeno Renda Familiar x Consumo

4.4.5.1.2 – Regressão Linear Múltipla

Nos exemplos trabalhados anteriormente, os fenômenos observados atribuíam à variável dependente apenas uma variável independente. A Regressão Linear Múltipla funciona de maneira análoga à Regressão Linear Simples, com o acréscimo de quantas variáveis independentes conforme necessário.

O modelo da regressão linear múltipla é dado pela seguinte equação:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_2 X_2 + ... + \beta_n X_n + \varepsilon$$

Equação 19

4.4.5.1.3 – Teste de Hipótese

O Teste de Hipótese (SHROEDER et al., 1986) nos permite comprovar teorias para uma população inteira a partir de algumas amostras. É preciso validar o modelo resultante da análise de regressão linear múltipla, no caso, através do teste de hipótese.

4.4.5.2 – Resultados

Computando todas as observações realizadas na etapa anterior de simulação, foi possível chegar a uma massa de dados capaz de alimentar o pacote estatístico que por sua vez gerou um modelo matemático para a análise do problema. O modelo foi validado através dos testes de Hipótese e pode ser visto com mais detalhes no Anexo D – Resultados da Etapa 3.

A análise de regressão linear múltipla nos forneceu os seguintes resultados para Eficiência do Algoritmo de Garantia de Disponibilidade (*E1*), Esforço de Replicação (*E2*) e Espalhamento das Réplicas (*E3*) em função das variáveis independentes que são os pesos das eleições de peers Masters (*Mw*) e peers que armazenam as réplicas (*Rw*):

$$E_1 = 1{,}11108871260058$$

Equação 20

$$E_2 = 58{,}8622 - 32{,}7575\mathrm{Mw}_0 - 4{,}5888\mathrm{Mw}_1 - 21{,}1939\mathrm{Mw}_2 - 7{,}2973\mathrm{Rw}_0 - 18{,}2387\mathrm{Rw}_1 + 0{,}8322\mathrm{Rw}_2$$

Equação 21

$$E_3 = 0{,}0102 - 0{,}0075\mathrm{Mw}_0 - 0{,}0134\mathrm{Mw}_1 - 0{,}0005\mathrm{Rw}_0 - 0{,}0057\mathrm{Rw}_1$$

Equação 22

O modelo apresentado pela Equação 20, Equação 21 e Equação 22 comprovam o que havia sido notado empiricamente na etapa dois: o esforço apresenta maior variação em função dos pesos das eleições. Portanto, o esforço passa a ser a variável principal que determina que pesos devem ser adotados para as eleições de forma a minimizar seus custos. Tendo em vistas que os pesos possuem valor absoluto entre 0 e 1 (*Mw1* e *Rw1* possuem pesos negativos), teremos menor esforço possível quando minimizarmos *E2*. A Tabela 4 mostra os esforços dados pelo modelo gerado a partir da regressão e pela simulação.

Tabela 4: Pesos que minimizam o esforço de replicação

Mw0	1,0
Mw1	0,0
Mw2	1,0
Rw0	1,0
Rw1	0,0
Rw2	1,0
Esforço Simulado	6,6700
Esforço da Equação 23	-1,5543

$$E_2 = 58,8622 - 32,7575(1) - 4,5888(0) - 21,1939(1) - 7,2973(1) - 18,2387(0) + 0,8322(1) \Rightarrow E_2 = -1,5543$$

Equação 23

A partir dos resultados computados para minimização do esforço de replicação, os critérios a serem considerados na eleição do nó Master são a disponibilidade e sua capacidade de armazenamento, assim como os critérios para eleição de um nó que armazena uma réplica terá mais chances de ser escolhido se possuir disponibilidade e capacidade de armazenamento alta. O critério de número de coordenações fica descartado durante as eleições.

4.4.6 – Quarta Etapa

Nesta etapa, optamos por trabalhar com outra abordagem para otimizar o resultado da busca. Aqui procuramos concentrar a atenção em obter o menor esforço de replicação possível nas simulações, visto que as etapas anteriores demonstraram uma variância pouco significativa para os parâmetros de saída eficiência e espalhamento.

A técnica adotada para esta etapa foi o uso de Algoritmos Genéticos, conforme veremos a seguir.

4.4.6.1 – Algoritmos Genéticos

Algoritmos Genéticos são algoritmos de busca baseados na idéia da seleção natural. A cada geração, um novo conjunto de indivíduos (comumente representados por uma string de bits) é criado usando características (pedaços da string de bits) dos mais aptos da geração anterior. Os Algoritmos Genéticos exploram de maneira eficiente informações históricas a fim de especular novos pontos que resultem em desempenho melhorado.

Quatro operações são aplicadas a cada geração: Avaliação (computar a adequação dos indivíduos), Seleção (selecionar os mais aptos), Crossover (fazer o

cruzamento dos genes dos selecionados) e Mutação (inserir um fator aleatório nos genes dos novos indivíduos). Quando os algoritmos genéticos executam pela primeira vez, os indivíduos da população recebem genes aleatórios.

No estudo feito por DE JONG (1975) sobre algoritmos genéticos, é sugerido que, para maximizar o desempenho, seu conjunto de parâmetros deve possuir uma alta probabilidade de crossover, uma baixa probabilidade de mutação e um tamanho moderado para população.

No COPS, os parâmetros atributos para as simulações com Algoritmos Genéticos foram definidos conforme mostra a Tabela 5.

Tabela 5: Parâmetros do Algoritmo Genético configurados para o COPS

Parâmetro	**Valor**
Número de Gens	60
Número de Indivíduos	128
Taxa de Crossover	100%
Taxa de Mutação	1%

4.4.6.2 – Resultados

Ao utilizar algoritmos genéticos, é preciso estabelecer critérios de parada. Alguns critérios comumente usados são:

1. Gerações – o algoritmo atinge um número pré-definido de gerações e deve parar.
2. Tempo – o algoritmo executa por um determinado tempo.
3. Limite de fitness – o melhor valor da função de avaliação da geração corrente supera um determinado limite pré-definido.
4. Stall generations – o algoritmo executa um determinado número de gerações sem melhoria na função de avaliação.
5. Stall time limit – o algoritmo executa um determinado tempo sem melhoria na função de avaliação.

No COPS, utilizamos o critério 4 pré-definindo 13 gerações como limite de melhoria na função de avaliação. Foram avaliadas 51 gerações, totalizando 6528 indivíduos, onde cada um foi simulado 3 vezes e extraída uma média aritmética de seu output. A Figura 26 mostra, em média, qual foi o esforço totalizado por cada geração.

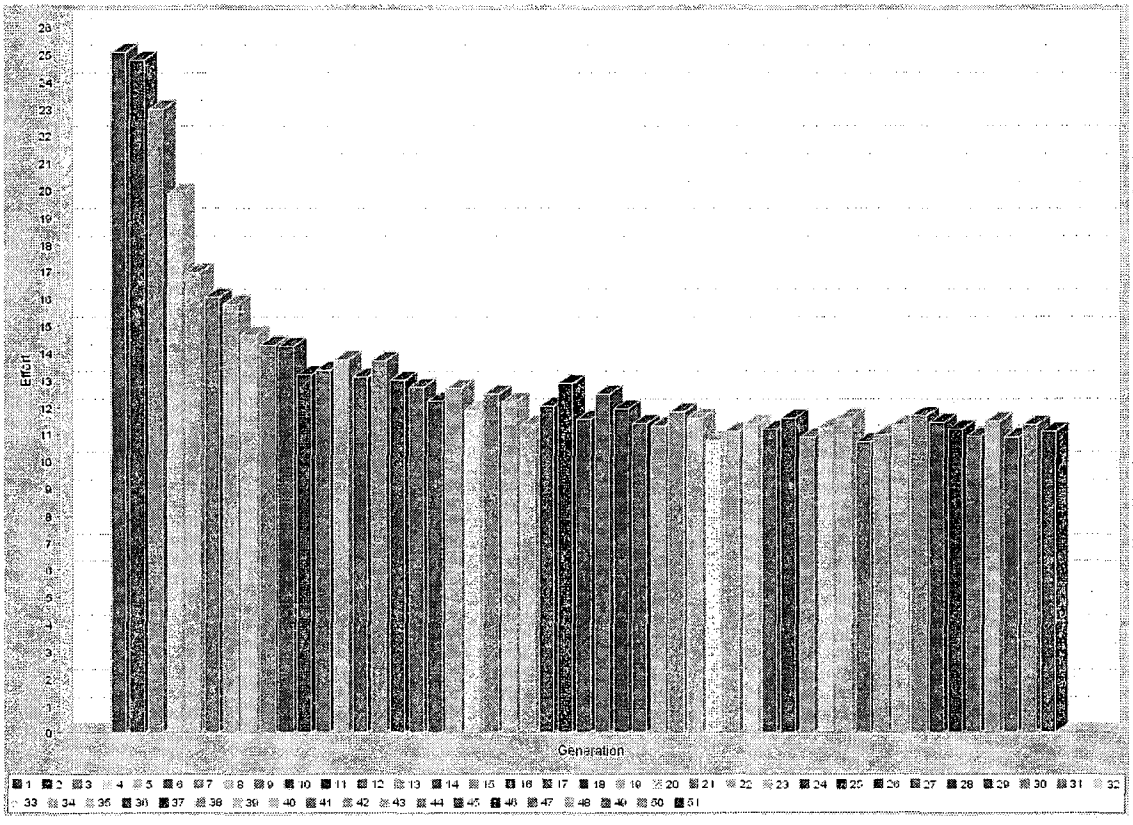


Figura 26: Esforço Médio por Geração

A seguir, na Figura 27, mostramos um gráfico do menor esforço obtido por geração. Note que a geração número 38 obteve o menor esforço: 2.5412. Após essa geração, nenhuma outra conseguiu gerar um indivíduo que obtivesse melhor desempenho. Ao fim das 13 gerações sem melhoria, na 51ª, o algoritmo parou.

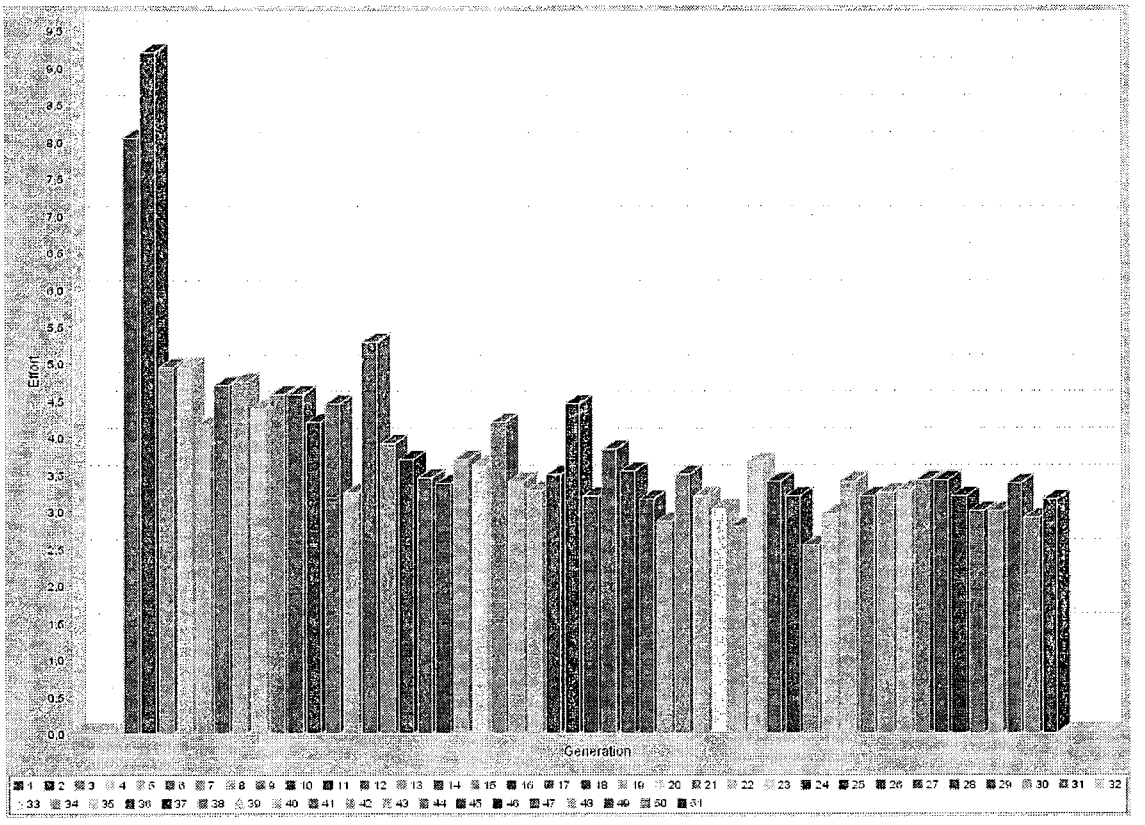


Figura 27: Esforço Mínimo por Geração

A tabela 6 mostra os pesos do melhor individuo simulado.

Tabela 6: Configuração Ótima obtida pelo Algoritmo Genético

Mw0	0.0313
Mw1	0.0
Mw2	0.9932
Rw0	0.3578
Rw1	-0.0196
Rw2	0.1359
Esforço Simulado	2.5412

Avaliando os resultados, temos que as simulações usando algoritmos genéticos avaliaram 6528 configurações e resultaram em um esforço mínimo de 2.541, 61.9% menor que o esforço da Etapa 2, cujo esforço mínimo foi de 6.67 e foram avaliadas 195 configurações. Essa ampliação de avaliações na etapa 4 foi possível graças às otimizações e refactorings realizados no código do simulador. Adicionalmente, note que com até 2½ réplicas por recurso (aproximadamente) foi possível garantir a disponibilidade de 90%.

Capítulo 5 – Conclusões e Trabalhos Futuros

Este trabalho se propôs a mostrar como tornar sistemas Peer-to-Peer (P2P) mais confiáveis, buscando os desafios comuns entre bancos de dados distribuídos e a arquitetura peer-to-peer, onde propostas de solução dos bancos de dados distribuídos podem e efetivamente são aplicadas ao P2P. Destacamos a oportunidade que é usar o JXTA que vem se consolidando como protocolo padrão para comunicação na área e o nicho em que se encaixa o COPPEER, um framework de aplicações P2P que apresenta uma série de vantagens ao desenvolvedor e conseqüentemente ao usuário final.

Uma das características as quais o COPPEER foi idealizado é um serviço que possibilite garantir os recursos da rede independentemente da volatilidade de seus peers, denominado Serviço de Garantia de Disponibilidade, o objetivo maior deste trabalho.

O modelo apresentado para o Serviço de Garantia de Disponibilidade (SGD) é baseado na redundância através da replicação de recursos de maneira coordenada, onde os peers exercem papéis distintos para efetuar um trabalho organizado e eficiente.

A eficiência do modelo proposto foi comprovada através de simulações realizadas em diferenciados cenários e técnicas de otimização. Além disso, obtivemos um resultado ótimo para a realização da replicação com o menor esforço possível, minimizando custos computacionais como o armazenamento e consumo de banda da rede.

Como trabalho futuro, é importante a implantação do serviço de garantia de disponibilidade como um plugin do microkernel do COPPEER, usufruindo de todos os resultados simulados. Pode se pensar também em uma possível integração entre o SGD e os *peer group services* do JXTA, que possuem um mecanismo de tolerância à falhas mas não minimizam o impacto da falhas no funcionamento dos serviços em grupo.

Uma primeira contribuição é que o Editor Colaborativo de Ontologias (COE) e outras aplicações P2P que venham a ser desenvolvidas poderão usufruir deste trabalho para manter seus objetos na rede com a disponibilidade que julgarem necessária.

Uma outra contribuição deixada por este trabalho é que outras pesquisas que desejem simular o comportamento de redes P2P podem aproveitar o COPS e estender suas funcionalidades.

Concluindo, o crescimento vertiginoso do número de computadores na sociedade revela a oportunidade de aproveitar esse potencial computacional para garantir a disponibilidade dos dados e quem sabe alcançar um dos objetivos mais antigos dos seres humanos: a imortalidade, porém apenas para os dados.

Referências

BECKETT, D., MCBRIDE, B., 2003, *RDF/XML Syntax Specification*, World Wide Web Consortium (W3C) Document, February.

BHAGWAN, R., SAVAGE, S., VOELKER, G., 2003, "Understanding Availability". In: *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03)*, Berkley, CA, February.

BORGHOFF, U. M., SCHLICHTER, J. H., 2000, *Computer-Supported Cooperative Work: Introduction to Distributed Applications*. 1 ed. New York, Springer-Verlag.

BRAGA, B., XEXÉO, G., DE SOUZA, J., 2004, *COPPEER: A Peer-to-Peer Application Framework*, Relatório Técnico, COPPE/UFRJ.

CLARKE, I., MILLER, S., et al., 2002, "Protecting Free Expression Online with Freenet", *IEEE Internet Computing*, v. 6, n. 1, pp. 40-49.

COHEN, E., SHENKER, S., 2002, "Replication strategies in unstructured peer-to-peer networks", In: *ACM SIGCOMM'02 Conference*, August.

COOPER, B., GARCIA-MOLINA, H., 2002, "Peer-to-peer resource trading in a reliable distributed system". In: *Proceedings of First International Workshop on Peer-to-Peer Systems*, Cambridge, MA.

DAVIDSON, S. B., GARCIA-MOLINA, H., SKEEN, D., 1985, "Consistency in Partitioned Networks". *ACM Computing Surveys*, v. 17, n.3, September.

DE JONG, K. A., 1975, *An analysis of the behavior of a class of genetic adaptive systems*. Ph.D. dissertation, The University of Michigan, Ann Arbor, MI.

DE SOUZA, J., BRAGA, B., XEXÉO, G., 2004, *COPPEER-DB: Fault-tolerant Distributed Transactions for a Peer-to-Peer Data Management System*. Relatório Técnico, COPPE/UFRJ.

DIERKS, T., ALLEN, C., 1999, *The TLS Protocol Version 1.0*. RFC 2246, Motorola and Cisco Systems, January.

EASTLAKE, D., JONES, P., 2001, *US Secure Hash Algorithm 1 (SHA1)*. RFC 3174, Motorola and Cisco Systems, September.

FRIER, A., KARLTON, P., KOCHER, P., 1996, *The SSL 3.0 Protocol*. Netscape Communications Corp. Document, November.

GRIBBLE, S., HALEVY, A., IVES, Z., RODRIG, M., SUCIU, D., 2001, "What can databases do for Peer-to-Peer". In: *Proceedings of Fourth International Workshop on Databases and the Web (WebDB 2001)*, Santa Barbara, May.

HAIFENG, Y., VAHDAT, A., 2001, "The Costs and Limits of Availability for Replicated Services". In: *Proceedings of 18th ACM Symposium on Operating Systems Principles (SOSP 01)*, Banff, Canada, October.

HALEPOVIC, E., DETERS, R., 2003, "The JXTA Performance Study". *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM 03)*, Canada.

HELAL, A., HEDDAYA, A., BHARGAVA, B., 1996, *Replication Techniques in Distributed Systems*. 1 ed., Boston, Kluwer Academic Publishers, August.

KANGASHARJU, J., ROBERTS, J., ROSS, K., 2001, "Object replication strategies in content distribution networks", In: *Proceedings of the Sixth International Workshop on Web Caching and Content Distribution (WCW 01)*.

KELLERER, W., 1998, *Dienstarchitekturen in der Telekommunikation - Evolution, Methoden und Vergleich*. Technical Report, Technische Universität München.

KENT, S., ATKINSON, R., 1998, *Security Architecture for the Internet Protocol*. RFC 2401, NCorp e @Home Network, November.

KUBIATOWICZ, J., 2003, "Extracting guarantees from chaos". *Communications of the ACM* v. 46, n. 2, pp. 33-38, February.

KUBIATOWICZ, J., BINDEL, D., CHEN, Y., CZERWINSKI, S., et al., 2000, "OceanStore: an architecture for global-scale persistent storage". *In Proc. ACM ASPLOS*, pp. 190-201, November.

NEJDL, W., WOLF, B., STAAB, S., TANE, J., 2002, "Edutella: Searching and Annotating Resources within an RDF-based P2P Network". In: *Proceedings of the Semantic Web Workshop*, Honolulu, Hawaii, May.

ORAM, A., 2001, *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*, 1 ed., O'Reilly, March.

ÖZSU, T., VALDURIEZ, P., 1999, *Principles of Distributed Databases*. 2 ed., Prentice-Hall International Inc., Englewood Cliffs, New Jersey.

PETERSON, L., ANDERSON, T., CULLER, D., ROSCOE, T., 2002, "A Blueprint for Introducing Disruptive Technology into the Internet", *Proceedings of the 1st ACM Workshop on Hot Topics in Networking (HotNets),* New Jersey, October.

RANGANATHAN, K., IAMNITCHI, A., FOSTER, I., 2002, "Improving Data Availability through Dynamic Model-Driven Replication in Large Peer-to-Peer Communities". *Global and Peer-to-Peer Computing on Large Scale Distributed Systems Workshop*, Berlin, Germany, May.

RATNASAMY, S., FRANCIS, P., HANDLEY, M., et al., 2001, "A scalable content-addressable network". In: *Proceedings of the ACM SIGCOMM*, pp. 161-172, San Diego, CA, August.

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

RIVEST, R., SHAMIR, A., ADLEMAN, L., 1978, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems", *Communications of the ACM*, v. 21, n.2, pp. 120-126, February.

ROSS, K., 2004, RUBENSTEIN, D., "P2P Systems Tutorial". *Proceedings of 23rd Conference of the IEEE Communications Society (Infocom 04)*, Hong Kong.

ROWSTRON, A., DRUSCHEL, P., 2001, "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems". *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*. Heidelberg, Germany, November.

SCHMITT, J., ON, G., STEINMETZ, R., 2003, "Quality of Availability: Replica Placement for Widely Distributed Systems", *The 11th International Workshop on Quality of Service (IWQoS 03)*, Monterey, California, July.

SCHOLLMEIER, R., 2002, "A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications", *Proceedings of the First International Conference on Peer-to-Peer Computing (P2P 02)*, pp. 101-203, IEEE, July.

SHROEDER, L., SJOQUIST, D., STEPHAN, P., 1986, *Understanding Regression Analysis – An Introductory Guide*. Sage Publications, California.

STOICA, I., MORRIS, R., et al., 2001, "Chord: A Peer-to-Peer Lookup Service for Internet Applications". *ACM SIGCOMM Conference*, San Diego, CA, September.

THOMAS, A., 1998, *Enterprise JavaBeans Technology – Server Component Model for the Java Platform*, Sun Microsystems Inc. Document, December.

TRAVERSAT, B., ABDELAZIZ, M., POUYOUL, E., 2003, *A Loosely-Consistent DHT Rendezvous Walker*, Sun Microsystems Inc. Document, March.

XEXÉO, G., DE SOUZA, J., NOGUEIRA D'ALMEIDA, J., et al., 2004, "Peer-to-Peer Collaborative Editing of Ontologies". In: *The Eighth International Conference on CSCW in Design*, Xiamen, China.

ZHAO, B., KUBIATOWICZ, J., JOSEPH, A., 2001, *Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing*, Berkeley Tech. Report, Computer Science Division, University of California, April.

Anexo A – Diagrama de Classes do Simulador

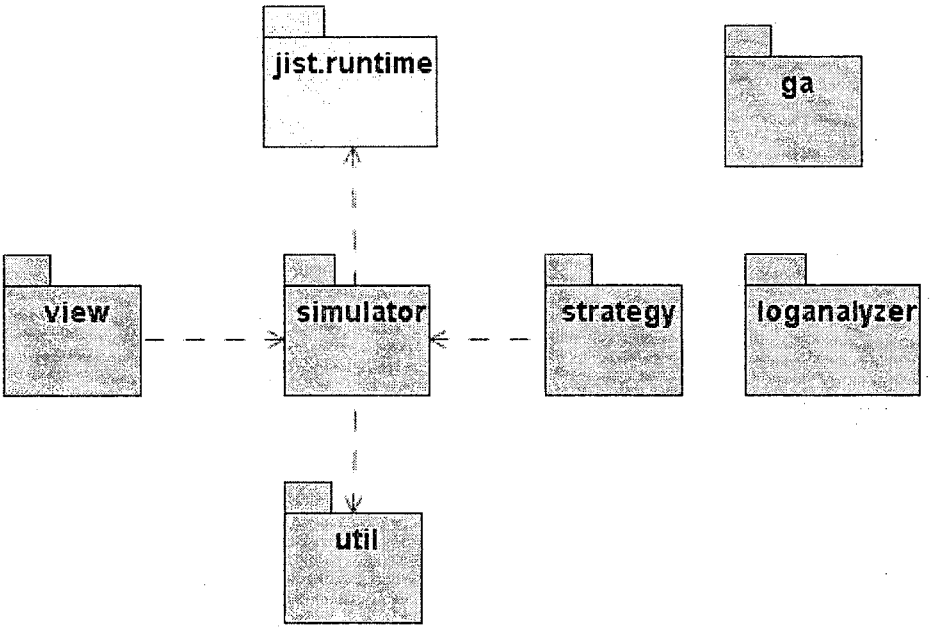


Figura 28: Diagrama de Pacotes

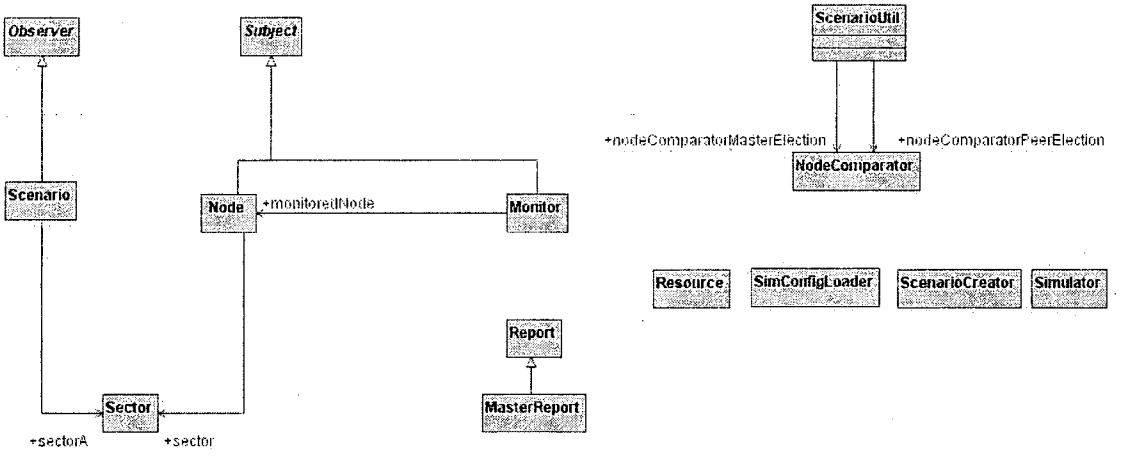


Figura 29: Pacote simulator

Figura 30: Pacote loganalyzer

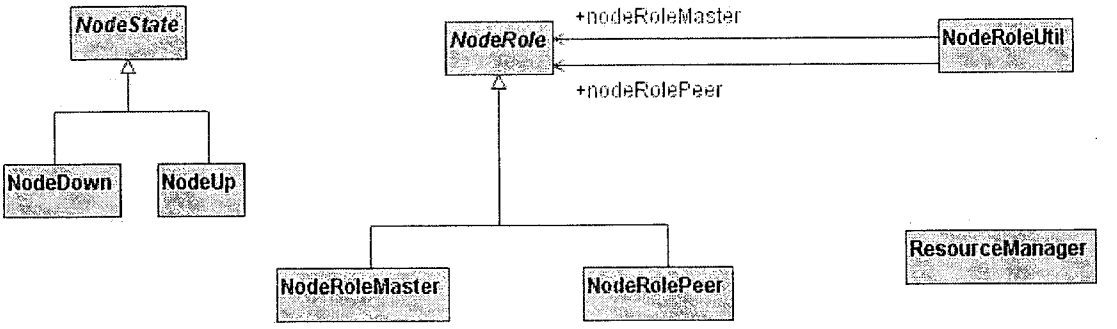


Figura 31: Pacote strategy

Figura 32: Pacote util

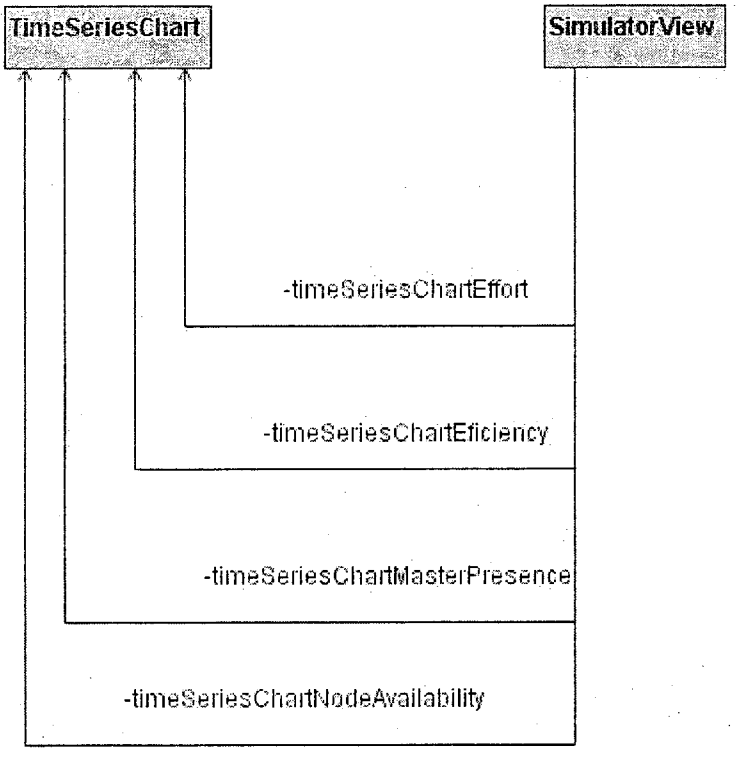


Figura 33: Pacote view

Anexo B – Resultados da Etapa 1

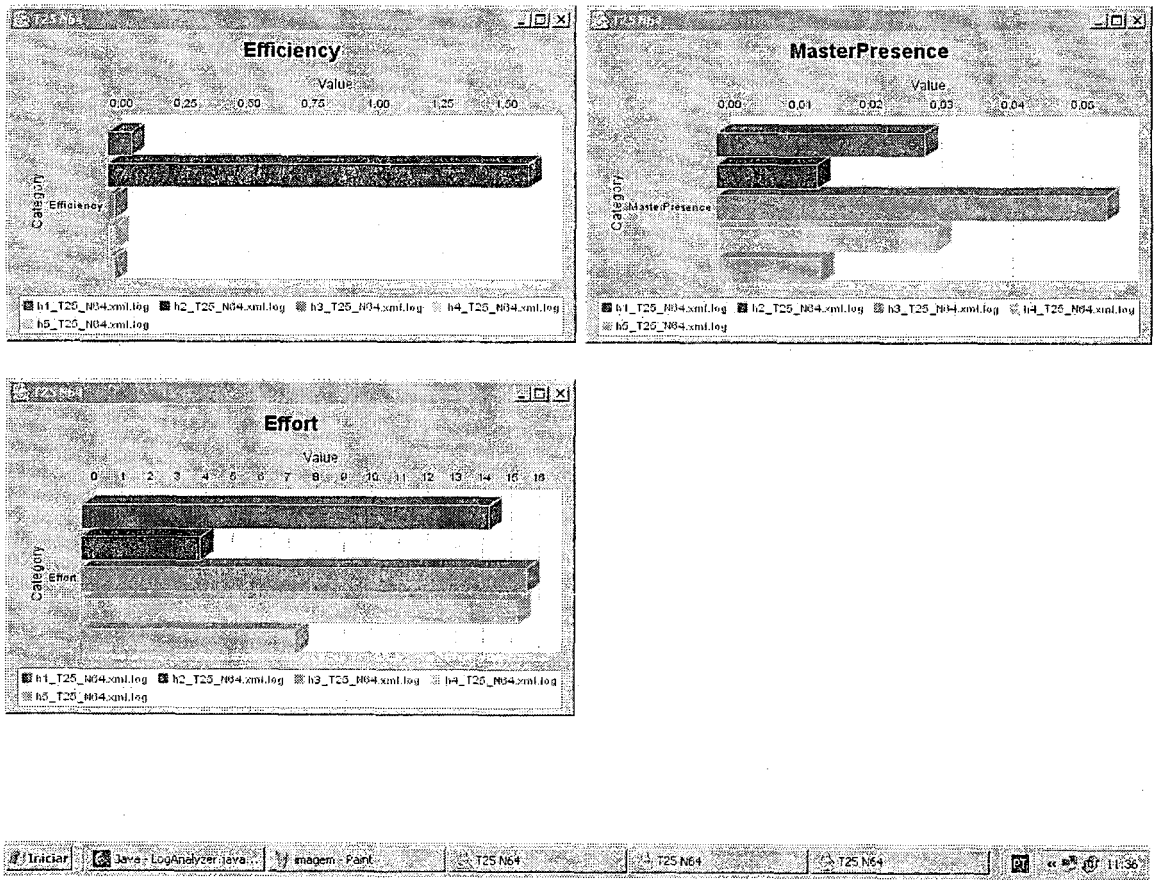


Figura 34: Rede com 64 nós e Threshold 25%

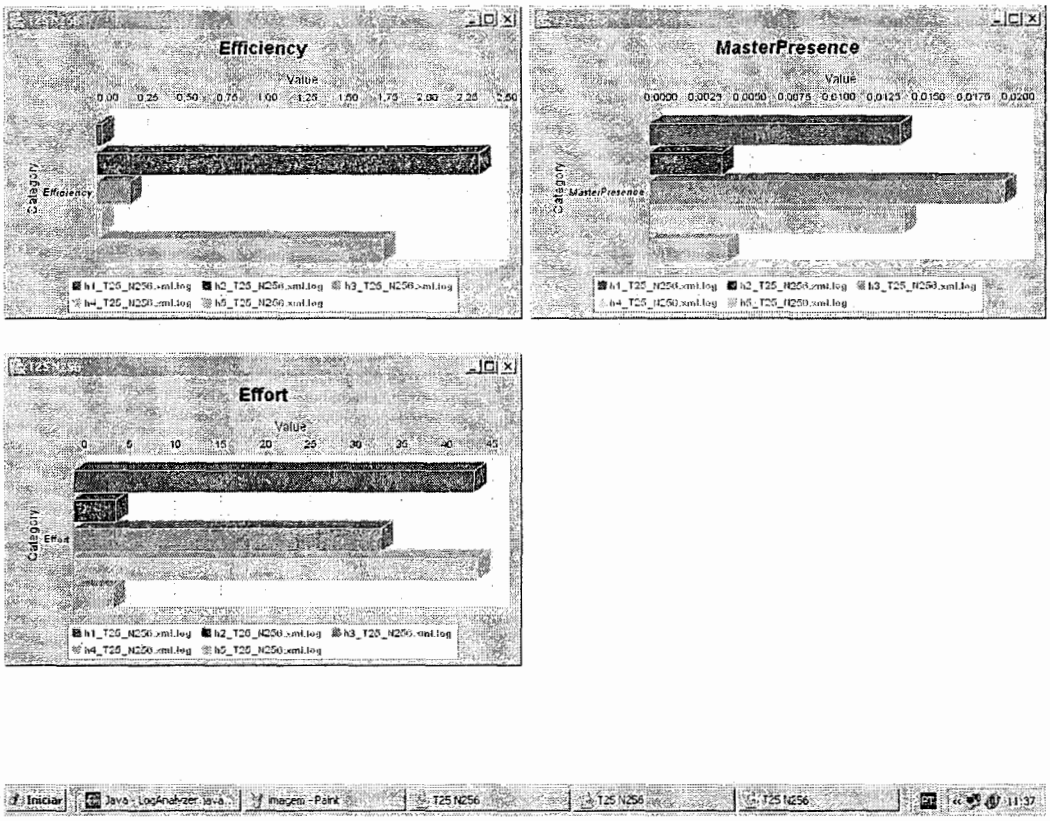


Figura 35: Rede com 256 nós e Threshold 25%

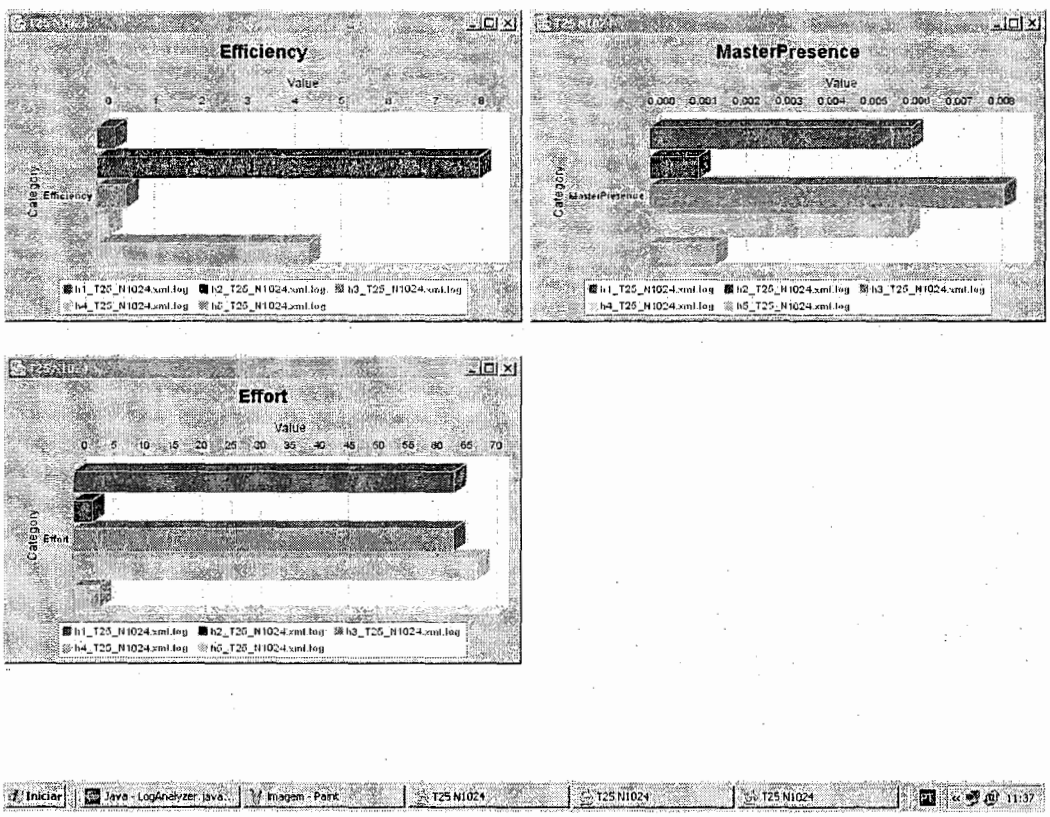


Figura 36: Rede com 1024 nós e Threshold 25%

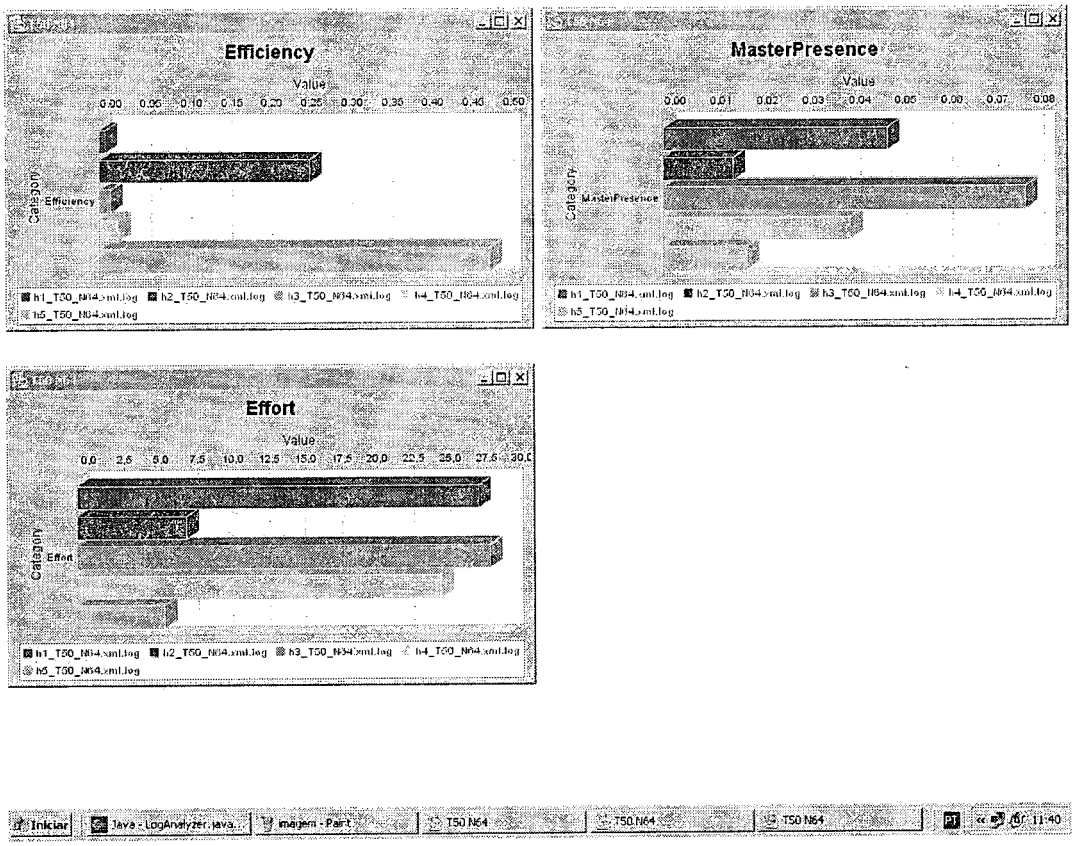


Figura 37: Rede com 64 nós e Threshold 50%

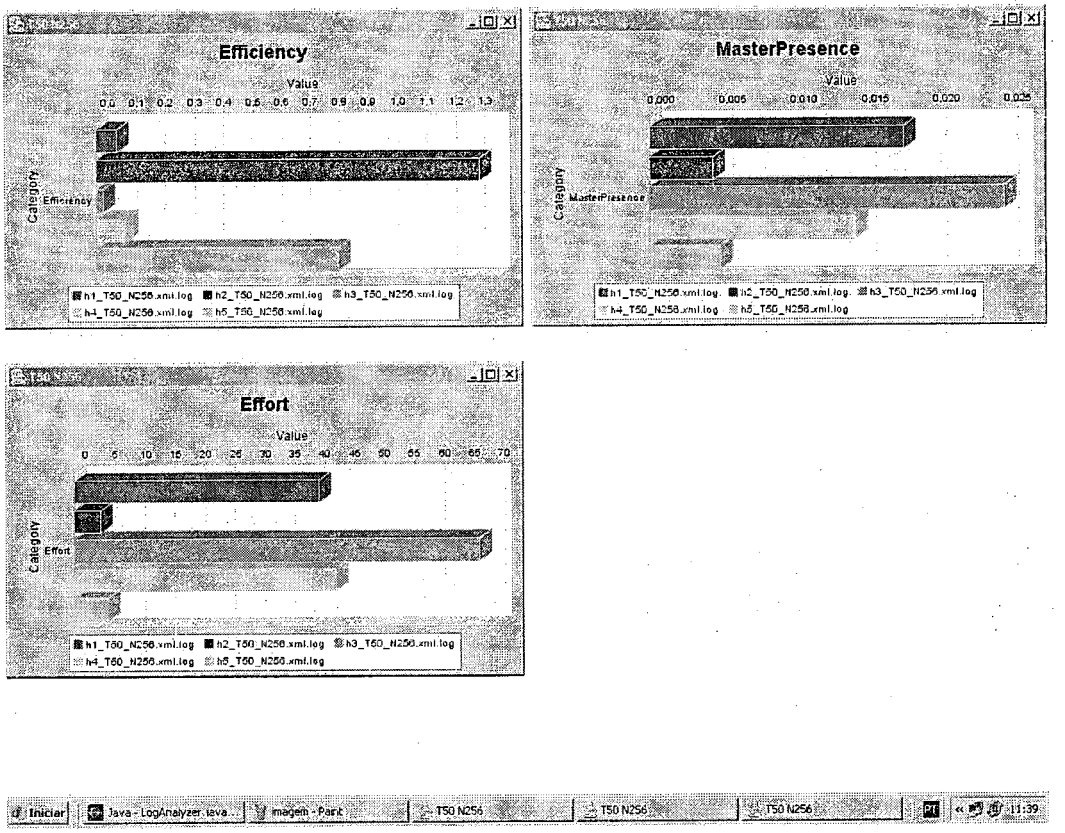


Figura 38: Rede com 256 nós e Threshold 50%

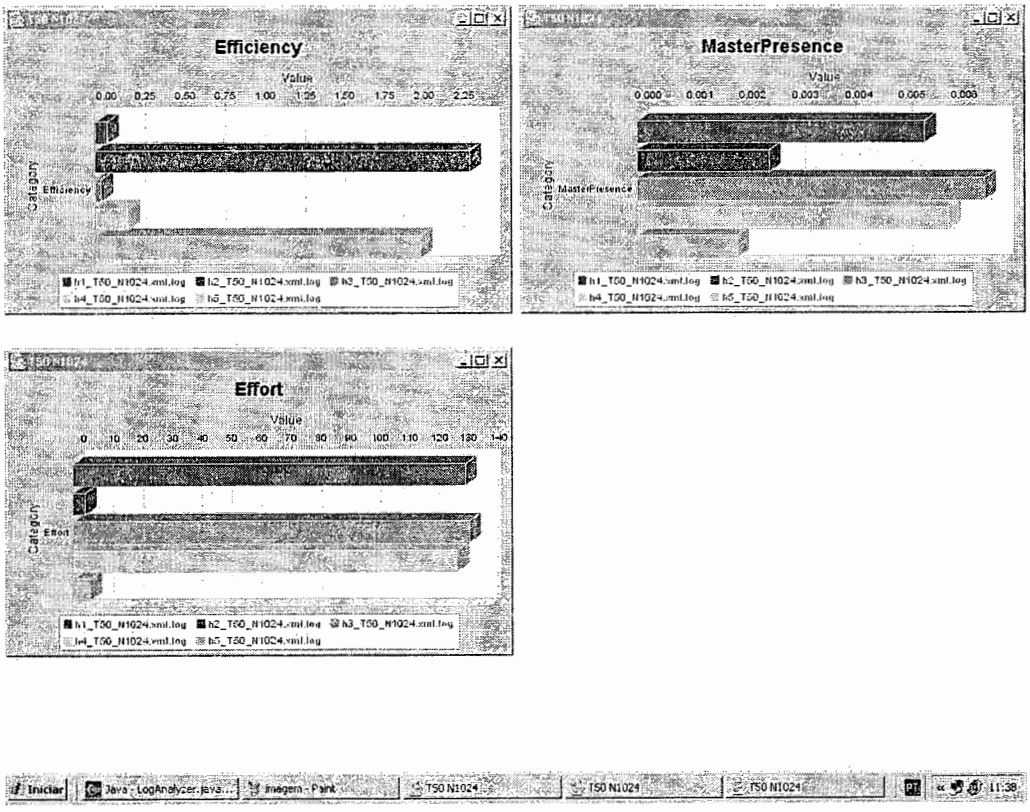


Figura 39: Rede com 1024 nós e Threshold 50%

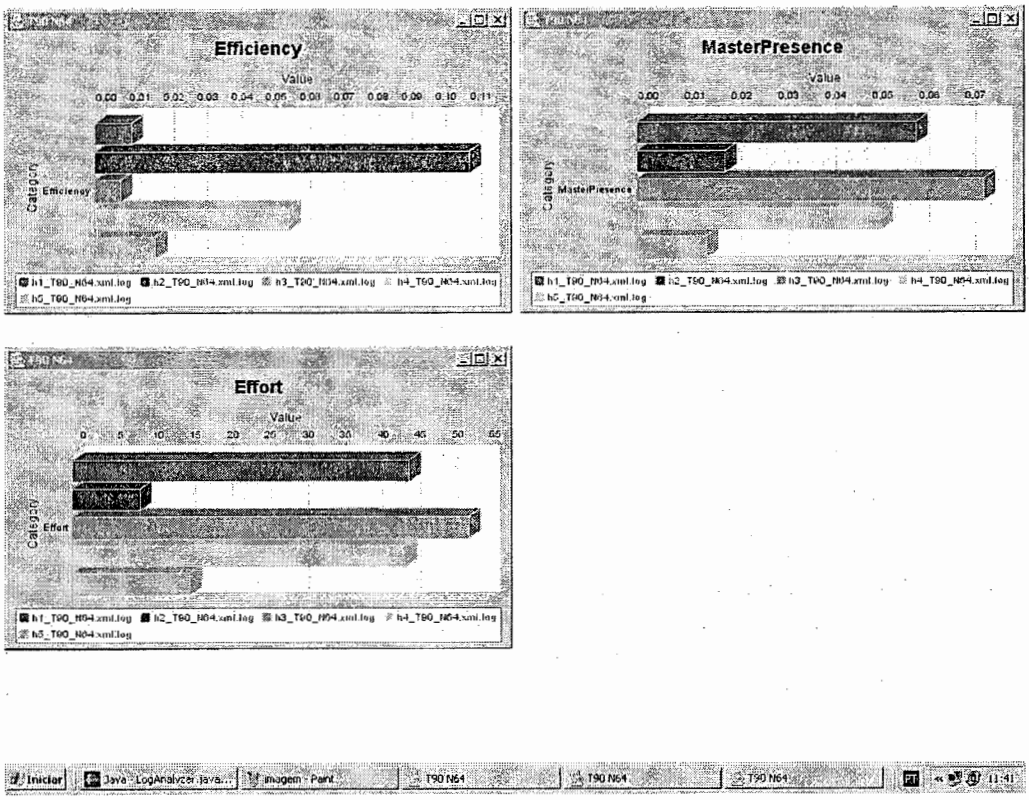


Figura 40: Rede com 64 nós e Threshold 90%

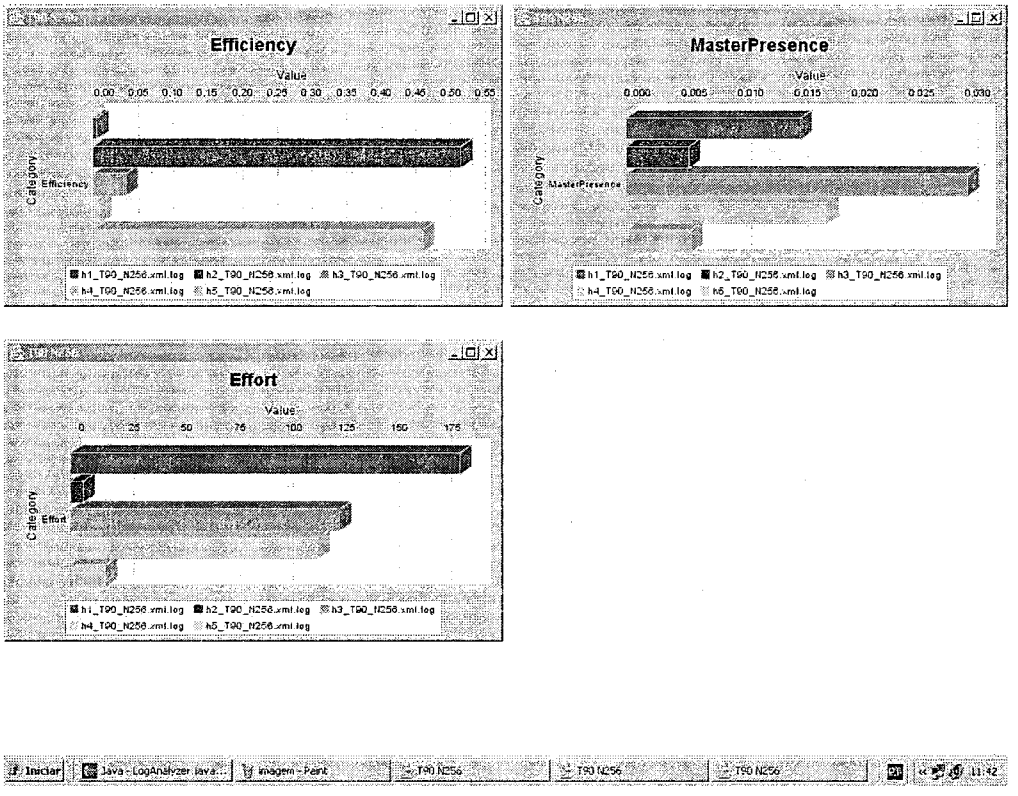


Figura 41: Rede com 256 nós e Threshold 90%

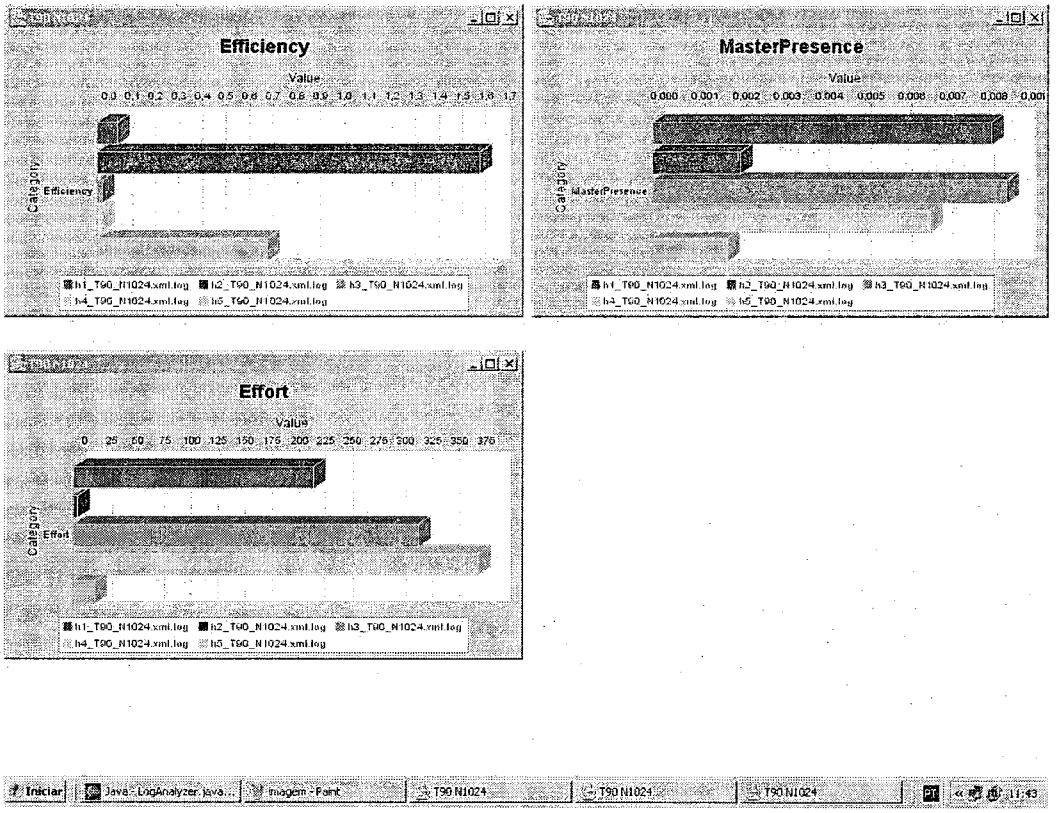


Figura 42: Rede com 1024 nós e Threshold 90%

Anexo C – Resultados da Etapa 2

Tabela 7: Média dos 3 rounds das 195 simulações

Id	mW0	mW1	mW2	rW0	rW1	rW2	eficiência	esforço	espalhamento
1	0	0	0	0,5	0	0	1,111086333	95,58553933	0,016011315
2	0	0	0	0,5	0	0,5	1,111107344	80,71438	0,012683737
3	0	0	0	0,5	-0,5	0	1,111093557	94,36056267	0,015748086
4	0	0	0	0,5	-0,5	0,5	1,111105177	82,14954267	0,013938372
5	0	0	0	1	0	0	1,111097622	88,58896933	0,01633293
6	0	0	0	1	0	0,5	1,111101211	101,5081013	0,015524935
7	0	0	0	1	0	1	1,111100063	112,5285747	0,017306758
8	0	0	0	1	-0,5	0	1,111102106	92,912028	0,015580078
9	0	0	0	1	-0,5	0,5	1,111097603	79,257418	0,014395404
10	0	0	0	1	-0,5	1	1,111091003	95,96919867	0,016611641
11	0	0	0	1	-1	0	1,111088271	72,36959867	0,011982695
12	0	0	0	1	-1	0,5	1,111106696	98,13320933	0,016954023
13	0	0	0	1	-1	1	1,111101075	94,89537133	0,016055938
14	0,5	-0,5	0	0,5	0	0,5	1,111096032	13,25466867	0,011523958
15	0,5	-0,5	0	0,5	-0,5	0	1,111095993	32,897406	0,019132552
16	0,5	-0,5	0	0,5	-0,5	0,5	1,11108867	49,72501467	0,019175677
17	0,5	-0,5	0	1	0	0	1,11110179	13,06975733	0,011939049
18	0,5	-0,5	0	1	0	0,5	1,111103212	12,26073267	0,011394935
19	0,5	-0,5	0	1	0	1	1,111102321	13,81685333	0,010457813
20	0,5	-0,5	0	1	-0,5	0	1,111108615	47,55373333	0,019160052
21	0,5	-0,5	0	1	-0,5	0,5	1,111069895	65,07764933	0,019038646
22	0,5	-0,5	0	1	-0,5	1	1,111108355	20,06735733	0,019169688
23	0,5	-0,5	0	1	-1	0	1,111104516	45,70233733	0,019148724
24	0,5	-0,5	0	1	-1	0,5	1,111091438	54,027962	0,018800391
25	0,5	-0,5	0	1	-1	1	1,111105422	43,11544733	0,019161107
26	0,5	-0,5	0	0,5	0	0	1,111101037	19,62194067	0,011343789
27	0,5	-0,5	0,5	0,5	0	0	1,111092164	14,19302867	0,011028034
28	0,5	-0,5	0,5	0,5	0	0,5	1,111105531	13,45532533	0,012009961
29	0,5	-0,5	0,5	0,5	-0,5	0	1,111032593	33,171274	0,018866927
30	0,5	-0,5	0,5	0,5	-0,5	0,5	1,111105963	22,186056	0,018682057
31	0,5	-0,5	0,5	1	0	0	1,111100333	14,76030867	0,013044049
32	0,5	-0,5	0,5	1	0	0,5	1,111087832	12,22941	0,013341992
33	0,5	-0,5	0,5	1	0	1	1,111089769	19,19654	0,013327422
34	0,5	-0,5	0,5	1	-0,5	0	1,111104451	30,317658	0,019022266
35	0,5	-0,5	0,5	1	-0,5	0,5	1,110911382	18,70267733	0,018685859
36	0,5	-0,5	0,5	1	-0,5	1	1,111099831	15,139984	0,018539961
37	0,5	-0,5	0,5	1	-1	0	1,111103863	19,342302	0,019085599
38	0,5	-0,5	0,5	1	-1	0,5	1,111086108	38,544122	0,019062969
39	0,5	-0,5	0,5	1	-1	1	1,111087834	24,694448	0,019171641
40	0,5	0	0	0,5	0	0	1,111103241	14,32108	0,004419961
41	0,5	0	0	0,5	0	0,5	1,111104804	16,31650333	0,003705286
42	0,5	0	0	0,5	-0,5	0	1,111102173	57,55164867	0,005121901
43	0,5	0	0	0,5	-0,5	0,5	1,111088775	64,51423333	0,004806289
44	0,5	0	0	1	0	0	1,111098732	8,990331333	0,003552083
45	0,5	0	0	1	0	0,5	1,11110294	9,803889333	0,003925599
46	0,5	0	0	1	0	1	1,111089361	15,386828	0,003425221

47	0,5	0	0	1	-0,5	0	1,111087534	33,06478067	0,004686836
48	0,5	0	0	1	-0,5	0,5	1,111102162	63,489246	0,005203568
49	0,5	0	0	1	-0,5	1	1,111107383	35,61642533	0,004603359
50	0,5	0	0	1	-1	0	1,111101025	34,93580533	0,004702318
51	0,5	0	0	1	-1	0,5	1,111081649	46,092648	0,004973411
52	0,5	0	0	1	-1	1	1,111105633	36,43887733	0,004969622
53	0,5	0	0,5	0,5	0	0	1,111070082	9,97712	0,002105352
54	0,5	0	0,5	0,5	0	0,5	1,111110272	12,10477733	0,003287904
55	0,5	0	0,5	0,5	-0,5	0	1,111106115	13,27989133	0,004726016
56	0,5	0	0,5	0,5	-0,5	0,5	1,111104862	11,83772333	0,004327435
57	0,5	0	0,5	1	0	0	1,111053857	9,553258	0,003561797
58	0,5	0	0,5	1	0	0,5	1,111105963	10,92028267	0,003755273
59	0,5	0	0,5	1	0	1	1,111104964	9,494746	0,002454115
60	0,5	0	0,5	1	-0,5	0	1,111100799	10,63078867	0,004645495
61	0,5	0	0,5	1	-0,5	0,5	1,111073256	10,345934	0,004703138
62	0,5	0	0,5	1	-0,5	1	1,111102091	12,16355133	0,004579049
63	0,5	0	0,5	1	-1	0	1,111096044	13,23985667	0,004536042
64	0,5	0	0,5	1	-1	0,5	1,111039694	14,33601333	0,00449224
65	0,5	0	0,5	1	-1	1	1,111108511	12,93631333	0,004797747
66	1	-0,5	0	0,5	0	0	1,111104571	11,56411867	0,011812031
67	1	-0,5	0	0,5	0	0,5	1,111102284	13,42144667	0,011102253
68	1	-0,5	0	0,5	-0,5	0	1,111102884	34,994872	0,019182721
69	1	-0,5	0	0,5	-0,5	0,5	1,111088874	59,75408467	0,01856418
70	1	-0,5	0	1	0	0	1,111069277	12,50265	0,007707448
71	1	-0,5	0	1	0	0,5	1,111105306	19,99252467	0,00865263
72	1	-0,5	0	1	0	1	1,11108982	13,3453	0,012261393
73	1	-0,5	0	1	-0,5	0	1,111096832	40,91163933	0,019051302
74	1	-0,5	0	1	-0,5	0,5	1,111090568	32,678862	0,019217695
75	1	-0,5	0	1	-0,5	1	1,111103973	29,34603733	0,018967305
76	1	-0,5	0	1	-1	0	1,11108539	53,008278	0,019052409
77	1	-0,5	0	1	-1	0,5	1,111101945	45,87248333	0,019073112
78	1	-0,5	0	1	-1	1	1,111105946	22,99120133	0,019134674
79	1	-0,5	0,5	0,5	0	0,5	1,111096723	10,97796133	0,004678633
80	1	-0,5	0,5	0,5	0	0	1,111047674	13,02849533	0,012679219
81	1	-0,5	0,5	0,5	-0,5	0	1,111066245	32,06277867	0,018528333
82	1	-0,5	0,5	0,5	-0,5	0,5	1,111040004	24,445852	0,018792135
83	1	-0,5	0,5	1	0	0	1,110944287	13,78311933	0,013120964
84	1	-0,5	0,5	1	0	0,5	1,111075647	13,72479333	0,012727057
85	1	-0,5	0,5	1	0	1	1,111059199	11,51984667	0,007333776
86	1	-0,5	0,5	1	-0,5	0	1,111109039	17,31322333	0,01846888
87	1	-0,5	0,5	1	-0,5	0,5	1,111103454	26,41568333	0,019036406
88	1	-0,5	0,5	1	-0,5	1	1,110787201	27,50528733	0,018377852
89	1	-0,5	0,5	1	-1	0	1,111074406	17,948374	0,01892082
90	1	-0,5	0,5	1	-1	0,5	1,111059403	22,16497333	0,018514466
91	1	-0,5	0,5	1	-1	1	1,111102698	35,41902867	0,018365286
92	1	-0,5	1	0,5	0	0	1,111071343	15,10501133	0,009053125
93	1	-0,5	1	0,5	0	0,5	1,111097632	13,00856733	0,007807773
94	1	-0,5	1	0,5	-0,5	0	1,111098555	27,439354	0,018878268
95	1	-0,5	1	0,5	-0,5	0,5	1,11107686	25,178756	0,019080352
96	1	-0,5	1	1	0	0	1,111086478	14,14515533	0,011211536
97	1	-0,5	1	1	0	0,5	1,111076265	12,686194	0,009044935
98	1	-0,5	1	1	0	1	1,111099754	13,67720267	0,011733568
99	1	-0,5	1	1	-0,5	0	1,111107301	21,12840933	0,018681367
100	1	-0,5	1	1	-0,5	0,5	1,111098679	21,08880333	0,018860404

101	1	-0,5	1	1	-0,5	1	1,111073775	15,30583467	0,019000091
102	1	-0,5	1	1	-1	0	1,111093516	22,16008467	0,018686888
103	1	-0,5	1	1	-1	0,5	1,111094587	24,393256	0,019169102
104	1	-0,5	1	1	-1	1	1,111074264	22,38691	0,01899875
105	1	-1	0	0,5	0	0	1,111102722	13,40660933	0,007614492
106	1	-1	0	0,5	0	0,5	1,111067921	19,897484	0,009275339
107	1	-1	0	0,5	-0,5	0	1,111089372	55,357692	0,019161302
108	1	-1	0	0,5	-0,5	0,5	1,111085282	51,21555333	0,019240768
109	1	-1	0	1	0	0	1,111102452	12,10655533	0,00990849
110	1	-1	0	1	0	0,5	1,111094028	13,95864333	0,007535013
111	1	-1	0	1	-0,5	0	1,111085417	60,26029133	0,019242904
112	1	-1	0	1	-0,5	0,5	1,111101373	28,20919133	0,019256771
113	1	-1	0	1	-0,5	1	1,111086238	43,61073067	0,019040938
114	1	-1	0	1	-1	0	1,111105782	55,341606	0,019161966
115	1	-1	0	1	-1	0,5	1,111085688	36,961542	0,01924444
116	1	-1	0	1	-1	1	1,111046314	67,566436	0,019102917
117	1	-1	0,5	0,5	0	0	1,111088205	13,74052867	0,012651029
118	1	-1	0,5	0,5	0	0,5	1,11110552	12,156716	0,008079401
119	1	-1	0,5	0,5	-0,5	0	1,111093808	20,08655333	0,018643281
120	1	-1	0,5	0,5	-0,5	0,5	1,110994624	28,12478867	0,018792682
121	1	-1	0,5	1	0	0	1,111106853	14,25402267	0,007066549
122	1	-1	0,5	1	0	0,5	1,111059919	13,08078533	0,013314857
123	1	-1	0,5	1	0	1	1,111089008	12,68948133	0,008525716
124	1	-1	0,5	1	-0,5	0	1,111022725	17,04326133	0,019146354
125	1	-1	0,5	1	-0,5	0,5	1,111108261	30,52765733	0,019223763
126	1	-1	0,5	1	-0,5	1	1,111100011	14,36201133	0,018780846
127	1	-1	0,5	1	-1	0	1,110898516	35,711126	0,018969362
128	1	-1	0,5	1	-1	0,5	1,111108289	22,53350933	0,018926549
129	1	-1	0,5	1	-1	1	1,111060353	29,49442133	0,01847931
130	1	-1	1	0,5	0	0	1,111106344	14,04404933	0,013275065
131	1	-1	1	0,5	0	0,5	1,110926946	12,4905	0,011081211
132	1	-1	1	0,5	-0,5	0	1,110676355	17,711688	0,018767292
133	1	-1	1	0,5	-0,5	0,5	1,11088462	20,96616333	0,01857737
134	1	-1	1	1	0	0	1,111014985	14,17151333	0,010906563
135	1	-1	1	1	0	0,5	1,111100013	12,21249533	0,010009427
136	1	-1	1	1	0	1	1,111088373	12,86004267	0,013431732
137	1	-1	1	1	-0,5	0	1,111097293	17,37002133	0,01883013
138	1	-1	1	1	-0,5	0,5	1,111091905	16,618152	0,019009648
139	1	-1	1	1	-0,5	1	1,111061206	15,00304467	0,019100299
140	1	-1	1	1	-1	0	1,111099745	26,15769933	0,019128477
141	1	-1	1	1	-1	0,5	1,111099123	25,736832	0,019067552
142	1	-1	1	1	-1	1	1,111104242	23,02013933	0,019166146
143	1	0	0	0,5	0	0	1,111099935	22,70293267	0,003635443
144	1	0	0	0,5	0	0,5	1,111109184	20,792314	0,003126484
145	1	0	0	0,5	-0,5	0	1,111104651	48,43567933	0,004845625
146	1	0	0	0,5	-0,5	0,5	1,111105358	47,351188	0,004274688
147	1	0	0	1	0	0	1,111087841	14,14082467	0,004247656
148	1	0	0	1	0	0,5	1,111102956	14,502462	0,003781966
149	1	0	0	1	0	1	1,11110164	27,29901333	0,004295247
150	1	0	0	1	-0,5	0	1,11110266	61,648422	0,004830677
151	1	0	0	1	-0,5	0,5	1,111104108	52,475196	0,005398672
152	1	0	0	1	-0,5	1	1,111087187	57,389382	0,004363841
153	1	0	0	1	-1	0	1,111090764	45,93405933	0,004696914
154	1	0	0	1	-1	0,5	1,111100387	33,40899133	0,004430091

155	1	0	0	1	-1	1	1,111104636	59,315242	0,004790625
156	1	0	0,5	0,5	0	0	1,111108128	9,095101333	0,003933958
157	1	0	0,5	0,5	0	0,5	1,111109165	8,808103333	0,001367578
158	1	0	0,5	0,5	-0,5	0	1,111109256	9,312452	0,004424779
159	1	0	0,5	0,5	-0,5	0,5	1,11109752	10,58435733	0,004265404
160	1	0	0,5	1	0	0	1,111087838	9,444492	0,003213659
161	1	0	0,5	1	0	0,5	1,111094467	8,742668	0,00314931
162	1	0	0,5	1	0	1	1,111107941	9,381615333	0,003310065
163	1	0	0,5	1	-0,5	0	1,111098699	13,02489267	0,004690977
164	1	0	0,5	1	-0,5	0,5	1,111084139	18,01087267	0,004281081
165	1	0	0,5	1	-0,5	1	1,111052604	13,42613867	0,004339935
166	1	0	0,5	1	-1	0	1,111096605	10,606236	0,004278151
167	1	0	0,5	1	-1	0,5	1,111102563	17,206218	0,004392526
168	1	0	0,5	1	-1	1	1,111064765	24,35934133	0,00447237
169	1	0	1	0,5	0	0	1,111089426	7,933258667	0,00347681
170	1	0	1	0,5	0	0,5	1,111103031	14,399898	0,003100951
171	1	0	1	0,5	-0,5	0	1,111106118	14,62860067	0,00449819
172	1	0	1	0,5	-0,5	0,5	1,111107567	26,86945467	0,004283659
173	1	0	1	1	0	0	1,111102712	8,469542667	0,003413268
174	1	0	1	1	0	0,5	1,111105961	9,188700667	0,003178086
175	1	0	1	1	0	1	1,111108422	6,67007	0,002665352
176	1	0	1	1	-0,5	0	1,111105203	10,5798	0,004669141
177	1	0	1	1	-0,5	0,5	1,111104756	10,002994	0,004304466
178	1	0	1	1	-0,5	1	1,111110282	8,707272	0,004649063
179	1	0	1	1	-1	0	1,111094848	11,886568	0,004994766
180	1	0	1	1	-1	0,5	1,111072803	14,18098467	0,004395339
181	1	0	1	1	-1	1	1,11109413	13,975114	0,004632773
182	0,5	0	0	0	0	0	1,111105782	42,48345	0,019166146
183	0,5	0	0,5	0	0	0	1,111085688	31,8865	0,003635443
184	0,5	-0,5	0	0	0	0	1,111046314	44,77785	0,003126484
185	0,5	-0,5	0,5	0	0	0	1,111088205	34,1809	0,004845625
186	1	0	0	0	0	0	1,11110552	26,1047	0,004274688
187	1	0	0,5	0	0	0	1,111093808	23,02013933	0,004247656
188	1	0	1	0	0	0	1,110994624	22,70293267	0,003781966
189	1	-0,5	0	0	0	0	1,111106853	20,792314	0,004295247
190	1	-0,5	0,5	0	0	0	1,111059919	48,43567933	0,004830677
191	1	-0,5	1	0	0	0	1,111089008	47,351188	0,005398672
192	1	-1	0	0	0	0	1,111022725	14,14082467	0,004363841
193	1	-1	0	1	0	1	1,111108261	14,502462	0,004696914
194	1	-1	0,5	0	0	0	1,111100011	27,29901333	0,004430091
195	1	-1	1	0	0	0	1,110898516	61,648422	0,004790625

Anexo D – Resultados da Etapa 3

Tabela 8: Análise de Regressão Linear Múltipla da Eficiência

Test	**Linear regression**		
Fit	efficiency v mW0, mW1, mW2, rW0, rW1, rW2		
Performed by	José Nogueira	**Date**	2 abril 2005

n	181
R^2	0,11
Adjusted R^2	0,08
SE	0,0000

Term	Coefficient	SE	p	95% CI of Coefficient
Intercept	1,1111	0,0000	<0.0001	1,1111 to 1,1111
mW0	0,0000	0,0000	0,6664	0,0000 to 0,0000
mW1	0,0000	0,0000	0,0010	0,0000 to 0,0001
mW2	0,0000	0,0000	0,0264	0,0000 to -0,0000
rW0	0,0000	0,0000	0,3012	0,0000 to 0,0001
rW1	0,0000	0,0000	0,4376	0,0000 to 0,0000
rW2	0,0000	0,0000	0,8971	0,0000 to 0,0000

Source of variation	SSq	DF	MSq	F	p
Due to regression	0,000	6	0,000	3,63	0,0020
About regression	0,000	174	0,000		
Total	0,000	180			

Tabela 9: Análise de Regressão Linear Múltipla do Esforço de Replicação

Test	**Linear regression**		
Fit	effort v mW0, mW1, mW2, rW0, rW1, rW2		
Performed by	José Nogueira	**Date**	2 abril 2005

n	181
R^2	0,51
Adjusted R^2	0,49
SE	16,4198

Term	Coefficient	SE	p	95% CI of Coefficient
Intercept	58,8622	5,5605	<0.0001	47,8876 to 69,8369
mW0	-32,7575	4,6097	<0.0001	-41,8556 to -23,6594
mW1	-4,5888	3,4624	0,1868	-11,4226 to 2,2450
mW2	-21,1939	3,4624	<0.0001	-28,0277 to -14,3601
rW0	-7,2973	5,7785	0,2083	-18,7022 to 4,1077
rW1	-18,2387	3,3351	<0.0001	-24,8212 to -11,6562
rW2	0,8322	3,3351	0,8032	-5,7503 to 7,4147

Source of variation	SSq	DF	MSq	F	p
Due to regression	48639,666	6	8106,611	30,07	<0.0001
About regression	46912,291	174	269,611		
Total	95551,957	180			

Tabela 10: Análise de Regressão Linear Múltipla do Espalhamento

Test	**Linear regression**		
Fit	masterPresence v mW0, mW1, mW2, rW0, rW1, rW2		
Performed by	José Nogueira	**Date**	2 abril 2005

n	181
R^2	0,66
Adjusted R^2	0,65
SE	0,0039

Term	Coefficient	SE	p	95% CI of Coefficient
Intercept	0,0102	0,0013	<0.0001	0,0076 to 0,0128
mW0	-0,0075	0,0011	<0.0001	-0,0096 to -0,0053
mW1	-0,0134	0,0008	<0.0001	-0,0151 to -0,0118
mW2	0,0000	0,0008	0,9577	-0,0016 to 0,0016
rW0	-0,0005	0,0014	0,7212	-0,0032 to 0,0022
rW1	-0,0057	0,0008	<0.0001	-0,0073 to -0,0042
rW2	0,0000	0,0008	0,9658	-0,0016 to 0,0015

Source of variation	SSq	DF	MSq	F	p
Due to regression	0,005	6	0,001	55,78	<0.0001
About regression	0,003	174	0,000		
Total	0,008	180			

Anexo E – Resultados da Etapa 4

Devido ao grande número de configurações geradas, incluímos as 50 melhores durante a etapa dos algoritmos genéticos.

mW0	mW1	mW2	rW0	rW1	rW2	Efficiency	Effort	MasterPresence
0.0313	0.0	0.9932	0.3578	-0.0196	0.1359	1.111	2.541	0.026
0.0587	0.0	0.6628	0.9717	-0.0538	0.1457	1.111	2.790	0.026
0.7058	0.0	0.9971	0.4233	-0.0108	0.7674	1.111	2.874	0.031
0.0645	0.0	0.9707	0.522	-0.0538	0.3969	1.111	2.915	0.031
0.6197	0.0	0.6628	0.8583	-0.0196	0.1906	1.111	2.957	0.020
0.783	0.0	0.6843	0.7361	-0.0235	0.6882	1.111	2.999	0.020
0.0753	0.0	0.9589	0.7889	-0.0059	0.3891	1.110	2.999	0.026
0.6022	0.0	0.7234	0.5142	-0.1848	0.4018	1.111	2.999	0.015
0.6178	0.0	0.9697	0.5064	-0.1427	0.1281	1.111	3.040	0.020
0.8143	0.0	0.4282	0.4282	-0.0059	0.132	1.111	3.040	0.015
0.783	0.0	0.1457	0.7879	-0.0235	0.2063	1.111	3.040	0.020
0.8974	0.0	0.9717	0.7234	-0.0587	0.2395	1.111	3.040	0.020
0.958	0.0	0.6588	0.7283	-0.0528	0.8309	1.111	3.040	0.020
0.7175	0.0	0.9756	0.6989	-0.0352	0.089	1.111	3.165	0.020
0.0753	0.0	0.6843	0.7928	-0.0538	0.3842	1.111	3.165	0.015
0.7273	0.0	0.741	0.3969	-0.5494	0.4878	1.111	3.165	0.020
0.7107	0.0	0.7097	0.7537	-0.0020	0.6403	1.110	3.206	0.026
0.7175	0.0	0.9756	0.6344	-0.0274	0.4018	1.111	3.206	0.026
0.87	0.0	0.6852	0.9717	-0.0108	0.7674	1.111	3.207	0.031
0.7322	0.0	0.9277	0.4282	-0.0059	0.132	1.111	3.207	0.015
0.6022	0.0	0.1584	0.7273	-0.0538	0.4115	1.111	3.207	0.020
0.9971	0.0	0.9795	0.4702	-0.0587	0.4321	1.111	3.207	0.015
0.6149	0.0	0.9795	0.4536	-0.5279	0.4018	1.111	3.207	0.015
0.87	0.0	0.6628	0.4712	-0.1281	0.4057	1.111	3.248	0.020
0.8104	0.0	0.9971	0.48	-0.0587	0.4018	1.111	3.248	0.015
0.5621	0.0	0.6491	0.2669	-0.0274	0.0753	1.111	3.248	0.015
0.8456	0.0	0.3978	0.1017	-0.0039	0.1398	1.111	3.288	0.026
0.6364	-0.0020	0.9365	0.7654	-0.0587	0.0323	1.111	3.289	0.026
0.783	0.0	0.7097	0.9531	-0.0499	0.1417	1.111	3.289	0.020
0.6022	0.0	0.3969	0.2884	-0.0205	0.2063	1.110	3.290	0.015
0.7322	0.0	0.9756	0.7195	-0.0059	0.132	1.111	3.290	0.015
0.0313	0.0	0.7449	0.4233	-0.0528	0.4272	1.111	3.290	0.031
0.7322	0.0	0.9306	0.7527	-0.2571	0.393	1.111	3.290	0.026
0.5709	0.0	0.6921	0.7234	-0.0303	0.6325	1.111	3.331	0.015
0.6178	0.0	0.1486	0.2278	-0.045	0.4008	1.110	3.331	0.015
0.9374	0.0	0.6413	0.9795	-0.0098	0.1398	1.111	3.332	0.031
0.5934	0.0	0.6921	0.3539	-0.0196	0.2053	1.110	3.372	0.015
0.783	0.0	0.6843	0.7361	-0.0196	0.1398	1.110	3.373	0.026
0.6256	0.0	0.9785	0.9844	-0.0528	0.3959	1.110	3.373	0.015
0.6237	0.0	0.9756	0.4282	-0.0059	0.1222	1.110	3.373	0.026
0.7195	0.0	0.9932	0.7674	-0.0489	0.1398	1.111	3.373	0.015
0.8974	0.0	0.9804	0.4536	-0.0469	0.6227	1.111	3.414	0.015
0.87	0.0	0.6852	0.9863	-0.5318	0.1281	1.111	3.414	0.015

0.9971	0.0	0.7234	0.5142	-0.1838	0.4018	1.111	3.414	0.015
0.0538	0.0	0.1486	0.1017	-0.0059	0.13	1.111	3.414	0.015
0.8368	0.0	0.9717	0.7312	-0.0059	0.4966	1.111	3.414	0.015
0.6178	0.0	0.1486	0.0958	-0.0020	0.6139	1.111	3.414	0.020
0.87	0.0	0.6843	0.8456	-0.0039	0.1281	1.111	3.415	0.031
0.782	0.0	0.1584	0.7302	-0.0538	0.3812	1.111	3.415	0.020
0.87	0.0	0.6843	0.7361	-0.0196	0.3803	1.111	3.415	0.020