UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
INSTITUTO DE MATEMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Um Modelo de Replicação em Ambientes P2P com Mobilidade

Dissertação de Mestrado

Izalmo Primo da Silva

Orientadores:
Profa. Vanessa Braganholo Murta
Prof. Carlo Emmanoel Tolla de Oliveira

Rio de Janeiro
2008

Izalmo Primo da Silva

UM MODELO DE REPLICAÇÃO EM AMBIENTES P2P COM MOBILIDADE

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Informática

Orientadores:

Profª. Vanessa Braganholo Murta
D.Sc. , Universidade do Rio Grande do Sul, 2004

Prof. Carlo Emmanoel Tolla de Oliveira
Ph.D., University College, 1993

Rio de Janeiro

2008

FICHA CATALOGRÁFICA

Silva, Izalmo Primo da

Um Modelo de Replicação em Ambientes P2P com Mobilidade/ Izalmo Primo da Silva. – Rio de Janeiro, 2008.

xi, 84 p.; il.

Dissertação (Mestrado em Informática) – Universidade Federal do Rio de Janeiro - UFRJ, Programa de Pós-Graduação em Informática, 2008.

Orientador: Vanessa Braganholo Murta
Orientador: Carlo Emmanoel Tolla de Oliveira

1. Sistemas Distribuídos. 2. Replicação de Dados. I. Braganholo, Vanessa (Orientador), Oliveira, Carlo Emmanoel Tolla de (Orientador), II. Universidade Federal do Rio de Janeiro. Programa de Pós-Graduação em Informática. III. Título.

FOLHA DE APROVAÇÃO

Izalmo Primo da Silva

UM MODELO DE REPLICAÇÃO EM AMBIENTES P2P COM MOBILIDADE

Rio de Janeiro, 06 de Agosto de 2008

_____

Profª. Vanessa Braganholo Murta, D.Sc., PPGI/UFRJ
Orientador.

_____

Prof. Carlo Emmanoel Tolla de Oliveira, Ph.D., PPGI/UFRJ
Orientador

_____

Prof. Sergio Lifschitz, D.Sc, PUC-Rio

_____

Maria Luiza Machado Campos, Ph.D, PPGI/UFRJ

Para minha mãe...

AGRADECIMENTOS

Muitas são as pessoas a quem devo esse momento de conclusão desse trabalho. Acredito que Agradecer é uma virtude. E mesmo correndo o risco de esquecer alguém, não posso deixar de citar alguns nomes e expressar minha eterna gratidão.

A Deus, o autor e consumador da minha fé! A Ele realmente seja a glória! Só nós sabemos como foi difícil... Muito obrigado não só pela força para chegar até o fim, pela sabedoria para as escolhas certas, pelo auxílio das pessoas que você colocou no meu caminho para me ajudar, mas principalmente por estar ao meu lado. Isso foi o que mais me motivou a ir até o fim. Obrigado.

A minha mãe, que foi a grande auxiliadora desse projeto. Você me dá todo suporte emocional e físico necessário para que eu possa atingir meus objetivos. Você é meu exemplo de garra e de que devemos lutar pelo que queremos. Eu te amo e te dedico essa vitória!

A minha esposa, que me entende, batalha comigo e é participante das minhas vitórias. Você soube me entender nos momentos de dificuldade e sempre me deu força para chegar até o fim. Deus me deu uma companheira. Te amo e muito obrigado!

Ao meu orientador Prof. Carlo Emmanoel, que auxiliou-me no desenvolvimento desse trabalho. Muito obrigado pela sua amizade, por acreditar em mim, por me ajudar e principalmente por ter paciência nos momento em que precisava de ajuda. Termino com a sensação de que ganhei mais do que um mentor... Ganhei um grande e bom amigo.

A minha orientadora Profa. Vanessa Braganholo, nesses últimos meses sua participação foi fundamental para conclusão desse trabalho. Acredito que sem seu apoio ele não teria se concretizado. Obrigado por suas idéias, pelo apoio e pela paciência comigo. Mais do que um auxilio de professora ganhai o apoio de uma amiga.

Aos amigos conquistados durante o mestrado e que facilitaram minha estada no rio e me auxiliaram durante o curso. A Gisele, Débora e Marcelo Índio que me deram muita força no início quando estava desesperado com as matérias. Ao Fabrício e Luis Fernando que não deixaram de acreditar e de me motivar, de se mostrarem solícitos no momento em que mais precisei. Ao Dr. Ednei pelo auxílio no período que fiquei no rio. Ao Prof. Luis Gustavo pelas idéias, pela amizade e pelo auxílio nas viagens.

A Petrobras, por me auxiliar durante esse período. Tenho um orgulho enorme de trabalhar na maior empresa do país.

Ao Pr. Ricardo, sua amizade foi fundamental nos momentos de crise e pelo incentivo. A Igreja Batista Jeová-Rafá minha segunda família.

Enfim, agradeço a todos que de alguma forma contribuíram para que este trabalho fosse finalizado. Espero um dia poder retribuir de alguma maneira toda força e apoio que me ofereceram.

RESUMO

SILVA, Izalmo Primo da. **Um Modelo de Replicação em Ambientes P2P com Mobilidade**. Orientadores: Vanessa Braganholo Murta e Carlo Emmanoel Tolla de Oliveira. Rio de Janeiro. Dissertação (Mestrado em Informática) – Programa de Pós-Graduação em Informática, Universidade Federal do Rio de Janeiro, 2008.

Ambientes ponto-a-ponto (*peer-to-peer* - P2P) com mobilidade são caracterizados pela heterogeneidade, pela descentralização dos dados, pela constante mudança na localização dos nós (mobilidade), e também pela constante entrada e saída dos nós da rede. Sistemas P2P desenvolvidos para esse ambiente adotam uma abordagem completamente descentralizada para gerenciamento de dados e recursos. Assim, um dos principais problemas consiste em garantir acesso a itens de dados de forma distribuída, mantendo a consistência nas atualizações. Para melhorar a disponibilidade dos dados e obter ganhos de desempenho, sistemas de gerenciamento de dados e recursos P2P utilizam replicação de dados. A idéia básica consiste em armazenar múltiplas cópias de itens de dados em diferentes nós, o que torna necessário manter consistência entre esses itens replicados após uma atualização.

Esse trabalho propõe um modelo de replicação que visa garantir consistência de forma seqüencial permitindo várias solicitações de atualização simultâneas, mantendo os dados atuais disponíveis na rede e facilitando sua localização em ambientes P2P com Mobilidade. A proposta consiste de algumas alterações no modelo único mestre, criando um modelo denominado múltiplos únicos mestres. A principal preocupação deste trabalho está na manutenção dos itens de dados mais atuais na rede para localização. Ambientes com alta mobilidade são vulneráveis a falhas constantes dos nós. Falhas do nó mestre no modelo único mestre são o grande gargalo do modelo, e podem fazer com que itens de dados mais atuais sejam perdidos, dependendo do momento em que a falha ocorre. Para evitar esse problema, propomos uma função de propagação de réplicas que visa aumentar a disponibilidade dos itens de dados mais atuais na rede. Dessa forma, em uma eventual falha do mestre, é possível escolher um novo mestre para um dado específico, sabendo que algum nó da rede possuirá a informação mais atual para a escolha de um novo mestre com garantia de consistência. Ainda, apresentamos a implementação do modelo na localização de dados, a avaliação do modelo de replicação e a aplicabilidade num estudo de caso.

ABSTRACT

SILVA, Izalmo Primo da. **A replication model in P2P environments with mobility**. Advisors: Vanessa Braganholo Murta and Carlo Emmanoel Tolla de Oliveira. Rio de Janeiro. Dissertação (Mestrado em Informática) – Programa de Pós-Graduação em Informática, Universidade Federal do Rio de Janeiro, 2008.

Peer-to-peer (P2P) environments with mobility are characterized by their heterogeneity, data decentralization, constant peer location change (mobility), and by peer's continuously arriving and leaving the system. P2P systems developed for these environments use a completely decentralized approach for data and resource management. Hence, one of the main problems consists in guaranteeing the data access in a distributed form, maintaining the consistency on the updates. To improve data availability and performance, P2P data and resource management systems use data replication. The basic idea consists in storing multiple copies of the data in different nodes, what then makes imperative to keep the consistency among the replicated items after an update.

This work proposes a replicated model to guarantee sequencial consistency accepting several concurrent update requests, keeping the current data available in the network and making it easier to locate data items in mobile P2P environments. The proposal consists in some changes in the single master model, creating a model called multiple single masters. The main concern of this work is to keep the most up-to-date data available on the network so they can be located. High mobile environments are vulnerable to constant failing nodes. The fault of the master node in the single master model is its great bottleneck, and may result in the loss of the most up-to-date data items depending on the moment it fails. To avoid this, we propose a replica propagation function that will increase the accessibility to the most up-to-date data items on the network. This way, in an eventual fault of the master node, it is possible to choose a new master for a specific data, knowing that some other node on the network will have the most up-to-date information to be chosen, with consistency guarantee. We show either the implementation of the model to locate data, its evaluation and applicability on a case study.

LISTA DE FIGURAS

TABELAS

SIGLAS

CPU	*Central Processing Unit*
DHT	*Distributed Hash Table*
IP	*Internet Protocol*
P2P	*Peer – to – Peer*
PYRO	*Python Remote Objects*
RMI	*Remote Method Invocation*
URI	*Uniform Resource Identifier*

Sumário

1 Introdução

Sistemas Computacionais estão cada vez mais presentes no dia-a-dia do ser humano. A integração dos computadores portáteis com as recentes tecnologias de comunicação celular, redes de comunicação sem fio e serviços via satélite estão possibilitando aos usuários de dispositivos móveis manterem-se conectados enquanto se movimentam livremente, tendo acesso a recursos, serviços e informações compartilhadas pelo ambiente. Esses ambientes são caracterizados pela sua heterogeneidade, pela descentralização dos dados, pela constante mudança na localização dos nós (mobilidade), e também pela constante entrada e saída dos nós da rede. Tais ambientes são viabilizados por sistemas ponto-a-ponto (*peer-to-peer* - P2P). Sistemas P2P adotam uma abordagem completamente descentralizada para gerenciamento de dados e recursos. A distribuição do armazenamento de dados e processamento entre vários nós (chamados *peers*) permite um alto grau de escalabilidade sem a necessidade de poderosos servidores. O sucesso dos sistemas P2P se deve aos diversos benefícios que eles proporcionam, tais como: alta escalabilidade, auto-organização, balanceamento de carga, processamento paralelo e tolerância a falhas através de massiva replicação. Apesar das muitas vantagens, alguns problemas são encontrados principalmente pela sua característica descentralizada e pela possibilidade de constante conexão e desconexão dos nós da rede. Um dos principais problemas consiste em garantir acesso a itens de dados de forma distribuída, mantendo a consistência nas atualizações.

Para melhorar a disponibilidade dos dados e obter ganhos de desempenho, sistemas de gerenciamento de dados e recursos P2P utilizam replicação de dados. A idéia básica consiste em armazenar múltiplas cópias de itens de dados em diferentes servidores, distribuídos através de uma rede de comunicação. Assim, as aplicações podem acessar os dados de qualquer uma das réplicas podendo inclusive continuar sua execução mesmo que algum dos nós falhe. Entretanto, estes benefícios trazem a necessidade de atualizar todas as cópias de um

item sempre que este for alterado em algum nó, ou seja, torna-se necessário manter consistência entre os itens replicados nos diversos nós que possuem o dado. Replicação de dados não é uma idéia nova, mas sua utilização tem crescido bastante principalmente devido às tecnologias de computação móvel (Barbará 1999) e pervasiva (Satyanarayanan 2001). Essa técnica tem se mostrado de extrema importância em redes com alta mobilidade e em redes fracamente conectadas, pois possibilita o compartilhamento eficiente dos dados apesar das restrições impostas pelo ambiente.

Abordagens que trabalham atualmente com replicação de dados são baseadas em dois modelos de replicação: modelo de único mestre e de múltiplos mestres. No primeiro, um nó é responsável pelas operações de escrita (denominado nó mestre) enquanto os demais nós possuem uma cópia do dado para leitura (denominados escravos) (Vidal, Pacitti, Valduriez 2007). Quando um nó escravo deseja atualizar um dado, ele solicita essa atualização ao mestre que aplica a atualização e a repassa para os demais nós escravos. No modelo múltiplos mestres, por sua vez, cada nó que possua um dado replicado tem a possibilidade de escrever sobre o mesmo. Muitas vezes, essas operações concorrentes causam divergências e conflitos entre as réplicas. Para resolvê-los é necessário um processo de reconciliação. Dessa forma, todas as réplicas chegarão a uma versão consistente algum tempo após a atualização ter ocorrido (Vidal, Pacitti, Valduriez 2007).

Ambos os modelos possuem suas vantagens e desvantagens. Modelos de único mestre são vantajosos pela simplicidade para garantia de consistência, mas possuem como desvantagem o fato de possuírem um único ponto de falha. Em ambientes com constante mobilidade, isso pode ser um problema, já que o mestre pode estar ausente durante uma solicitação de atualização. Em modelos único mestre as atualizações são repassadas para todos os nós escravos após terem sido atualizados no mestre. Isso é uma desvantagem em

ambientes P2P não-estruturados, pois a rede não possui uma estrutura lógica, e o número de nós nela existente não é conhecido. Assim, para compensar tal problema é necessário que todos os nós se comuniquem através de algum algoritmo de inundação (Genç, 2005), o qual acarreta aumento de tráfego na rede. Algumas soluções para estes problemas são apresentadas por Budhiraja, Marzullo, Schneider e Toueg (1993) e Tanenbaum e Steen (2007). Budhiraja, Marzullo, Schneider e Toueg (1993) propõem um modelo de replicação onde, quando uma atualização é aplicada por um cliente sobre um servidor mestre (chamado de servidor mestre primário), ele encaminha as atualizações para servidores mestre secundários, os quais também efetuam a atualização em suas réplicas, avisando ao servidor primário o término da atualização (*commit*). O servidor primário, por sua vez, envia uma confirmação ao cliente indicando que a atualização ocorreu com sucesso. Em caso de falha do servidor mestre primário, um servidor replicado assumirá a função deste. Tanenbaum e Steen (2007) propõem que sempre que uma atualização precisa ser feita em um item de dado *X*, o nó solicitante assume como mestre da informação e prossegue com a atualização. Problemas em ambos os casos são encontrados quando o nó mestre atual está fora da rede, sendo necessário um mecanismo que trate este tipo de problema.

Modelos múltiplos mestre resolvem o problema da presença de um único mestre, permitindo que todos os nós possam ser mestres de todas as informações presentes na rede. Assim, é possível que todos os nós alterem uma mesma informação. Essa vantagem torna-se um problema quando é necessário reconciliar essas alterações. Algumas soluções (Monteiro, Brayner, Lifschitz 2007; Thomas 1979; Terry *et al.* 1997; Terry *et al.* 1995) tratam a reconciliação em modelos múltiplos mestres. Monteiro, Brayner e Lifschitz (2007) propõem a criação de um grafo representando as transações aplicadas sobre um dado. Quando um problema de inconsistência ocorre, o grafo forma um ciclo. Resolver este ciclo garante a consistência da informação. Thomas (1979) propõe um algoritmo que trata as atualizações

mediante um consenso entre os servidores que possuem a réplica da informação. Dessa forma, quando um cliente deseja atualizar um dado ele precisa que um conjunto mínimo de servidores esteja disponível e concordem com a atualização. Terry *et al.* (1995, 1997) propõem que os dados sejam replicados entre servidores e que as atualizações feitas pelos clientes sejam consideradas tentativas de escrita que ficam disponíveis para leitura automaticamente após a operação. A proposta garante que os nós servidores trocam informações sobre as atualizações e chegam a um consenso. Todas essas soluções garantem a consistência em níveis diferentes. É importante notar que as técnicas, apesar de atingirem o objetivo, possuem um custo elevado de processamento e de troca de mensagens. Além disso, são necessários algoritmos complexos para resolver o problema de consistência.

Nesse trabalho apresentamos um modelo de replicação que visa garantir consistência de forma seqüencial permitindo várias solicitações de atualização simultâneas, mantendo os dados atuais disponíveis na rede. A proposta consiste de algumas alterações no modelo único mestre, criando um modelo denominado múltiplos-únicos mestres. Esse modelo captura algumas características positivas dos modelos múltiplos mestre e único mestre permitindo que qualquer nó possa assumir o papel de mestre e escravo para um conjunto de dados diferentes. Além disso, qualquer nó pode atualizar uma determinada informação, bastando para isso confirmar a atualização com o mestre daquela informação. Sua principal aplicabilidade está em sistemas onde a consulta aos dados na sua versão mais atual não é crucial, podendo-se aguardar um tempo para obter a versão mais atual. Exemplos deste cenário são: páginas web ou descritores de serviços web, onde a informação armazenada no cliente pode estar desatualizada em relação ao servidor, o que só será percebido quando o usuário solicitar uma atualização da página, por exemplo.

Como prova de conceito, o modelo proposto foi aplicado em um ambiente para localização de serviços.

1.1 Objetivos

O principal objetivo deste trabalho está na manutenção dos dados na sua versão mais atual disponíveis na rede. Ambientes com alta mobilidade são vulneráveis a falhas constantes dos nós. Falhas do nó mestre no modelo único mestre são o grande gargalo do modelo, e podem fazer com que itens de dados mais atuais sejam perdidos, dependendo do momento em que a falha ocorre. Para evitar esse problema, propomos uma função de propagação de réplicas que visa manter a informação atual sempre disponível na rede. Dessa forma, em uma eventual falha do mestre, é possível escolher um novo mestre para um dado específico, sabendo que algum nó da rede possuirá a informação mais atual para a escolha de um novo mestre com garantia de consistência. Ainda, o modelo tenta evitar a troca de informações de atualização entre o mestre e os demais nós, permitindo que a informação no nó só seja atualizada quando este desejar realizar uma alteração. É importante notar que, neste cenário, consultas podem ser realizadas a versões antigas de dados. Para situações onde é necessário garantir consulta sempre à versão corrente do dado, propomos uma alternativa na Seção 4.1 .

Outro objetivo é permitir a localização de dados que expressem a localização física de serviços web móveis. Serviços web comuns são estáticos, mas a proposta de Gonçalves *et al* (2007a;2007b) permite que serviços possam ser migrados. Assim uma vez mudada sua localização, os nós que consomem o serviço perdem sua referência. O modelo de replicação visa criar um ambiente que desatrele a descrição da localização do serviço e o serviço em si. Assim, a descrição do serviço pode estar disponível mesmo que o serviço esteja em transição (sendo migrado de uma máquina para outra).

1.2 Contribuições

Como contribuição principal este trabalho propõe um modelo de replicação para ambientes P2P com mobilidade. Assim, o modelo visa manter dados consistentes na rede, desatrelando o dado do seu respectivo provedor e mantendo a consistência e a atualidade da informação. Essa proposta visa manter dados compartilhados por dispositivos móveis com baixa capacidade de armazenamento e auxiliar na localização desses dados.

Outra contribuição que este trabalho agrega está na possibilidade de localizar dados que representem serviços mesmo que estes sejam migrados ou ainda replicados para qualquer nó em uma rede P2P.

Finalmente, a criação um modelo que possa ser utilizado para trabalhar com dados e com localização de serviços em ambientes P2P com mobilidade, promovendo um mercado de compra e venda de serviços e recursos nesses ambientes.

1.3 Organização do Texto

O restante deste trabalho está organizado como segue. O capítulo 2 apresenta a revisão teórica contendo os principais conceitos aplicados durante todo este trabalho. No capítulo 3 é apresentado o estado da prática referente aos conceitos apresentados. O Modelo de replicação é apresentado no capítulo 4. No capítulo 5 é apresentado um estudo de caso e as simulações feitas sobre o modelo de replicação. O capítulo 6 apresenta a conclusão e trabalhos futuros.

2 Distribuição e Replicação

Este capítulo apresenta os principais conceitos referentes à proposta do modelo de replicação.

2.1 Sistemas Distribuídos Ponto-a-Ponto (P2P)

Um sistema distribuído pode ser definido como aquele "formado por componentes de hardware e software, situados em redes de computadores, e que se comunicam e coordenam suas ações apenas através de trocas de mensagens" (Coulouris, Dollimore, & Kindberg, 2001). A separação espacial dos computadores que formam esses sistemas distribuídos pode variar desde poucos metros, em uma rede local, até quilômetros de distância, em uma rede largamente dispersa. Da mesma maneira, podem variar bastante os desafios enfrentados pelos projetistas desses sistemas frente aos problemas de heterogeneidade, segurança e confiabilidade (tolerância a falhas).

Envios e recepções de mensagens não ocorrem instantaneamente, e a conseqüência imediata do fato da comunicação entre os componentes dos sistemas distribuídos se dar apenas por trocas de mensagens é a inexistência de um relógio global, no qual todos os computadores possam se basear. Essa característica dificulta a obtenção de um estado global dos componentes do sistema, importante para a resolução de uma série de problemas de sincronização que surgem nas aplicações. Questões de coordenação e sincronização em sistemas distribuídos decorrem exatamente da característica de concorrência entre seus componentes e a necessidade de compartilhamento de recursos entre eles. Esses problemas formam uma classe que inclui a necessidade, entre outros, de algoritmos de eleição, detecção de *deadlocks* (impasses) distribuídos, algoritmos de conciliação, etc. (Lynch, 1996).

Em comparação ao modelo cliente-servidor, muitas vantagens podem ser observadas no modelo distribuído. Em contrapartida, dificuldades são encontradas principalmente no que diz respeito à necessidade de troca de mensagens para garantia de consistência no modelo. Características positivas são vistas no modelo distribuído, tais como transparência, balanceamento de carga, processamento distribuídos, etc., os quais trazem uma enorme vantagem aos sistemas adeptos desse modelo.

Sistemas Ponto-a-ponto (P2P) são sistemas distribuídos que adotam uma abordagem completamente descentralizada para gerenciamento de recursos. A distribuição do armazenamento e processamento dos dados entre vários nós (chamados *peers*) permite um alto grau de escalabilidade sem a necessidade de poderosos servidores. Sistemas P2P têm sido usados para compartilhamento computacional (Seti, 2007), comunicação ICQ (Icq, 2007) ou de dados (Gnutella, 2007;Kazaa, 2007). O sucesso dos sistemas P2P se deve aos diversos benefícios que eles proporcionam, tais como: alta escalabilidade, auto-organização, balanceamento de carga, processamento paralelo e tolerância a falhas através de massiva replicação. Além disso, eles podem ser muito úteis no contexto de computação móvel ou computação pervasiva. No entanto, existem sistemas que só funcionam para aplicações simples (ex.: compartilhamento de arquivos), suportam funções limitadas (ex.: somente busca por palavra-chave) e usam técnicas simples as quais causam problemas de desempenho. Muitas pesquisas têm desafiado os problemas referentes aos sistemas P2P, dentre esses podemos destacar o alto grau de compartilhamento de dados com segurança, eficiência e consistência (Valduriez & Pacitti, 2004).

Quando consideramos gerenciamento de dados, os principais requisitos de um sistema P2P são (Daswani, Molina, & Yang, 2003):

- **Autonomia:** um nó autônomo deve ser capaz de entrar e deixar o sistema em qualquer tempo e sem nenhuma restrição. Ele deve também ser capaz de controlar os dados que ele armazenou ou que qualquer outro nó tenha armazenado nele.
- **Expressividade das Consultas:** a linguagem de consulta deve permitir ao usuário descrever o dado desejado em um nível apropriado de detalhe. A forma simples de consulta baseada em palavras-chave é apropriada apenas para arquivos. Para dados estruturados é necessário outro tipo de linguagem de consulta.
- **Eficiência:** o uso eficiente dos recursos do sistema P2P (largura de banda, poder de computação, armazenamento) deve resultar em um baixo custo e assim num alto *throughput* das consultas, isto é, um alto número de consultas processados em um sistema P2P num mesmo intervalo de tempo.
- **Qualidade do serviço:** refere-se à percepção que o usuário tem da eficiência do sistema, ex.; integridade dos resultados da consulta, consistência dos dados, disponibilidade dos dados, tempo de resposta da consulta, etc.
- **Tolerância a Falhas:** eficiência e qualidade dos serviços devem ser providas apesar das ocorrências de falhas dos nós. Dada a natureza dinâmica dos nós, os quais podem deixar a rede ou falhar em qualquer tempo, a única solução é confiar na replicação de dados.
- **Segurança:** pela natureza aberta dos sistemas P2P, garantir segurança é um dos maiores desafios. É preciso controlar o acesso aos dados, o que inclui garantir os direitos de acesso somente ao proprietário ou a quem ele delegue acesso.

Existem muitas arquiteturas e topologias de rede diferentes que são possíveis em sistemas P2P. Dependendo da arquitetura, os requisitos acima têm uma dificuldade maior ou menor de serem atendidos. Para simplificar, consideramos as três classes principais de topologia: não-estruturado, estruturado e super-nós. Sistemas não-estruturados e estruturados

são chamados sistemas P2P "puros" enquanto que sistemas de super-nós são qualificados como "híbridos". Sistemas P2P puros consideram que todos os nós são iguais, ou seja, nenhum nó tem algum tipo de funcionalidade específica dentro do sistema.

2.1.1 Sistemas Não Estruturados

Em ambientes P2P não-estruturados, a rede é criada de uma maneira não determinística (*ad hoc*) e a localização dos dados não tem nenhuma relação com a topologia da rede. Cada nó conhece seus vizinhos e se comunica com eles, mas não conhece os recursos que eles têm. O roteamento das consultas é tipicamente feito pela inundação (*flooding*) da consulta para os nós vizinhos em uma determinada distância de passos do nó que originou a consulta. Mecanismos de consulta baseados em inundação geralmente não consultam toda a rede por causa do grande tráfego que isso acarretaria. Com isso, a possibilidade de uma informação não ser encontrada é alta se o nó que contém a informação estiver muito longe do nó que originou a consulta.

Como todos os nós são igualmente capazes de trocar informações, é necessário que a tolerância a falhas seja muito alta. A Figura 2.1 ilustra um sistema não-estruturado simples onde os nós possuem alta autonomia. Cada nó suporta o mesmo tipo de comunicação P2P. Deste modo, basta que um nó conheça seu vizinho para poder se comunicar com ele, ou consultar seus dados (Valduriez & Pacitti, 2004).

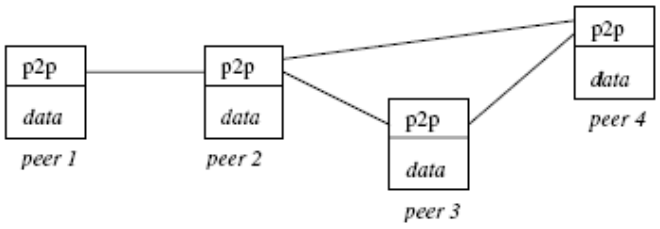

Figura 2.1 – Rede ponto-a-ponto não estruturada. (Valduriez & Pacitti, 2004)

2.1.2 Sistemas Estruturados

Sistemas P2P estruturados têm emergido para resolver o problema de escalabilidade dos sistemas não estruturados. Eles atingem esse objetivo controlando a topologia da rede e a localização dos dados. Dados (ou ponteiros para eles) são encontrados precisamente em uma localização específica e mapeamentos entre dados e suas localizações são providos na forma de uma tabela de roteamento distribuído (Vidal, Pacitti & Valduriez, 2007).

Um dos maiores representantes desse tipo de sistemas são os sistemas estruturados baseados em tabelas *hash* distribuídas (DHT), como por exemplo, CAN (Ratnasamy *et al* , 2001) e CHORD (Stoica *et al*, 2001). Um sistema DHT provê uma interface para uma tabela *hash* com primitivas *put(chave,valor)* e *get(chave)*, onde *chave* é um identificador único e cada nó é responsável por armazenar o valor (conteúdo associado ao identificador). Existe um esquema de consulta sobre a rede que entrega a requisição por uma chave para um nó responsável por aquela determinada chave. Isto permite desempenho da ordem de $O(\log n)$ nas consultas, onde n é o número de nós na rede.

Como um nó é responsável por armazenar o valor correspondente para um grupo de chaves, a autonomia é limitada. Além disso, consultas DHT restringem-se a uma comparação exata de chaves, embora alguns pesquisadores estejam trabalhando com o objetivo de estender a capacidade de consultas em DHTs (Valduriez & Pacitti, 2004).

2.1.3 Sistemas com Super-Nós

Redes P2P não estruturadas e estruturadas são consideradas "puras" porque todos os seus nós provêm funcionalidades iguais. Em contraste, redes do tipo super-nós são híbridos entre sistemas cliente-servidor e redes P2P puras. Como nos sistemas cliente servidor, alguns nós, os super-nós, atuam como servidores dedicados para alguns outros nós e podem executar funções complexas como indexação, processamento de consultas, controle de acesso e

gerenciamento de meta-dados. Se houver apenas um super nó, o modelo é reduzido a um modelo cliente servidor comum.

Super-nós podem ser dinamicamente eleitos (ex.: baseado em largura de banda e poder de processamento) e substituídos em caso de falha. Numa rede de super-nós, quando um nó *n* quer encontrar um outro nó *m*, o nó *n* simplesmente envia a requisição, a qual pode ser expressa em uma linguagem de alto nível, para o super-nó responsável. O super-nó pode então encontrar o nó *m* solicitado diretamente através do índice ou indiretamente consultando os super-nós vizinhos.

As principais vantagens das redes com super-nós são a eficiência percebida pelo usuário, como por exemplo, o tempo de resposta da consulta, e a qualidade do serviço. O tempo necessário para encontrar uma informação pelo acesso direto ao índice em um super-nó é muito menor do que na inundação da rede (*flooding*). Além disso, as diferentes capacidades dos nós em relação a poder da CPU, largura de banda ou capacidade de armazenamento são levadas em conta quando se promove um nó a super-nó. Nas redes P2P puras, todos os nós são igualmente carregados, sem levar em consideração suas capacidades. Controle de acesso e segurança da informação podem ser melhor gerenciados em super-nós. Entretanto, a autonomia é restrita visto que os nós não podem conectar-se livremente a qualquer super-nó. A tolerância a falhas é baixa, pois os super-nós são pontos únicos de falha para os seus sub-nós. No entanto, a mudança dinâmica de super-nós pode aliviar este problema (Vidal, Pacitti & Valduriez, 2007).

2.1.4 Comparativo entre redes P2P

Abaixo apresentamos a comparação entre os tipos de redes P2P descritos por Vidal, Pacitti e Valduriez (2007) baseado nos requisitos autonomia, expressividade das consultas,

eficiência, qualidade do serviço, tolerância a falhas e segurança, apresentados no início desta seção.

Nota-se que os sistemas de super-nós são os que melhor atendem aos requisitos.

Tabela 1 – Comparativo entre os tipos de rede P2P (Vidal, Pacitti e Valduriez, 2007)

Requisitos	Não Estruturado	Estruturado	Super-Nó
Autonomia	Alta	Baixa	Moderada
Expressividade das Consultas	Alta	Baixa	Alta
Eficiência	Baixa	Alta	Alta
Qualidade do Serviço	Baixa	Alta	Alta
Tolerância a Falhas	Alta	Alta	Baixa
Segurança	Baixa	Baixa	Alta

2.2 Modelos de Replicação de Dados

Replicar dados (Vidal, Pacitti, & Valduriez, 2007) significa manter múltiplas cópias de objetos de dados (chamadas réplicas) em lugares separados. Objetos de dados (também referidos como objetos) podem ser desde tabelas a apenas tuplas, dependendo se a tabela é replicada completamente ou se apenas uma tupla é replicada. Uma réplica é a cópia de um objeto armazenado em algum lugar. Estado é o grupo de valores associados a um objeto em uma réplica em um tempo.

Há duas razões primárias para replicar: confiabilidade e desempenho. Em primeiro lugar, dados são replicados para aumentar confiabilidade de um sistema. Se um sistema foi replicado, é possível continuar trabalhando após a queda de uma réplica simplesmente alternando, em determinado instante de tempo, para uma das outras réplicas. Outra razão é o aumento de desempenho, que é importante quando um sistema distribuído sofre o aumento de demanda. Isso ocorre, por exemplo, quando um número cada vez maior de processos precisa acessar dados que são gerenciados por um único servidor. Nesse caso, o desempenho pode ser melhorado ao se replicar o servidor e, na seqüência, dividir o trabalho (Tanenbaum & Steen, 2007).

Réplicas de objetos podem ser classificadas como: cópia primária (aceita operações de leitura e escrita) ou cópia secundária (aceita somente operações de leitura). Todo nó que armazena uma cópia primária é chamado de nó mestre, enquanto o nó que armazena uma cópia secundária é denominado nó escravo.

Dentre as vantagens da replicação de dados podemos citar:

- Melhorar a disponibilidade do sistema removendo os pontos de falha únicos;
- Melhorar o desempenho do sistema reduzindo a quantidade de comunicação e aumentando a taxa de desempenho do sistema;
- Suportar o crescimento do sistema com tempo de resposta aceitável melhorando assim a escalabilidade do sistema.

De acordo com Gray, Helland, O'Neil, & Shasha (1996) mecanismos de controle de replicação de dados devem estar de acordo com dois parâmetros: *qual* réplica pode ser atualizada e *quando* as atualizações são propagadas para todas as réplicas. Assim, de acordo com o primeiro parâmetro, modelos de replicação podem ser classificadas como: *único* mestre ou *vários* mestres. Já de acordo com o segundo parâmetro, estratégias de propagação podem ser divididas em abordagens *síncronas* ou *assíncronas.* Os mecanismos de controle são afetados ainda pelo tipo de replicação que pode ser *total* ou *parcial*. Abaixo cada uma dessas características é explicada em maiores detalhes.

2.2.1 Modelo Único Mestre x Múltiplos Mestres

O modelo de replicação de mestre único (Figura 2.2), também chamado de replicação mestre/escravo, guarda apenas uma cópia primária para cada réplica do objeto. Nesse caso, o objeto é primeiro alterado na cópia primária (cópia principal) no nó mestre e então a alteração é aplicada nas cópias secundárias (réplicas) armazenadas nos nós escravos. Esta abordagem

tem um gargalo por possuir um ponto único de falha, pois uma eventual falha do nó mestre bloqueia qualquer operação de atualização.

Modelos de replicação de múltiplos mestres (Figura 2.3), por sua vez, permitem que todos os nós armazenem cópias primárias do mesmo objeto. Assim, todas as cópias podem ser atualizadas concorrentemente. Esta estratégia não possui o problema de um ponto único de falha, mas por conseqüência, cria-se um problema de garantia de consistência de dados. Atualizações concorrentes devem ser coordenadas ou um algoritmo de reconciliação deve ser aplicado para resolver a divergência entre as réplicas.

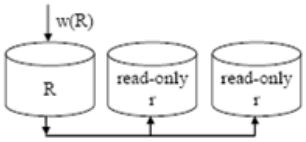

Figura 2.2 - Replicação de Mestre Único (Vidal, Pacitti & Valduriez, 2007); R é a cópia primária e r a cópia secundária.

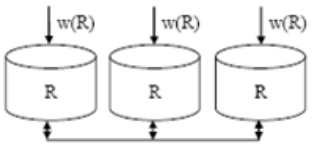

Figura 2.3 - Replicação de Múltiplos Mestres (Vidal, Pacitti & Valduriez, 2007); R é a cópia primária do mesmo objeto.

2.2.2 Propagação Síncrona x Propagação Assíncrona

Estratégias de propagação podem ser síncronas ou assíncronas. Na estratégia de propagação síncrona (Figura 2.4) uma atualização é feita em todas as réplicas dentro de uma mesma transação. Como resultado, quando a transação finaliza, todas as réplicas têm o mesmo estado. Na estratégia assíncrona (Figura 2.5), a atualização é feita sobre uma única réplica. Assim, uma transação é iniciada para atualizar esta réplica e finalizada após a atualização da mesma. Depois de algum tempo as atualizações dessa réplica são propagadas para as demais. Deste modo, as réplicas podem ter estados diferentes num mesmo instante de tempo.

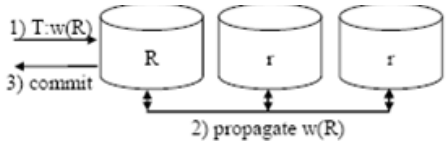

Figura 2.4 - Princípio de Propagação Síncrona (Vidal, Pacitti & Valduriez, 2007).

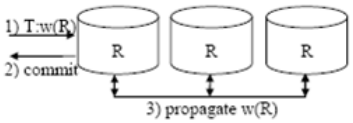

Figura 2.5 - Princípio de Propagação Assíncrona (Vidal, Pacitti & Valduriez, 2007).

2.2.3 Replicação Total x Replicação Parcial

Replicação total (Figura 2.6) consiste em armazenar uma cópia de todos os objetos compartilhados em todos os nós participantes do sistema, enquanto que na replicação parcial (Figura 2.7) cada nó guarda uma cópia de um grupo de objetos compartilhados de forma que objetos replicados em um nó devem ser diferentes de objetos replicados em outro nó.

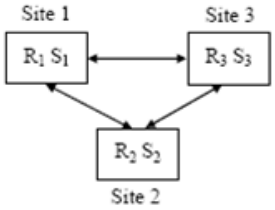

Figura 2.6 - Modelo de Replicação Total com dois objetos R e S (Vidal, Pacitti & Valduriez, 2007).

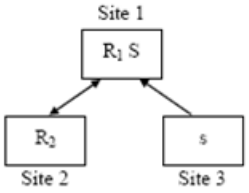

Figura 2.7 - Modelo de Replicação Parcial com dois objetos R e S (Vidal, Pacitti & Valduriez, 2007).

2.3 Considerações Finais

Este capítulo apresentou os principais conceitos referentes à proposta do modelo de replicação. A seção 2.1 descreve os principais conceitos sobre sistemas distribuídos P2P. Na nossa proposta, os dispositivos presentes na rede fazem parte de um ambiente de P2P com mobilidade, assim, a comunicação e a replicação dos dados acontecerá numa rede P2P baseada em super-nós.

Na seção 2.2 é apresentado o conceito de replicação de dados. No próximo capítulo, apresentamos o estado da prática analisado para viabilizar nossa proposta.

3 Estado da Prática

Neste capítulo apresentaremos as soluções existentes para replicação de dados em ambientes P2P e soluções sobre ambientes de memória compartilhada. Essas soluções serviram de base de conhecimento para o desenvolvimento da implementação e do modelo de replicação.

3.1 Soluções de Replicação em Sistemas P2P

3.1.1 JXTA

JXTA (Wilson, 2007) está posicionada como uma pilha P2P, uma camada localizada acima do sistema operacional ou máquina virtual e abaixo das aplicações e dos serviços P2P. Utiliza uma abordagem *pull* centralizado em *Peers Rendezvous*. Pode ser descrita simplesmente como uma tecnologia que permite a comunicação entre *peers*. *Peer* é qualquer dispositivo em uma rede JXTA. Cada *peer* é associado a um identificador único, um "*peer* ID", e pertence a um ou mais *peergroups*. Dentro dos *peergroups*, os *peers* cooperam e têm funções similares sob um conjunto unificado de capacidades e restrições. JXTA provê protocolos para as funções básicas: criar e encontrar grupos, entrar e sair de grupos, monitorar os grupos, conversar com outros grupos e *peers*, compartilhar conteúdo e serviços – tudo isso é realizado através da publicação e troca de anúncios XML e mensagens entre os *peers*.

Numa rede JXTA, alguns *peers* recebem uma função especial. Estes *peers* são chamados de *rendezvous peers*. *Rendezvous Peers* são nós especiais na topologia da rede que atuam como pontos de descobertas para os serviços oferecidos em uma rede JXTA.

Por exemplo, considerando a Figura 3.1 como a topologia atual de uma rede JXTA, se o *Peer A* fizer um pedido de alguma informação, este é enviado para o *Peer R1*. Este *peer* é de *rendezvous*. Caso este não possua a informação pedida, ele encaminha o pedido para os *peers*

que conhece (R2 e R3). Estes efetuam o mesmo procedimento: verificar se a informação pedida existe e encaminhar. O funcionamento de cada *peer de rendezvous* é semelhante ao de um roteador da Internet.

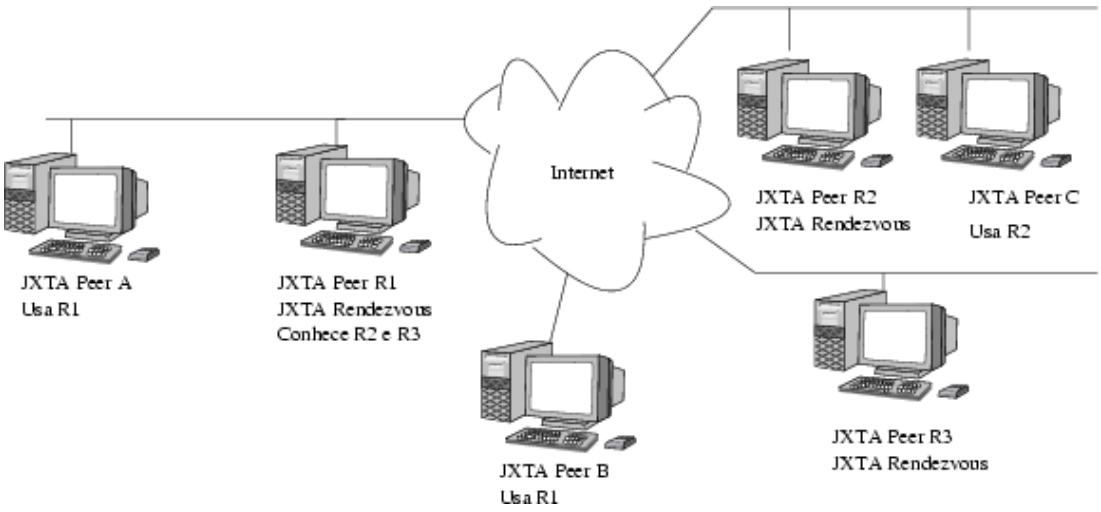

Figura 3.1 – Exemplo de Topologia de uma rede JXTA.

3.1.2 Napster

O Napster (Napster, 2007) foi a aplicação propulsora da plataforma P2P que utiliza uma abordagem de *pull* distribuída. Ele consiste de uma aplicação de rede P2P híbrida. Nele há um servidor centralizado onde todas as funcionalidades de procura de anúncios e de *peers* (nós da rede) são realizadas. O Napster pode ser modelado como um único *Rendezvous peer* e diversos *peers* simples. O *Rendezvous peer* do Napster provê a todos os *peers* simples a capacidade de localizar arquivos de música anunciados em um único arquivo, endereço de IP e porta de comunicação. Todos os *peers* simples utilizam essas informações para se conectarem diretamente com outros *peers*, e assim podem realizar cópias dos arquivos compartilhados.

Napster é um aplicação P2P, suportado por uma rede super-nós que depende de servidores centrais para mediar a interação entre nós, como apresentado na Figura 3.2. Todo nó que compartilha arquivos se conecta a um super-nó e publica os arquivos que ele contem.

O super-nó guarda a informação da conexão (ex. endereço IP, conectividade de banda) e uma lista dos arquivos providos por cada nó. Para retornar um arquivo de qualquer parte da rede, um nó envia uma requisição (denotado por *query* na Figura 3.2) para o super-nó, o qual pesquisa para comparar no seu índice e retornar a lista dos nós que guardam o arquivo desejado (denotado por *reply* na Figura 3.2). O nó que submeteu a consulta então abre uma conexão direta com um ou mais nós que pertencem à resposta enviada pelo super-nó e então baixam o arquivo desejado.

O Napster apóia-se em replicação para melhorar a disponibilidade dos arquivos e aumentar o desempenho, mas ele não implementa uma solução de replicação particular. De fato, a replicação ocorre naturalmente com os nós requisitando e copiando os arquivos de um outro nó. Esse modelo é denominado de replicação passiva. O Napster é simples de se implementar e eficiente para localizar arquivos, mas tem duas limitações principais. Primeiro, ele armazena somente dados estáticos (ex. arquivos de música). Segundo, os super-nós são pontos únicos de falha e são vulneráveis a ataques maliciosos, o que pode acontecer com qualquer aplicação que utiliza este tipo de rede.

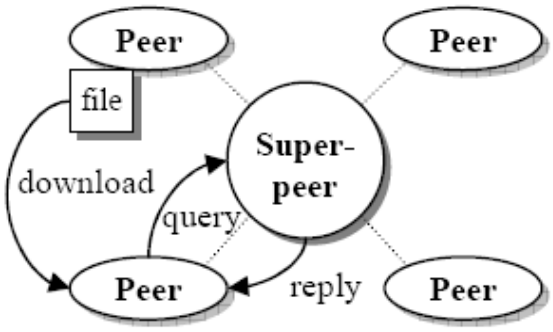

Figura 3.2 – Modelo de Replicação Passiva do Napster

3.1.3 Gnutella

Na rede Gnulella (Wilson, 2007) cada *peer* atua como *peer* simples e *Rendezvous peer* baseado numa abordagem de *pull* distribuído. As consultas realizadas na rede são propagadas por todos os *peers* conhecidos deste *peer*, e estes a propagam para os seus *peers* conhecidos.

Gnutella (Jovanovic, 2000; Jovanovic, Annexstein, & Berman, 2001) é um sistema de arquivos P2P compartilhados construído sobre serviço de rede IP. Ele trabalha sobre uma rede não estruturada. Para obter um arquivo compartilhado, o nó que requisita o arquivo (requisitante) deve executar três tarefas: entrar na rede Gnutella, procurar o arquivo desejado e baixá-lo. Para entrar na rede Gnutella, o requisitante se conecta a um grupo de nós já conectados (uma lista está disponível em um banco de dados como em *gnutellahosts* (*Gnutellahost*, 2008)) e envia para eles uma requisição para se anunciar a eles. Cada um desses nós então envia de volta uma mensagem contendo seu IP e porta bem como o número e o tamanho dos seus arquivos compartilhados. O requisitante ainda propaga a requisição de anúncio para seus vizinhos.

Uma vez conectado, o requisitante pode pesquisar o arquivo desejado como ilustrado na Figura 3.3. Nesta figura, foi utilizado o número antes da mensagem para indicar o tempo no qual elas são trocadas (ex. todas as mensagens precedidas de 1 são trocadas no mesmo tempo t1). O mecanismo de pesquisa inicia com uma mensagem de consulta (*q*) enviada pelo requisitante para seus vizinhos (1:q na Figura 3.3a) e distribuída para toda a rede por inundação (*flooding*) (2:q e 3:q na Figura 3.3a). Uma replicação de um nó que possa satisfazer a *q* é chamada de *query hit (*denominado por *qh*). Quando o requisitante recebe uma *query hit (qh na* Figura 3.3*b)*, ele se conecta diretamente com o nó que armazena o arquivo desejado e inicia o processo de baixar o arquivo (*download*).

Napster e Gnutella implementam replicação passiva, isto é, os arquivos são somente replicados para os nós que os requisitam. Para melhorar a localização dos dados, tanto quanto a disponibilidade e desempenho, métodos de replicação ativa foram propostos (Lv, Cao, Cohen, Li, & Shenker, Junho 2002) no qual arquivos devem ser proativamente replicados para nós arbitrários. Porém, Gnuttela continua com sua maior limitação: ele somente trabalha com arquivos estáticos.

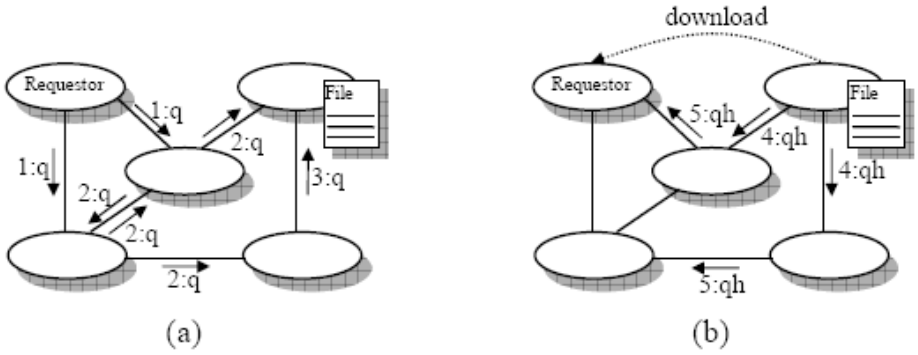

Figura 3.3 – Gnutella: um exemplo do mecanismo de busca. (a) O nó requisitante (*requestor*) submete uma consulta *q* que é propagada por inundação (flooding). (b) Quando o requisitante recebe um *query hit* (*qh*), ele conecta ao nó que guarda a informação para baixá-la.

3.1.4 Mecanismo para Consistência de Dados em Ambientes de Computação Móvel

O mecanismo proposto em (Monteiro, Angelo, & Lifschitz, 2007) é baseado em uma estratégia similar à utilizada pelo mecanismo de teste de grafo de serialização proposto em (Casanova, 1981). Nela um grafo é construído representando as transações aplicadas sobre um dado. Este é monitorado e gerenciado de forma dinâmica para que sempre seja um grafo acíclico. Adicionalmente o protocolo explora informações temporais referentes ao momento em que um item de dado foi lido ou atualizado.

Nesse protocolo, as funções para controle de concorrência estão distribuídas entre os clientes, servidores e o servidor primário (provedor do serviço de consistência). Assim, é

assumido que o servidor primário, os demais servidores (réplicas) e os clientes apresentam funcionalidades específicas para gerenciamento de transações.

Cada item de dado possui associado a ele um marcador de tempo (*timestamp).* O marcador é composto por duas partes: um marcador de versão e um de sub-versão. O marcador de versão é incrementado a cada confirmação (*commit*) de uma transação qualquer *Ti* que executa uma operação de escrita sobre um item de dados. Toda confirmação de atualização (alteração da versão) é feita sobre o servidor primário. Já a sub-versão é incrementada a cada escrita de uma transação ativa (não efetivada, ou seja, *uncommitted*) escrita em qualquer réplica.

Após a confirmação (*commit)* de uma determinada transação *Ti*, o servidor primário propaga os novos valores dos itens de dados atualizados por *Ti*, juntamente com os marcadores, para todas as réplicas com as quais tiver comunicação. Caso um servidor que armazena as réplicas não esteja ativo, uma tarefa é agendada para atualização no momento em que o servidor de réplicas e o servidor primário estiverem conectados.

Periodicamente cada servidor replicado deve enviar um pacote ao servidor primário contendo as operações de leitura e escrita executadas até o momento em sua cópia local, juntamente com seus respectivos marcadores de tempo, além do seu próprio identificador. As operações já informadas não precisam ser enviadas novamente. Essas informações são recebidas por um escalonador, no servidor primário, e são utilizadas para sincronizar as operações das diversas transações, preservando (segundo algum critério) a consistência dos dados replicados.

Quando um cliente recebe um pedido de confirmação (*commit*) ou cancelamento (*abort*) de uma transação móvel, ele deve submeter este pedido a qualquer servidor replicado, o qual encaminhará esta requisição ao servidor primário.

Para assegurar a consistência dos dados replicados o mecanismo baseia-se na comparação dos marcadores de versão e sub-versão associados às operações de leitura e escrita. Dependendo dessa comparação, a transação é inserida no grafo de transações numa ordem. Se houver ciclos, ou seja, uma transação T1 aponta para uma transação T2 e vice-versa, o algoritmo rejeita a transação mantendo apenas a primeira transação e garantindo a consistência. Esse tipo de atitude garante que se duas transações T1 e T2 tentarem confirmar (*commit)* suas respectivas atualizações, isso não irá inserir informações inconsistentes na cópia principal, mesmo que cada transação tenha lido e alterado um mesmo dado em dois servidores replicados diferentes,

Semelhante à abordagem do Bayou (detalhada na próxima seção), um servidor primário é utilizado e a coerência entre as réplicas é garantida apenas eventualmente. Porém este protocolo resolve a detecção de conflitos através da comparação dos marcadores de tempo (*timestamp*) o que possui um custo menor.

3.1.5 Bayou

Bayou (Terry, Theimer, Petersen, Demers, & Spreitzer, 1997; Terry, Theimer, Petersen, Demers, Spreitzer, & Hauser, 1995) é um sistema de armazenamento replicado que utiliza um nível fraco de consistência e que foi projetado para ambientes de computação móvel que inclui dispositivos portáteis fracamente conectados. O sistema *Bayou* provê uma infra-estrutura para o desenvolvimento de uma variedade de aplicações colaborativas, que não tenham requisitos de tempo real, tais como calendários, *e-mail* e edição de documentos para grupos desconectados.

No sistema *Bayou*, cada banco de dados é inteiramente replicado em um conjunto de servidores. Ele não provê suporte à replicação transparente para as aplicações que executam na camada superior. As aplicações têm o conhecimento de que podem ler dados

inconsistentes, e que suas escritas são uma tentativa. Entretanto, o conhecimento específico do domínio da aplicação é explorado com a finalidade de detectar e solucionar conflitos de forma automática.

Este sistema suporta um esquema de replicação *read-any/write-any*. Quando um cliente submete uma operação de escrita a um determinado servidor, este executa a escrita imediatamente em sua réplica, sem qualquer processo de detecção de conflitos. As escritas que foram aceitas localmente são chamadas de tentativas (*tentative*). O valor resultante de uma tentativa de escrita (*tentative write*) é imediatamente disponibilizado para leitura. Os servidores propagam as escritas entre si através de um processo baseado na comunicação entre pares, denominado sessão de anti-entropia (*anti-entropy*). Durante uma sessão, os dois servidores envolvidos trocam suas escritas, e então, ao final do processo, os servidores entram em um consenso sobre que escritas devem executar e em que ordem. O sistema garante que todos os servidores irão eventualmente receber todas as escritas e que dois servidores que receberam o mesmo conjunto de escritas irão apresentar o mesmo estado (conteúdo) do banco de dados. Para alcançar este objetivo, é necessário que as escritas sejam executadas na mesma ordem global em todos os servidores e que os procedimentos para detecção e resolução de conflitos sejam determinísticos.

Uma vez que os servidores podem receber as operações de escrita dos clientes e dos outros servidores em uma ordem diferente da ordem global, os servidores podem necessitar desfazer os efeitos de uma tentativa de escrita (*tentative write*) previamente executada e executá-la novamente em outra ordem. Assim, uma dada escrita pode ser executada diversas vezes em um mesmo servidor. Uma escrita se torna estável quando o servidor recebe e executa todas as escritas que a precedem. A fim de aumentar a taxa com que as escritas se tornam estáveis, *Bayou* usa um esquema denominado *primary-commit*: um servidor designado

como primário tem a responsabilidade de executar o *commit* das atualizações, ou seja, torná-las permanentes. As escritas já consolidadas (*committed writes*) são ordenadas de acordo com o momento em que elas são consolidadas no servidor primário, sendo ordenadas antes das tentativas de escrita. A informação de que escritas foram consolidadas (e em que ordem) são propagadas para os demais servidores durante o processo denominado anti-entropia. Cada servidor mantém duas visões do banco de dados: uma cópia que reflete somente os dados consolidados (*committed*) e outra cópia que reflete as tentativas de escrita conhecidas até o momento pelo servidor.

O sistema *Bayou* provê um método automático baseado em validação de dependências para detectar conflitos e procedimentos para a resolução automática de conflitos. As validações de dependências e os procedimentos para resolução de conflitos são incluídos juntos a cada operação de escrita. A validação de dependência consiste de uma consulta e o resultado esperado. Antes da operação de escrita ser executada no servidor, a consulta que compõe a validação de dependência é executada sobre a cópia corrente do banco de dados. Um conflito é detectado se a consulta não recupera o resultado esperado. Neste caso, a atualização requisitada não é executada e o servidor invoca o procedimento para resolver o conflito detectado. Este procedimento é um programa escrito em uma linguagem interpretada de alto nível, o qual é responsável por solucionar conflitos e produzir uma atualização modificada que será executada. Quando a resolução automática não é possível, o procedimento de resolução de conflito deve armazenar esta informação em um *log*, o qual será utilizado para a resolução manual de conflitos. *Bayou* permite que as réplicas permaneçam acessíveis mesmo quando conflitos são detectados e ainda não resolvidos. Isto permite que os clientes continuem suas operações, mas pode levar à criação de conflitos em cascata.

3.2 Considerações Finais

Nesse capítulo foram apresentadas as principais soluções estudadas que aplicam os conceitos descritos no capítulo 2. Essas aplicações servem de base para a construção da proposta do modelo de replicação, bem como da solução que utiliza este modelo para localizar informações em ambientes P2P com mobilidade.

A seção 3.1 apresentou algumas soluções para replicação em ambientes P2P. Essas soluções apresentam aplicações práticas que utilizam as diversas combinações de formas de replicação apresentadas no capítulo 2. O estudo dessas soluções servem de insumo para a proposta apresentada no capítulo 4.

Nos próximos capítulo apresentamos o modelo de replicação (capítulo 4) e sua avaliação (capítulo **Erro! Fonte de referência não encontrada.**).

4 Modelo de Replicação Múltiplos-Únicos Mestres

Nossa arquitetura propõe usar replicação de dados com o objetivo de localizá-los em um ambiente P2P com mobilidade. Assim, os nós estão constantemente deixando e retornando ao ambiente, bem como se movem dentro da rede. Com isso, manter informações com alta disponibilidade e garantir a consistência dessas informações tornam-se desafios dessa proposta.

O modelo proposto neste trabalho é denominado modelo múltiplos únicos mestres. Seu objetivo principal é utilizar as características positivas do modelo múltiplos mestres, dentre elas a possibilidade de atualização em qualquer nó da rede, e o mecanismo simplificado do modelo único mestre, onde a garantia de consistência é forte e serializável.

O modelo visa aumentar a disponibilidade dos dados utilizando a replicação para permitir que mais nós tenham acesso às informações, e a garantia de consistência para permitir que as informações disponíveis sejam sempre as mais atuais. Para tal, é necessário que as informações estejam disponíveis e atualizadas entre os nós que as compartilham na rede. Essa disponibilização traz alguns impactos, pois em ambientes distribuídos, como no caso proposto, quanto maior o número de nós mais difícil será o gerenciamento para garantir a consistência.

Vislumbrando este problema e também percebendo que, de fato, nem todos os nós serão capazes de armazenar informações replicadas, criamos um mecanismo que seleciona alguns nós da rede (baseado em um critério determinado – espaço de armazenamento, no caso) para que estes formem uma rede de replicação (Figura 4.1). Assim, os nós que não possuem capacidade de armazenamento de informações replicadas se conectam a esses nós com o objetivo de disponibilizar suas informações na rede. De fato, cria-se uma rede de super-

nós lógica, onde os nós com espaço de armazenamento para replicação tornam-se super-nós dos nós que não possuem tal característica, como celulares e PDAs por exemplo.

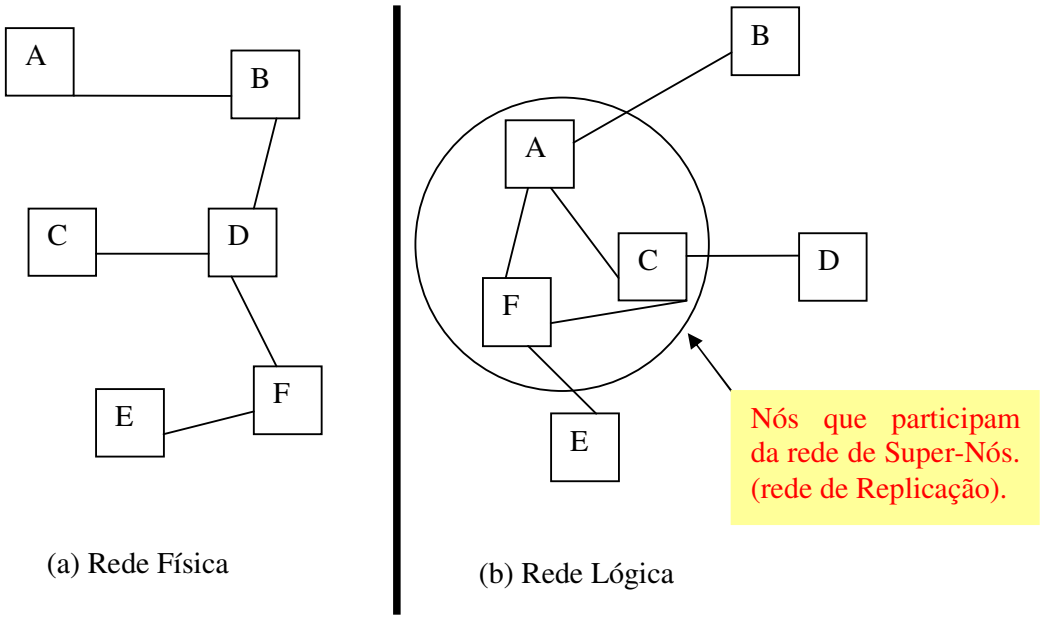

Figura 4.1 (a) Arquitetura da rede física e (b) Arquitetura da rede de replicação.

Na Figura 4.1(a) uma rede P2P não estruturada é apresentada. Após os nós iniciarem o modelo de replicação, uma rede de replicação é formada entre os nós A, C e F Figura 4.1(b). Os nós participantes da rede de replicação trocam informações entre si.

Nossa abordagem prevê uma mesclagem dos modelos de único mestre e múltiplos mestres. Ele se caracteriza por permitir que existam vários mestres para conjuntos distintos de objetos, ao contrário do modelo de único mestre, onde há apenas um mestre para todos os objetos, e do modelo múltiplos mestres, onde todos os nós são mestres de todos os objetos. Assim, o objetivo dessa abordagem é criar um modelo de replicação denominado ***múltiplos únicos mestres*** que possui as seguintes características: (a) permite atualizações concorrentes através da adaptação do modelo de único mestre, evitando problemas referentes a essas atualizações, mas mantendo simples o controle de concorrência; (b) possui um mecanismo de

propagação das atualizações nas cópias principais para as cópias escravas, com objetivo de garantir que sempre existirá um nó com o objeto mais atual e evitar a sobrecarga de mensagens de sincronização das réplicas na rede; (c) permite a escolha de um novo mestre em caso de falha do nó mestre do objeto, evitando assim os pontos únicos de falha que causam gargalos no sistema.

Um nó será mestre de um conjunto de objetos se esses forem criados por ele e será escravo de informações que ele tenha recebido por meio de replicação. Assim, um mesmo nó assume papel de mestre e escravo dependendo de qual objeto este nó esteja se referindo. Para o conjunto de nós de uma rede, representado por *N*, podemos afirmar que todo objeto criado por um nó *n*, com $n \in N$, será nele uma cópia primária e terá *n* como o mestre desse objeto.

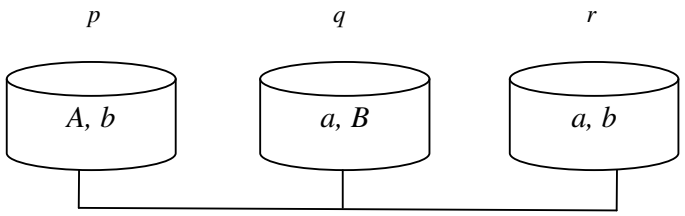

Figura 4.2 - Representação do modelo múltiplos únicos mestres;*p, q e r* são nós que armazenam objetos *a* e *b*, onde *A* é cópia principal, *a* é réplica e *B* é cópia principal enquanto *b* é a réplica.

A Figura 4.2 apresenta um exemplo de uma rede com seus mestres e escravos. Dessa forma, para *N* representando o conjunto de nós da rede e *O* o conjunto de objetos da rede temos $N = \{p,q,r\}$ e $O = \{a,b,A,B\}$ onde *A* e *B* são cópias primárias e *a* e *b* são réplicas de *A* e *B* respectivamente. Assim, *p* trabalha como um nó mestre quando alguém se refere ao objeto *A* e, ao mesmo tempo, trabalha como nó escravo do objeto *b* enquanto *q* é o nó mestre do objeto *B* e é um nó escravo do objeto *a*. O nó *r* por sua vez é um nó escravo de ambos os objetos. Reforçando, podemos ter vários escravos para um mesmo objeto, mas em nenhuma hipótese podemos ter mais de um mestre para um mesmo objeto.

4.1 Atualização das Réplicas

Seja $n \in N$, onde N é o conjunto de nós da rede. Seja DE_n o conjunto de objetos existentes em n tal que $DE_n = \{D_n, DR_n\}$, onde D_n é o conjunto de dados criados por n, dos quais diz-se que n é mestre e DR_n é o conjunto de réplicas armazenadas em n, das quais diz-se que n é escravo. Definimos que qualquer nó que deseje alterar um objeto em D_n precisa solicitar autorização a n por essa alteração, da mesma forma que qualquer réplica existente em DR_n exigirá de n uma solicitação de atualização para esta réplica a qual será enviado ao mestre da mesma. Os objetos existentes em D_n podem ser atualizados por n sem solicitação ou aviso de atualização para as cópias. Exemplificando, seja $N=\{p,q\}$, $D_p=\{A\}$, $DR_p=\{b\}$, $D_q=\{B\}$ e $DR_q=\{a\}$, para que p possa alterar o objeto b é necessária uma solicitação de atualização a q, da mesma forma que, para que q possa alterar o objeto a ele precisa de uma autorização de p[1].

Para que uma réplica possa ser atualizada por um nó escravo, é necessário que este execute os seguintes passos:

a. Antes de atualizar a réplica, o nó escravo (solicitante) deve se comunicar com o nó mestre daquela réplica, solicitando uma cópia do objeto em seu estado mais atual;
b. Uma vez de posse da réplica mais atual o nó solicitante informa ao nó mestre que deseja alterar a réplica;
c. O nó mestre retorna uma mensagem ao nó escravo que indica uma permissão de alteração. Essa mensagem contém um conjunto de identificadores referentes à versão e à transação a ser realizada (esses identificadores são explicados posteriormente);

[1] *N*ote que *A* e *a* cópias de um mesmo objeto. Adotamos letras maiúsculas para denotar cópias primárias de objetos, e letras minúsculas para denotar cópias sedundárias.

d. Após a alteração, o nó deve devolver ao mestre o objeto alterado, solicitando a ele que finalize a operação;
e. Nesse momento o mestre valida se é possível ou não efetivar (*commit*) a alteração (essa validação é feita através da verificação dos identificadores referentes à versão e à transação). Caso seja possível, o mestre finaliza a operação, caso contrário é gerada uma exceção. Em ambos os casos mensagens de confirmação são enviadas ao nó solicitante da alteração.

Após uma atualização bem sucedida, as demais réplicas não são avisadas da alteração. Por isso, é possível afirmar que réplicas podem estar em estados diferentes em algum instante de tempo. No entanto, sempre antes de uma solicitação de alteração, elas serão sincronizadas com o nó mestre. Isso ocorre porque cada nó é responsável por sua cópia principal, por isso uma das premissas do modelo apresentado é que o nó mestre deve sempre ter a versão do objeto mais atual, possibilitando que qualquer outro nó da rede que precise da versão atual solicite a ele. Essa decisão de não avisar as demais réplicas evita o tráfego desnecessário de dados na rede, ou seja, um nó que desejar executar alguma operação sobre um determinado item de dado irá atualizá-lo somente quando necessário.

Exemplificando, suponha uma rede composta por três nós N= {*p, q, r*}, onde D_p= {*A*}, DR_p= {*b, c*}, D_q= {*B*}, DR_q= {*a, c*} e D_r= {*C*}e DR_r= {*a, b*}. Quando o nó *p* deseja atualizar o objeto *b*, ele solicita ao nó *q* que retorne para ele o estado mais atual de *B*. Uma vez de posse do estado mais atual, *p* atualiza o objeto e envia para *q* a versão atualizada. Assim, *q* sincroniza a versão e informa a *p* que a atualização foi feita com sucesso. Nesse momento *r* não foi informado da atualização, mas para que *r* possa alterar o objeto *b* ele deve seguir os mesmos passos realizados por *p*. Assim sendo, *r* deve solicitar a *q* pelo estado mais atual de *B*. Nesse momento, as atualizações de *p* serão visualizadas por *r* para que o mesmo possa alterar a informação.

Em alguns casos não é possível aguardar a atualização do objeto pelo algoritmo de replicação e é necessário ter a versão mais atual de um objeto *o* em um nó escravo qualquer. Isso pode ocorrer em situações onde, por exemplo, uma consulta crítica é feita para uma tomada de decisão ou em situações onde passou-se muito tempo sem que este nó tenha atualizado o objeto *o* (e portanto possui uma versão obsoleta de *o*). Essas situações são caracterizadas por momentos onde apenas a informação mais atual garante a validade de outra operação ou da tomada de uma decisão, e são consideradas como uma exceção ao ambiente de atuação proposto pelo modelo. Em tais situações, nossa abordagem prevê que o nó escravo consulte a informação do nó mestre sem que seja necessário solicitar uma atualização.

4.2 Consistência dos Dados

A replicação tem por objetivo prover uma alta disponibilidade dos dados. Em contrapartida podemos ter diversas cópias em estados diferentes, pelo motivo apresentado acima. Para resolver a consistência da informação de forma que a réplica a ser alterada seja sempre a mais atual, propomos que cada cópia principal e, por conseguinte suas réplicas possuam um identificador de versão, um identificador de transação e um identificador de mestre representado por *C(V, T, M)*, onde *V* é o identificador de versão, *T* é o identificador de transação e *M* é o identificador do mestre. Esses identificadores são incorporados à cópia principal e são enviados para os nós que recebem sua réplica.

Dessa forma, uma cópia principal *o* é composta pelos seguintes atributos: *o*={*chave, dados, C(V, T, M)*}, onde *chave* é o identificador da cópia principal *o* e *dados* encapsula o objeto representado por *o* (é o conjunto de pares atributo/valor com as informações deste objeto). Quando um nó *n* cria um objeto *o*, os valores referentes *V*, *T* e *M* em *C(V, T, M)* são tais que *C(V, T, M)=(0, 0, n)*. Essa informação indica que a cópia original ainda não sofreu nenhuma alteração.

A cada solicitação de alteração o identificador de transação é incrementado em uma unidade no objeto *o* e repassado para a réplica solicitante da atualização. Assim *C(V, T, M)=(v, t+1, n)*, onde *v* é o valor atual do identificador de versão e *t* é valor do identificador de transação anterior à solicitação. Incrementar o identificador de transação possibilita solicitações de atualização simultâneas do objeto sem a necessidade de bloqueá-lo para um único solicitante.

Quando uma transação é finalizada (*commit*), uma nova versão é gerada na cópia principal de forma que *C(V,T,M)=(v+1, 0, n)*, onde *v* é o valor da versão anterior, que será primeiro alterado na cópia principal e após encaminhado para a réplica que solicitou a atualização. Nesse momento *T* tem seu valor zerado, pois uma nova versão foi gerada, descartando as solicitações de atualização antigas.

A estratégia de utilizar o identificador de transação visa atingir dois objetivos: (1) permitir que mais de um nó possa solicitar a atualização de um mesmo objeto, sem bloqueio (*lock*); e (2) garantir que apenas o nó que solicitou a atualização possa confirmá-la, impedindo que um nó com a informação desatualizada atualize o objeto. Permitir atualizações concorrentes sem bloqueio é muito importante em ambientes onde há grande mobilidade e grande possibilidade de conexão e desconexão dos nós, pois evita que o objeto fique bloqueado para um nó que se desconectou da rede.

Quando um nó *n* ausenta-se da rede (seja por desconexão ou falha) este passa a não ser mais mestre das informações criadas por ele, desde que estas já tenham sido replicadas para os demais nós. Caso isso não tenha ocorrido, ele permanece como mestre das informações. Alguns problemas podem ocorrer em função dessa decisão. Tais problemas são explicados na próxima seção.

4.3 Tratando problemas de Sincronizando na Atualização

Uma vez criada uma cópia principal e suas réplicas encaminhadas para os demais nós, alguns problemas de sincronização podem ocorrer entre as réplicas e a cópia principal durante uma atualização. Esta seção descreve alguns destes problemas e as soluções propostas.

a) *O nó mestre se desconecta (saída programada) ou falha (saída abrupta).*

Se um nó mestre *n* deseja se desconectar, antes de executar esta ação ele encaminha para *r*, um dos seus vizinhos selecionado aleatoriamente, todos os objetos dos quais ele é mestre e solicita que *r* se torne mestre das cópias principais. O nó *n* também replica as informações apontando para o novo mestre *r*.

Se o nó mestre *p* falha, essa falha só será sentida por um nó que desejar atualizar ou consultar uma réplica de uma informação da qual *p* é mestre. Caso isso aconteça, o nó que deseja atualizar a informação deve executar um *algoritmo de busca de novo mestre*. Caso nenhum nó responda, o nó deve executar o *algoritmo de escolha de novo mestre.* Ambos os algoritmos serão explicados mais adiante.

Quando um nó mestre *p* volta à rede após uma falha, ele deve verificar se os objetos dos quais ele era mestre possuem um novo mestre. Para tal, ele deve executar o *algoritmo de busca de novo mestre* para cada um dos objetos dos quais ele era mestre antes da falha. Para cada resposta negativa do algoritmo, ele se nomeia novamente como mestre. Caso algum outro nó tenha assumido como mestre, *p* passa a apontar para este nó como novo mestre do objeto em questão. Esta abordagem é mais segura, pois evita que algum nó com um objeto *o* desatualizado assuma como mestre e permaneça por muito tempo com *o* desatualizado (mais detalhes na seção 4.4). Porém, a abordagem possui o problema da necessidade de validar todos os objetos a cada falha. Se o nó possuir uma grande quantidade de cópias principais essa

verificação pode gerar um grande número de mensagens trafegadas na rede causando perda de desempenho.

b) *O nó que solicitou a atualização se desconecta (saída programada) ou falha (saída abrupta) antes de publicar sua atualização.*

Se um nó *p* se desconecta após ter solicitado uma atualização a um nó mestre *n*, a transação é abortada e não é necessário avisar ao mestre. Se ele falha, ao tentar voltar, este deve reenviar o pedido de atualização ao mestre. O mestre recebe o pedido e verifica o identificador de versão e transação. Se esse tiver sido alterado, a atualização é descartada e o nó deve solicitar novamente a versão mais atual para aplicar a atualização. Caso contrário, a atualização é aplicada no mestre normalmente. Se o mestre houver mudado, o nó deve executar o algoritmo de *busca do novo mestre* e enviar para ele sua solicitação.

4.3.1 Algoritmo de Busca de Novo Mestre

Quando um nó *p* deseja atualizar um objeto do qual o nó *q* é mestre e *p* percebe que *q* está ausente da rede, *p* deve executar o *algoritmo de busca de novo mestre.* Esse algoritmo permite que *p* possa encontrar o novo mestre do objeto. A proposta do algoritmo é enviar uma mensagem para os nós vizinhos questionando se os mesmos são os nós mestres de um determinado objeto. Se algum nó mestre do objeto for encontrado, o valor do novo mestre do objeto é setado e um valor booleano é retornado indicando se a busca foi bem sucedida (*Verdadeiro*) ou não (*Falso*).

A Figura 4.3 apresenta o algoritmo para busca do novo mestre. Para execução do algoritmo são necessárias as seguintes informações: o identificador do objeto, que é representado pelo parâmetro *chave* e o parâmetro *C(V, T, M)* que representa o identificador de versão (*V*), de transação (*T*) e o mestre referenciado pelo objeto atualmente (*M*).

Para buscar o novo mestre, o algoritmo chama a função *SolicitaNovoMestre* passando a chave do objeto como parâmetro. Essa função tem como objetivo retornar o *C(V, T, M)* do novo mestre para que o solicitante possa atualizar o seu *M* (parâmetro que indica o endereço atual do mestre desse objeto). A solicitação é enviada para os vizinhos através de um algoritmo de consulta de informações em ambientes ponto-a-ponto (P2P) tal como *flooding* (Genç, 2005) ou *gossiping* (Datta & Aberer, 2004; Modiano, Shah & Zussman, 2006; Nandy, Carter & Ferrante, 2005; Genç, 2005). Uma vez terminada a execução da função, seu valor de retorno é armazenado na variável *valC*.

```
encontrarNovoMestre(chave,C(V,T,M))
Inicio
 /* Essa função retorna um objeto que é o novo mestre. Para uma chave,
  * valC retorna o C(V,T,M) do novo Mestre. Se não for encontrado o novo mestre ele
   * retorna Falso.
  */
  valC:=SolicitaNovoMestre (chave)

  Se valC <> VAZIO então
  inicio
     C.M := valC.M
     Retorna Verdadeiro
  Senao
      Retorna Falso
  Fim se
fim
```

Figura 4.3 - Algoritmo para encontrar Novo Mestre

O valor contido em *valC* será o *C(V,T,M)* do mestre encontrado ou *VAZIO,* que representa a ausência de um mestre para determinada informação. Se *valC* for diferente de *VAZIO*, o solicitante atualiza o valor do *M* em seu objeto (representado por *C.M*, que indica o valor de *M* em *C(V, T, M)*) e a função retorna *Verdadeiro*, caso contrário a função retorna *Falso*.

Se o retorno da função for *Falso*, isso indica que nenhum mestre foi encontrado para o objeto na rede, sendo necessário escolher um novo mestre. Para tal, deve-se executar o *algoritmo de escolha de novo mestre* apresentado na próxima seção.

4.3.2 Algoritmo de Escolha de Novo Mestre

O algoritmo de escolha de novo mestre, é um algoritmo de eleição (Paris e Long 1988; Jajodia e Mutchler 1990), que tem por objetivo eleger um novo mestre para a cópia principal no caso da ausência do mesmo. Para isso, ele utiliza o algoritmo de replicação dos objetos enviando um tipo de objeto especial. Esse objeto representa uma solicitação de novo mestre. Ele é composto por três parâmetros: <*chave, C(V,T,M), n*>, onde *chave* é o identificador do objeto, *n* é o nó que está enviando a solicitação e *C(V,T,M)* é a versão da informação constante em *n* no momento em que a requisição é enviada. Cada nó, ao receber esta requisição, verifica se possui a réplica (comparando o valor da chave) e compara os valores de *V* e *T* recebidos com os seus. Se os valores de *V* e *T* do nó forem maiores que o de *n* seguindo a *Regra R*1 abaixo, este deve enviar uma mensagem para *n* indicando quem ele é e o seu *V* e *T*. Ao final, *n* verifica quem possui o maior *V* e *T* (na respectiva ordem) e envia uma mensagem para este nó indicando-o como o novo mestre e envia uma mensagem para os outros nós indicando quem é o novo mestre.

Regra R1: Dado um objeto *o* replicado nos nós *p* e *n* com valores $C(V_p, T_p, M)$ no nó *p*, e $C(V_n, T_n, M)$ no nó *n*, diz-se que *p* tem a versão mais atual de *o* se *(Vp > Vn) ou (Vp=Vn e Tp>Tn).*

Se o novo mestre sair da rede (falhar ou se desconectar) no momento de ser nomeado como novo mestre de uma informação *o*, o nó solicitante escolherá o próximo nó com o maior *V* e *T* até que um seja escolhido e nomeado. Se o nó escolhido for nomeado e depois sair da rede, o nó solicitante reiniciará novamente o processo de escolha de novo mestre. Se nenhum nó tiver *V* e *T* maior que o do solicitante ou se todos os que possuírem os maiores *V* e *T*

falharem no momento da nomeação, o nó solicitante assumirá como novo mestre da informação. A Figura 4.4 apresenta o algoritmo.

A probabilidade mínima de sucesso na busca por um novo mestre (P_s) pode ser expressa por uma função de probabilidade binomial dada por $P_s = 1 - P_f$, onde P_f é a probabilidade de fracasso na busca de novo mestre em todos os nós que possuem a cópia mais atual. Deduzindo P_f , dizemos que a probabilidade de fracasso na busca de um novo mestre por todos os nós com a cópia mais atual pode ser dada por $P_f = (1 - P_{pt})^H$, onde P_{pt} é a probabilidade de um nó *p* estar ativo num instante *t* após a falha do mestre e *H* o conjunto de nós vizinhos do nó mestre que possuem a cópia mais atual do objeto. Assim, seja *o* um objeto presente no nó *q* do qual *q* é mestre. Seja P_{pt} a probabilidade de um nó *p* estar ativo no instante de tempo *t* e seja *H* o conjunto de nós vizinhos do nó *q* que possuem a réplica do objeto *o*. Podemos afirmar que:

$$P_s = 1 - (1 - P_{pt})^H$$

Essa regra vale se levarmos em consideração que o valor de P_{pt} , ou seja, a probabilidade de um nó *p* estar conectado num instante de tempo *t,* seja igual para todos os nós da rede.

Exemplificando, suponha que $H = 4$, ou seja, existem quatro nós vizinhos de *q* com a réplica de *o* e que a probabilidade dos nós estarem ativos em um determinado instante seja de 25% ou 0,25. Podemos afirmar que a probabilidade de se encontrar um desses nós ativos para que este seja eleito o novo mestre para o objeto *o* na ausência de *q* será: $P_s = (1 - (1 - 0{,}25)^4 = 68\ \%$. Assim, podemos afirmar que há 68% de chances de que um dos 4 nós que contém a réplica de *o* esteja ativo no momento da ausência de *q*, para uma probabilidade de 75 % de falha dos nós.

```
Escolher_Novo_Mestre(chave,C(V,T,M),n)
Inicio
/*Esse algoritmo busca o novo mestre para a chave, se não encontrar executa a operação
*abaixo.
*/
Se não encontrarNovoMestre(chave,C(V,T,M)) então
Inicio
 /* Essa função retorna uma coleção de objetos, onde cada objeto contem
  * os valores de V e T (versão do objeto em p ) e o próprio p que é Nó.
  * Essa coleção é composta pelos nós que possuem o V e T maiores que o de n.
  */
 colC:=SolicitaMaioresReplicas(chave,C(V,T,M),n)

 /* Ordena a coleção pela ordem decrescente
 *de V e T seguindo a Regra R1
 */
 colOrdenada := ordenaColecao(colC)

 Aceitou:=false;

 Para cada objordenado ϵ colOrdenada faça
  Inicio
    Se não (aceitou) então
          Inicio
           //Nomeia o nó com o maior V e T
          Aceitou:=NomeiaMestre(chave,objordenado.C,objordenado.N)
          Fim se
   Fim para
  //Se nenhum nó foi nomeado então N se nomeia como mestre
   Se não(aceitou) então
    Inicio
          Aceitou:= NomeiaMestre(chave,C(V,T,M),n)
    Fim se
Fim Se
Fim
```

Figura 4.4 - Algoritmo para Nomeação de Novo Mestre.

Nessa abordagem é possível que mais de um nó inicie a busca por um novo mestre e, em algum instante de tempo, ambos estejam procurando pelo mestre de um mesmo objeto. Essa possibilidade não é encarada como problema, pois ambos encontrarão o mesmo mestre. Por trabalharmos em um ambiente onde partimos do princípio que o particionamento da rede não acontece, problemas referentes a essa situação não são tratados por esse algoritmo nesse trabalho, sendo abordados como melhorias para trabalhos futuros.

Um problema ocorre quando todos os nós que contêm a cópia mais atual do objeto estão ausentes. Neste caso, o novo mestre será escolhido baseado em uma versão antiga do objeto. Para tentar minimizar esse problema propomos uma função de propagação de réplicas.

4.4 Função de Propagação de Réplicas

A abordagem proposta neste trabalho permite que se tenha, em algum determinado instante de tempo, a cópia principal e todas as réplicas com uma versão desatualizada do objeto. Isso pode ocorrer quando o nó mestre e todos os nós que possuem a réplica mais atual estão fora da rede no momento da escolha de um novo mestre. Com o objetivo de minimizar tal problema é necessário disponibilizar a cópia principal atual (cópia principal que contenha o identificador de versão do objeto com o maior valor) em um maior número de réplicas possíveis.

Disponibilizar novas réplicas implica em mais processamento e tráfego na rede. Com o objetivo de contornar este problema, propomos que essa replicação seja feita sob demanda de atualização, permitindo copiar apenas os dados que forem mais atualizados e à medida que sofram atualizações.

Para evitar a ausência do dado na versão mais atual da rede propomos a *função de propagação de réplicas*. Essa função é atrelada à cópia principal e será executada a cada atualização dela. Ela é uma função matemática, definida pelo usuário ao criar a cópia primária, que retorna o número de cópias a serem feitas do objeto em questão. Assim, é possível que objetos com maior quantidade de atualizações sejam replicados mais vezes para outros nós e evita a replicação desnecessária de objetos pouco atualizados.

Por exemplo, objetos de banco de dados que são altamente acessados precisam ter o maior número de réplicas atuais possíveis, enquanto objetos que representam informações sobre serviços (como WSDL que não mudam muito) podem ter um número de réplicas menor. Dessa forma é possível criar *níveis de replicação*, onde estes expressem a importância de manter o objeto atualizado.

A *função de propagação de réplicas* retorna sempre o número de réplicas que devem ser feitas e enviadas para os vizinhos do nó mestre, no momento em que a cópia principal é atualizada. Os nós vizinhos que receberão a réplica da cópia original alterada serão sempre diferentes do nó que solicitou a alteração. Essa função pode variar de cópia principal para cópia principal em um mesmo nó e uma vez inserida pode ser alterada somente pelo mestre.

Para exemplificar de forma simples a funcionalidade da função, suponha uma *função de propagação de replicas f(v)=v*, onde *v* é o número do identificador de versão. A Figura 4.5 mostra a rede e o objeto presente nos nós após a atualização.

Na Figura 4.5, *p, q, r, s* são os nós da rede e a informação armazenada neles é representada pela figura do quadrado. Para facilitar o entendimento, dentro do quadrado estão as informações de número de versão (*V*) e número de transação (*T*) do objeto. No exemplo, *p* é o mestre do objeto e ele mesmo faz a atualização da informação (quadrado). O nó *p* finaliza a atualização passando a ficar com a informação mais atual. Neste momento, o objeto quadrado fica com versão 1 (um) e transação 0 (zero), conforme ilustrado na figura. Após a finalização da atualização, *p* executa a *função de propagação de réplicas,* e sabe que precisa propagar a atualização para 1 nó da rede. O nó *q* é escolhido aleatoriamente, e recebe a nova versão da informação.

Nesse exemplo, sem a *função de propagação de replicas*, a probabilidade da rede ficar sem a informação atual é de 75%, enquanto que com a função ela cai para 50% na primeira atualização. A partir da segunda atualização a probabilidade é de 25% e na terceira atualização 0% se a cada atualização for escolhido um nó diferente.

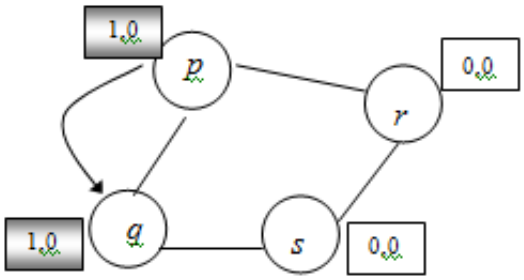

Figura 4.5 - Exemplo de rede onde a função de replicação é aplicada.

Voltando ao exemplo, se após a execução da função de propagação e a atualização do nó *q*, os nós *p* e *q* caírem, a rede ficará sem a informação atual. Digamos que *r* assuma como mestre do objeto quadrado. Como demonstrado na figura, *r* está com uma versão desatualizada do objeto, assim sendo é necessário que *r* saiba dessa defasagem quando *p* ou *q* retornem à rede. Em (Akbarinia, Pacitti & Valduriez, 2007) é proposta uma solução que checa se os valores da versão (*V*) e da transação (*T*) do objeto em *r* foram alterados ou não e decide a ação a ser tomada. Baseado na solução descrita em (Akbarinia, Pacitti & Valduriez, 2007) e no exemplo acima, apresentamos três situações possíveis:

a. Somente *p* retorna à rede – *p* executa o algoritmo de *busca de novo mestre* e ao descobrir *r* como o novo mestre, ele pede a *r* seus dados atuais. Se os parâmetros *V* e *T* de *r* forem menores que os de *p*, *p* então atualiza *r* com os seus dados e seus valores de *V* e *T*. Se, por outro lado, os valores forem maiores, *p* se atualiza com os dados de *r* (note que este é um caso patológico);

b. Somente *q* retorna à rede – quando *q* solicitar uma atualização a *p*, ele descobrirá que *p* está fora da rede. Nesse momento, *p* executa o algoritmo de *busca de novo mestre*, o qual descobrirá *r*. Assim, *q* solicita a *r* por sua versão mais atual do objeto. Se os parâmetros *V* e *T* de *r* forem menores que os de *q*, então *q* atualiza os dados e os valores de *V* e *T* de *r*, caso contrário *q* atualiza apenas os dados com o valor de *r* .

c. Se *q* retorna após *p* ter retornado, ele verificará que *p* não é mais o mestre e executará os passos descritos em (b).

É importante ressaltar que só serão replicados os dados que forem atualizados e que estes não serão replicados para todos os nós. Dessa forma diminuímos o tráfego de dados na rede e aumentamos a probabilidade de que algum nó terá a informação mais atual na rede.

Para definir uma boa função de propagação de réplicas, é necessário definir uma forma de calcular a quantidade mínima de vizinhos que devem ser atualizados, de forma a garantir um percentual mínimo de sucesso na busca de um novo mestre. Seja *o* um objeto do qual o nodo *q* é mestre e C_q o conjunto de vizinhos do nó *q*. Seja ainda P_c a probabilidade mínima de garantia de réplica, ou seja, a probabilidade de garantia de que a cópia mais atual de *o* será encontrada após uma atualização. Supondo que pelo menos um vizinho de *q* não irá falhar na ausência do mesmo, podemos, usando uma regra de três simples, afirmar que:

$Qc_o = (P_c * C_q) / 100$, onde Qc_o é a quantidade de cópias do objeto *o*.

Este cálculo parte do princípio de que pelo menos um vizinho de *q* não falhará e que se atualizarmos todos os nós com a cópia mais atual podemos ter 100% de garantia que numa falha do nó mestre e de seus vizinhos, pelo menos um nó estará ativo e este conterá a cópia principal no estado mais atual. No caso, se quisermos variar este percentual precisamos alterar o valor de P_c, o que nos dará um novo valor de vizinhos que devem ser atualizados para atingir este percentual.

Exemplificando, seja *q* o mestre de um objeto *o* e $C_q = 10$ e $P_c = 25$ %. Neste caso, temos $Qc_o = (25 * 10) /100 \approx 3$. Assim, para podermos garantir no mínimo que 25% dos nós conterão o objeto mais atualizado na rede, teremos que aplicar a replicação para 3 nós, quando *q* sofrer a primeira atualização. Para elevar esse valor gradualmente à medida que o objeto *o* sofre novas atualizações propomos a seguinte *função de propagação de réplica*:

$$FPR = Qc_o + (Log_{Qco}V)$$

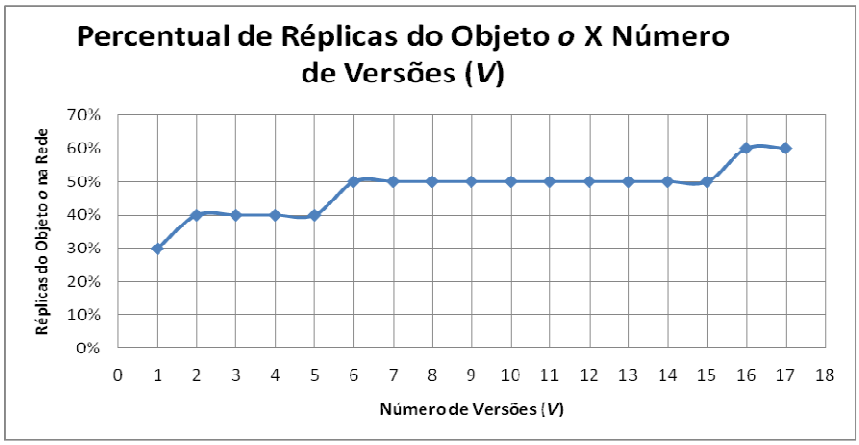

Figura 4.6 – Percentual de vizinhos que terão a réplica mais atual de um objeto *o* à medida que o número de atualizações aumenta.

Nesta função, quanto maior o número de atualizações, maior será o percentual de nós que conterão o dado mais atual. A utilização da função $Log_{Qco}V$ visa balancear a função de forma que: (a) se o número de Qco for alto, então, um percentual alto de nós vizinhos receberão a cópia mais atual logo na primeira atualização e a função Log forçará um número múltiplo de Qco, ou seja, um número de atualizações alto para que mais nós recebam a cópia mais atual; (b) se o número de Qco for baixo, então, um percentual baixo de nós vizinhos serão atualizados com a cópia mais atual do objeto logo na primeira atualização e a função Log fará que com um número de atualizações baixo, mais nós recebam a cópia mais atual. Assim, com poucas atualizações o percentual de nós vizinhos com a cópia mais atual do objeto aumentará, conforme Figura 4.6.

O gráfico da Figura 4.6 mostra o arredondamento do valor da função, para valores de versão de 1 a 17, para os valores dos parâmetros apresentados no exemplo. O gráfico mostra o crescimento dos valores da função de propagação em função do número de atualizações de um determinado objeto. Note que o número da versão reflete o número de atualizações feitas em um determinado objeto.

4.5 Comparativo

A Tabela 2 apresenta um comparativo entre os modelos de replicação apresentados e a proposta deste trabalho. Os requisitos para expressar o comparativo são os seguintes:

Autonomia – O usuário que colabora deve ser capaz de armazenar localmente os dados que desejar. Isso habilita colaboração assíncrona independente de desconexão ou falha do sistema.

Tipo de Replicação – Identifica a forma como as atualizações ocorrem. Único Mestre - apenas no nó mestre; Múltiplos mestres – Todos os nós podem atualizar; ou Múltiplos Únicos Mestres – Somente o nó mestre do objeto e a cópia que solicitou a atualização participam da transação de atualização.

Atualização das Cópias – Indica no momento de uma atualização da cópia primária como as cópias secundárias são atualizadas. A indicação "atualização individual", significa que cada cópia faz sua atualização independente das demais, nesse caso todas as cópias são tidas como primárias.

Tratamento de Falha do Nó Mestre – Uma vez que o nó que possui a cópia principal falha, como o modelo de replicação trata essa falha? Quando o modelo é múltiplo mestre este tipo de tratamento não precisa ser feito, pois cada nó tem autonomia para atualizar todas as informações.

Garantia de Consistência – Indica a forma de garantir a consistência dos dados para conjuntos de atualizações sucessivos da cópia principal e das réplicas. No caso da Eventual, réplicas podem divergir por algum tempo, mas diversas e contínuas reconciliações fazem com que o nível de divergência diminua. Assim, se as réplicas de um objeto param de receber atualização, eventualmente em algum instante elas chegarão a um mesmo estado final. A garantia de consistência probabilística é feita pela verificação do quorum mínimo de nós que

possuem a versão mais atual do objeto. Quando isso ocorre, indica-se que aquela versão é a mais atual.

Tabela 2- Comparativo das abordagens de replicação

Sistema P2P	**Tipo de Rede**	**Autonomia**	**Tipo de Replicação**	**Atualização das Cópias**	**Tratamento de Falhas do Nó Mestre**	**Garantia de Consistência**
Budhiraja *et al.* 1993	P2P Estruturada	Baixa	Único mestre	Atualiza todas as cópias	Seleciona um dos Servidores de BackUp	Seqüencial
Tanenbaum e Steen 2007	P2P Estruturada	Baixa	Único Mestre	Atualiza todas as cópias .	Seleciona um dos Servidores de BackUp	Seqüencial
Monteiro *et al.* 2007	P2P não Estruturado	Alta	Múltiplos Mestres	Atualização individual	___	Eventual
Terry *et al.* 1995	P2P não Estruturado	Alta	Múltiplos Mestres	Atualização individual	___	Eventual
Thomas 1979	_____	Alta	Múltiplos Mestres	Atualiza todas as cópias que fazem parte do Quorum	____	Probabilística
Múltiplos-Únicos Mestres	*P2P não Estruturado*	*Alta*	*Múltiplos únicos Mestres*	*Atualiza o mestre, a cópia que solicitou a atualização e os nós da FPR*	*Seleção de novo mestre. FPR permite que um conjunto de nós tenha a versão mais atual .*	*Seqüencial*

Podemos observar que a proposta expressa nesse trabalho tem a vantagem de garantir consistência seqüencial em um ambiente P2P não estruturado, sem ter que bloquear as operações de atualização. Sendo assim, a proposta deste trabalho tem a forma de garantia de consistência tão simples quanto os modelos únicos mestres e com um grau maior de eficiência comparado a alguns dos propostos. Os dois primeiros modelos que trabalham com único mestre obrigam a atualização a ocorrer em todos os servidores (primário e secundário) causando problemas no tempo de resposta ao cliente, pois é necessário aguardar que todos os nós sejam atualizados antes de encaminhar a resposta para o cliente. Os dois seguintes são modelos múltiplos mestre que utilizam garantia de consistência mediante reconciliação e de forma eventual. O modelo baseado em quorum é mais simples para garantir a consistência

porque precisa apenas que o quorum mínimo seja satisfeito, mas ao mesmo tempo é dependente do quorum que em determinada situação pode causar conflitos quando duas solicitações o atingem.

4.6 Considerações Finais

Esse capítulo apresentou o modelo de replicação que visa manter os dados atualizados e consistentes na rede. Como seu objetivo principal é permitir a localização de dados na rede, é necessário garantir que os dados estejam o mais atualizado possível nos nós que participam da replicação. O modelo de múltiplos-únicos mestres visa garantir essa disponibilidade principalmente pela atuação da função de propagação de réplicas.

Ainda apresentamos um comparativo entre nossa abordagem e outras abordagens citadas nesse trabalho mostrando os pontos fortes de nossa abordagem em relação as demais.

No próximo capítulo apresentamos a implementação da Modelo de Replicação, o qual aplica o conteúdo abordado nesse capítulo juntamente com o conceitos apresentado nos capítulos 2 e 3.

5 Avaliação e Estudo de Caso

Nesta seção, apresentamos uma avaliação realizada sobre o Modelo de Replicação e um Estudo de Caso indicando a aplicabilidade da solução proposta.

A avaliação foi feita mediante simulação e leva em consideração o tempo necessário para replicação das informações e localização de novo mestre em caso de falha durante uma atualização. Discutiremos também alguns problemas encontrados durante a realização dessas análises bem como os resultados obtidos ao longo desse processo.

O Estudo de Caso por sua vez apresentará uma Tese de Doutorado em desenvolvimento na COPPE/UFRJ pelo aluno Luiz Gustavo Lourenço Moura. Nela, serviços computacionais são oferecidos com intuito de se aplicar um mercado baseado na lei da oferta e procura permitindo a compra e venda dos mesmos. Como será visto, nossa proposta auxiliará para que a Tese do aluno Luiz Gustavo Lourenço Moura possa atingir seus objetivos.

Vale ressaltar que esse estudo foi escolhido por se tratar de um ambiente de serviços onde é necessário aplicabilidade de garantia de consistência. Assim, o Teseus, mostra-se não apenas uma solução direcionada a dados, mas com reais aplicabilidades a serviços.

Por fim, apresentaremos a conclusão desse capítulo com as considerações sobre os resultados apresentados.

5.1 Avaliação

A avaliação da proposta foi feita mediante simulação e leva em consideração o tempo necessário para disseminação de um dado, para localização de novo mestre em caso de falha durante uma atualização e o percentual de sucesso na localização de um novo mestre quando

um tempo limite é determinado. Discutiremos também alguns problemas encontrados durante a realização dessas análises bem como os resultados obtidos ao longo desse processo.

Para a simulação, geramos a topologia de rede de forma aleatória, seguindo as idéia utilizada por Benevenuto, Júnior, & Almeida (2005). Desta forma, o simulador gera a topologia de rede de forma aleatória a cada simulação. O número de nós categoria A (referenciados aqui como super-nós) corresponde a 10% do total de nós. Os super-nós são eleitos dentre todos os nós no início da simulação.

A conexão entre os super-nós é composta de forma que cada super-nó tenha no máximo 20 vizinhos. Cada nó folha se conecta a um super-nó, conectando-se a outro super-nó em caso de falha do primeiro. Apenas um dado será inserido na rede o qual será replicado para os demais super-nós. Além disso, o TTL (*Time–To-Live*) utilizado para disseminar as informações entre os super-nós é de 7 *hops* e o tempo gasto para comunicação entre dois nós é de 1 segundo .

As métricas utilizadas na simulação serão:

- **Latência da Disseminação**: tempo médio gasto na disseminação do dado, medido a partir do momento em que um dado sai de um super-nó até chegar a outro super-nó.
- **Latência de Consistência:** tempo médio gasto para localização de um novo mestre para um dado uma vez que foi descoberta a ausência do primeiro. Esse tempo será medido em segundos a partir do momento em que se descobre a ausência do nó mestre até a seleção do novo mestre.
- **Taxa de Sucesso**: Percentual de Novos Mestres Selecionados dentro de um limite de tempo estabelecido para simulação.

O primeiro experimento visa avaliar o tempo necessário para disseminação da informação entre os super-nós. A Figura 5.1 apresenta o tempo médio de disseminação de um dado entre dois super-nós selecionados aleatoriamente à medida que se aumenta a quantidade de nós da rede. Por múltiplos nós disseminarem os dados (através do *beacon*) e a configuração da rede ser dinâmica a cada simulação, o tempo necessário para disseminação tem alguns reflexos de oscilação, assim tempos menores são encontrados mesmo que se aumente a quantidade de nós.

Foram feitas 500 simulações de execução do algoritmo para cada quantidade de nós indicada no gráfico. Em cada simulação dois nós eram escolhidos aleatoriamente, onde um era responsável por iniciar o processo de envio de replicação enquanto o outro era responsável por indicar quando a informação chegou a ele. Ainda, a cada simulação uma nova estrutura de rede era gerada com o objetivo de testar o algoritmo em diversas topologias de rede, dando uma maior fidelidade ao funcionamento do mesmo.

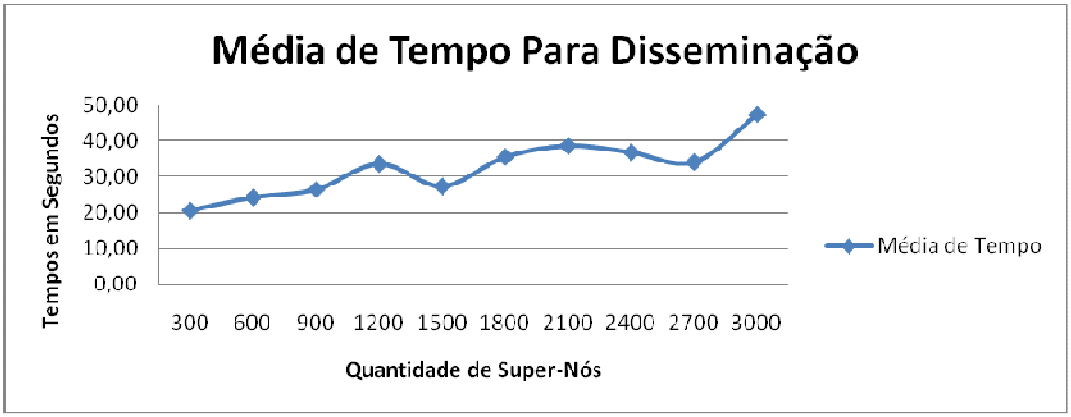

Figura 5.1 – Tempo médido para atualização de dados entre dois super-nós.

A segunda avaliação visa apontar a eficiência do algoritmo na seleção de um novo mestre, em caso de ausência do atual. Para tal, foram feitas 500 simulações com topologias de rede geradas aleatoriamente para cada simulação, bem como o número de nós que possuíam a cópia atual do objeto. Para o segundo parâmetro, o número mínimo de nós que possuíam a

cópia atual do objeto era 0 e o máximo era 10. Assim, a simulação parte do principio que o nó mestre de um dado está atualmente fora da rede e torna-se necessário procurar um novo mestre. É avaliado o tempo médio das simulações para um conjunto de nós. Abaixo apresentamos na Figura 5.2 com os resultados.

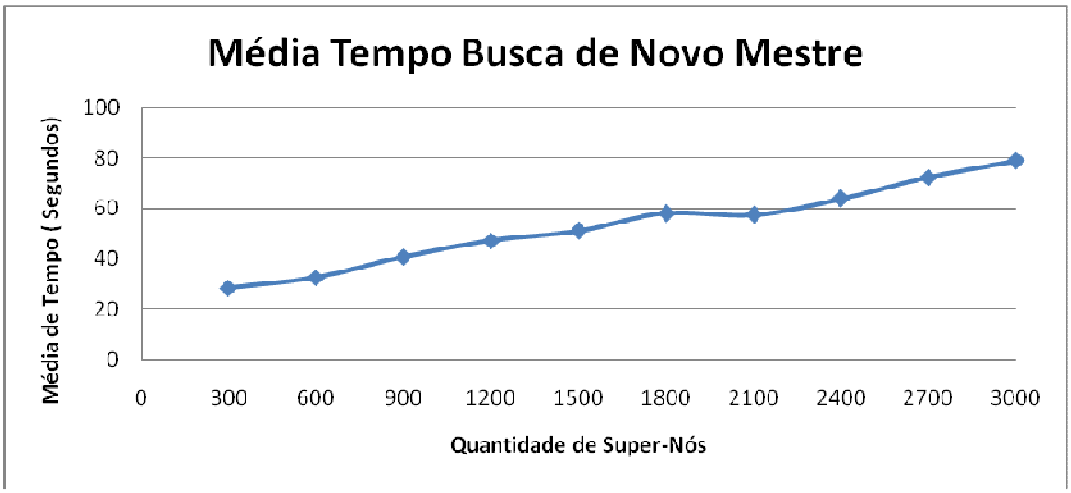

Figura 5.2 - Tempo médio para execução do algoritmo de localização de novo mestre.

A terceira avaliação é bem parecida com a segunda, mas o objetivo principal é verificar o percentual de taxa de sucesso na localização de novo mestre dentro de um tempo pré-definido de simulação. Para tal, estabelecemos tempos pré-definidos de simulação e aplicamos uma avaliação no intuito de verificarmos o percentual de novos mestres que são selecionados dentro desse limite de tempo. A intenção é mostrar que mesmo em tempos menores aos da média (indicados na Figura 5.2) é possível se ter um percentual bom de novos mestres encontrados.

Para essa simulação, foi escolhida uma rede com 300 super-nós e 500 simulações foram feitas para cada tempo de simulação apresentado na Figura 5.3.

Os tempos de simulação apresentados nos gráficos das figuras (Figura 5.1, Figura 5.2 e Figura 5.3) foram altos, mas ainda assim satisfatórios se levarmos em consideração o

dinamismo da rede, a grande quantidade de super-nós e ainda a necessidade de pesquisar todos os super-nós para o caso da busca de um novo mestre.

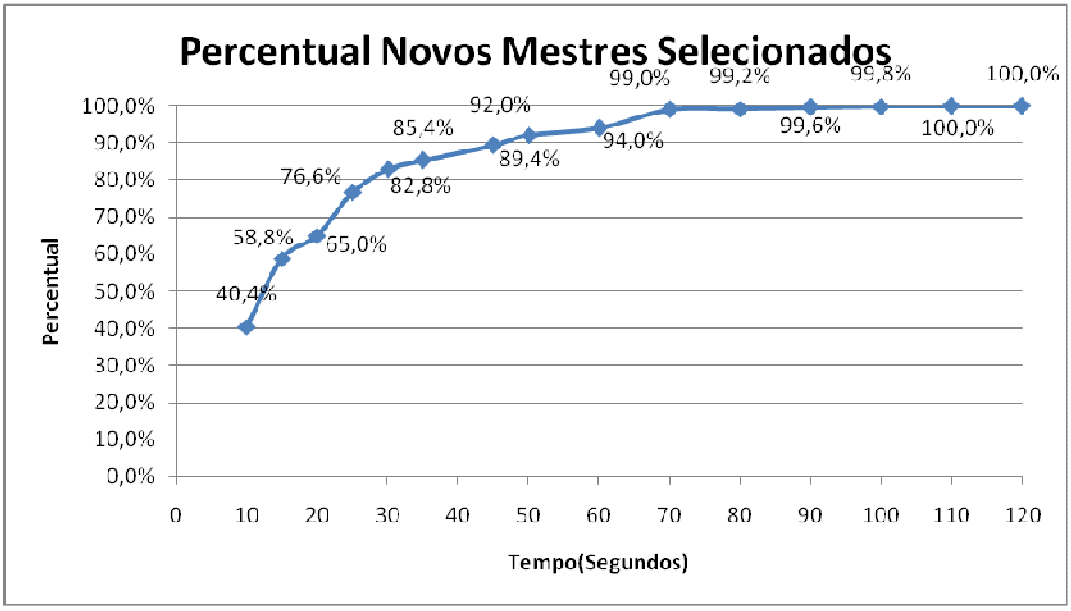

Figura 5.3 – Percentual de Novos Mestres selecionados X Tempo de Simulação

Após analisarmos os resultados, verificamos uma possibilidade de melhoria do algoritmo de busca de novo mestre com o intuito de diminuir o tempo médio de seleção de novo mestre bem como o percentual de seleção de novo mestre para tempos menores de simulação. Conforme apresentado na seção 4.4 , a Função de Propagação de Réplicas é utilizada para manter o nó mestre mais atual sempre existente na rede. Assim, a cada atualização do nó mestre atual de um determinado dado, ele seleciona alguns nós para os quais serão enviadas as cópias atuais do objeto principal. Na solução implementada, esse nós são selecionados de forma aleatória a cada atualização. A proposta de alteração estaria em não mais uma seleção aleatória dos nós, mas sim uma seleção fixa desses nós, desde que os mesmos se encontrem na rede. Dessa forma, sempre os mesmos nós seriam atualizados com o valor mais atual enquanto eles permanecerem ativos na rede. Quando algum desses se ausentar da rede, ele será substituído por um novo nó qualquer, que uma vez selecionado passará a ter sempre a cópia mais atual do objeto a cada atualização do mesmo.

Alguns problemas referentes à consistência podem surgir dessa alteração na proposta. Não iremos entrar em detalhes referentes a esses problemas, mas apenas queremos apresentar uma alternativa ao tempo gasto nessa seleção de novo mestre. Apresentamos abaixo os resultados referentes às métricas da simulação: **Latência de Consistência** e **Taxa de Sucesso**, seguindo os mesmo parâmetros apresentados na segunda e terceira avaliação.

Aplicados os mesmos parâmetros da segunda avaliação (Latência de Consistência) percebe-se pela Figura 5.4 que o algoritmo de busca de novo mestre baseado nos nós selecionados pela função de replicação (linha vermelha) são muito menores que os tempos da solução atualmente utilizada pelo algoritmo (baseado em inundação, representado pela linha azul). Isso se deve ao fato da busca ser feita em apenas alguns nós e não em toda rede, como é na solução original.

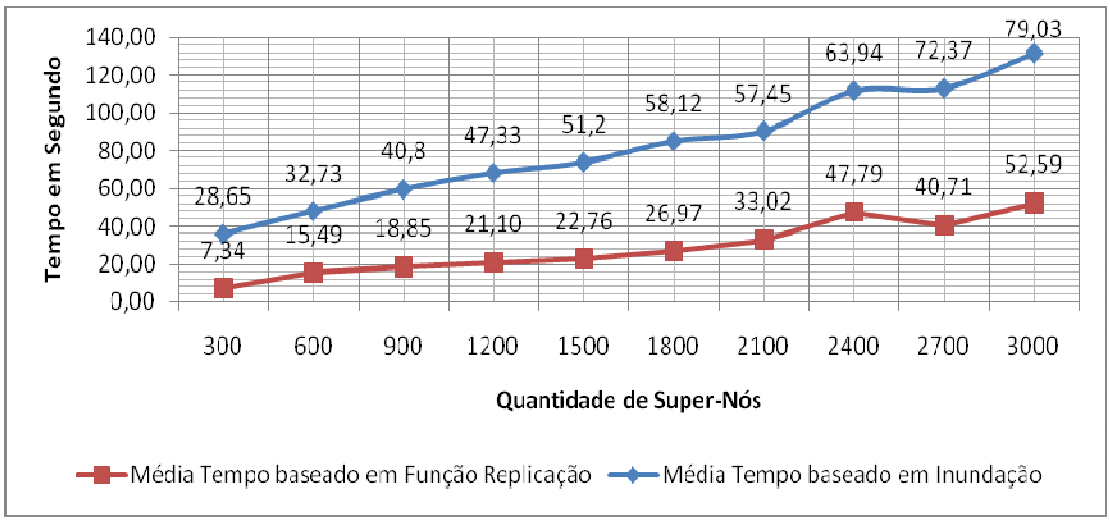

Figura 5.4 – Comparativo de Tempo entre os algoritmos de busca de novo mestre.

Vale ressaltar que, uma vez utilizando o modelo baseado na função de replicação, se nenhum dos nós atualizado pela função estiver na rede no momento da escolha de um novo mestre, não é possível se atingir esses resultados, sendo necessário recorrer ao algoritmo atual, ou seja, buscar através de inundação. Isso causaria um aumento no tempo de busca.

Assim, o maior tempo do algoritmo baseado na função de propagação de réplicas será igual ao do algoritmo baseado em inundação (que é o utilizando atualmente).

Outra verificação feita é referente à taxa de sucesso (terceira avaliação) na busca por um novo mestre. Na Figura 5.5 é feita a comparação entre o modelo de busca por novo mestre atualmente utilizado (através de inundação) e a melhoria vislumbrada baseada na função de replicação.

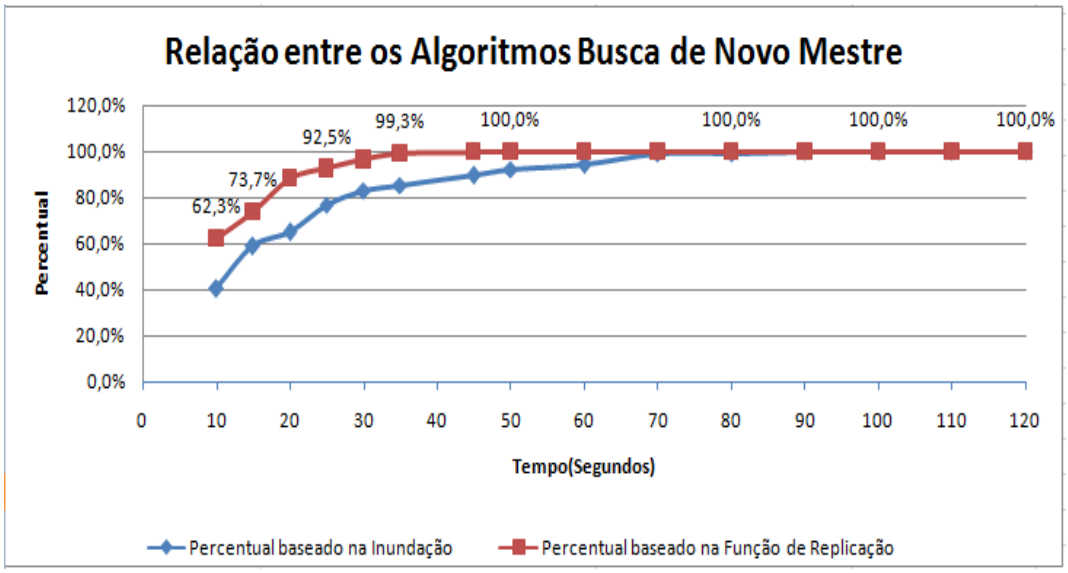

Figura 5.5 - Comparativo de Percentual de Novos Mestres Selecionados X Tempo de Simulação.

No comparativo da Figura 5.5 é visível notar que com apenas 10 segundos de tempo de simulação, mais de 50% das solicitações de busca de novo mestre são respondidas antes da finalização do tempo de simulação, utilizando o algoritmo baseado na Função de replicação. Já com o algoritmo baseado em inundação, apenas 40% das solicitações são respondidas.

Outra verificação do comparativo está no fato de que com 25 segundos já se tem mais de 90% de solicitações de novo mestre encontrados pelo algoritmo baseado na Função de Replicação. E finalmente, após 70 segundos ambos os algoritmos já estão em 100%, sendo que o baseado na Função de Replicação já se encontra nesse percentual desde os 45 segundos.

Isso mostra que essa estratégia é mais eficiente que a primeira apresentada no que se refere ao tempo para busca de novo mestre.

5.2 Estudo de Caso

O modelo proposto no capítulo 4 faz parte da arquitetura do Teseus (Silva, Braganholo & Tolla, 2008b). O propósito dessa arquitetura é criar um modelo que, através de replicação de dados, possibilite a cooperação entre sistemas distribuídos heterogêneos em um ambiente P2P com mobilidade, o qual compreende uma arquitetura distribuída com mobilidade de dados e de serviços. Para essa finalidade, serviços são disponibilizados através de um descritor (representado por um objeto) e mantidos em um ambiente totalmente distribuído e com garantia de consistência de dados.

A idéia é baseada fundamentalmente nos modelos para localização de serviços, tais como Jini (Edwards, 1999; Arnold *et al,* 1999) e WebService (Web Service, 2007), e faz parte do trabalho apresentado em (Gonçalves *et al,* 2007a; Gonçalves *et al,* 2007b). A intenção é criar uma arquitetura distribuída baseada em um conjunto de espaços de tuplas distribuídos com replicação e sincronização para localização de serviços (sendo esse conceito referente a dados, aplicações ou recursos) em ambientes P2P com mobilidade. Como metas, o projeto visa:

- Permitir a localização de serviços mesmo que estes estejam se movimentando na rede, e

- Criar um mecanismo onde serviços de mesma categoria possam ser escolhidos (baseados em algum critério) para permitir o balanceamento de carga na escolha dos nós que disponibilizam os serviços aumentando, assim, a robustez e impedindo que,

se, por exemplo, dois nós oferecem o mesmo serviço, um fique sobrecarregado enquanto outro fica ocioso.

De forma bem simples e macro, informações são disponibilizadas em espaços de tuplas e localizadas através do mesmo. Cada nó conhece um espaço de tuplas e o utiliza para compartilhar suas informações. Todos os espaços de tuplas estão conectados entre si e trocam informações constantemente através do modelo de replicação múltiplos únicos mestres. Essa característica permite a criação de um espaço de tuplas completamente distribuído. Para isso adaptações foram feitas na proposta inicial da especificação de espaço de tupla JavaSpace, das quais surge uma nova especificação de espaço de tuplas distribuído que faz parte da arquitetura do ***Teseus***.

Nesta seção apresentaremos o estudo de caso para validação da proposta do Modelo de Replicação através de uma necessidade real onde a arquitetura do Teseus foi utilizada como solução. Nas próximas seções apresentaremos: o estudo de caso, a utilização do Teseus no estudo de caso, a simulação do funcionamento do estudo de caso e por fim a conclusão deste capítulo.

5.2.1 Definindo o Estudo de Caso

Para este estudo de caso apresentaremos o Economic Cnossos (Moura, 2007). Este projeto faz parte da tese de Doutorado do aluno Luiz Gustavo Lourenço Moura e está em fase de desenvolvimento na instituição COPPE/UFRJ. O objetivo é utilizar o Teseus como um auxiliador do Economic Cnossos em sua proposta. Abaixo apresentaremos um resumo do Economic Cnossos para um maior entendimento e visualização de onde o Teseus se enquadra nessa proposta.

5.2.2 Aplicabilidade do estudo: Economic Cnossos

Observando-se o número de computadores interligados pela internet e analisando a porcentagem de quantos deles estão sempre trabalhando com grandes processos, descobre-se a ociosidade de processamento na maioria deles. Observa-se também, que não só existe ociosidade em processamento, mas há uma grande quantidade de espaço de memória e disco que poderiam ser utilizados de diferentes maneiras dentro de uma rede P2P.

Adar e Huberman (Agosto, 2000) descobriram que quase 70% dos usuários não compartilham recursos na rede P2P e aproximadamente 50% de todas as respostas são retornadas por 1% dos nós que possuem o recurso (nó hospedeiro).

Assim, o Economic Cnossos (Moura, 2007) visa definir medidas que incentivam os usuários a disponibilizar seus recursos, na forma de serviços, para compra e venda através de uma rede P2P não estruturada. O objetivo do projeto é trocar um recurso de que se necessita por um que se possa oferecer (através de compra e venda). Assim, todos os nós podem se beneficiar desse modelo. Para chegar nesse objetivo à seguinte premissa é utilizada: "quem deseja recursos deve necessariamente também dispor de recursos para efetuar as operações".

Assim, um modelo de compra/venda é utilizado para o gerenciamento de recursos e serviços dentro de uma rede P2P e deve se tornar um grande atrativo para que diversos usuários possam disponibilizar seus recursos nessa rede.

Cada nó possuirá um Economic Cnossos executando localmente. Assim, para ter direito a compra/venda, o nó deverá efetuar um cadastro onde irá definir quais os recursos e serviços que estarão disponíveis para venda. O nó deverá definir também a quantidade de recursos e os horários que os mesmos estarão disponíveis entre outras características que serão analisadas nas transações. Dessa forma, não será preciso o cadastro desse recurso a

cada vez que quiser oferecê-lo para compra/venda. Após esse cadastramento, os nós que utilizarem esse modelo possuirão uma conta onde, a qualquer momento, poderão contabilizar todas as suas transações. Nessa conta o usuário terá a visualização de todos os créditos obtidos através das vendas bem como todos os históricos das transações efetivadas. A moeda utilizada será uma única moeda denominada DEDALECAS.

Para exemplificar o funcionamento da compra/venda no Economic Cnossos, simularemos a procura por um recurso na rede P2P totalmente descentralizada. A transação inicia-se quando um determinado nó *X* deseja uma quantidade especifica de um recurso ou esteja necessitando de um determinado serviço. Digamos que esse nó *X* deseje armazenar uma quantidade expressiva de dados em memória, por exemplo, 120Gb, em uma determinada hora do dia, ou esteja necessitando desse recurso de imediato. Suponha também que o mesmo não tenha essa capacidade de armazenamento local. O nó *X* irá buscar na rede pelos recursos e serviços de que necessita.

Nesse momento, é estabelecida a relação entre a demanda de um recurso - isto é, a procura - e a quantidade que é oferecida, a oferta, conhecida na economia como Lei da Oferta e da Procura. Baseada nessa lei entra em ação um mecanismo de consulta na rede P2P que irá buscar todos os nós que possuem o recurso desejado e que esteja disponível de acordo com as especificações publicadas - oferta de serviços e recursos. O número de nós encontrados bem como a quantidade recursos disponíveis poderá ser o mais variado possível. O funcionamento do mecanismo ocorre da seguinte forma: o nó cliente faz uma requisição do recurso/serviço solicitado. Essa solicitação devolve as informações dos vizinhos que tenham publicado serviços/recursos que satisfaçam as condições pré-estabelecidas na pesquisa.

Encontrados os nós, será criado um ambiente no qual estarão presentes todos os nós que estiverem dentro das especificações publicadas juntamente com o nó *X* onde se iniciará a

negociação entre os mesmos. Esta negociação dar-se-á através de mecanismo de leilão reverso chamado de pregão. No pregão, embora a disputa comece com propostas secretas, apenas após rodadas sucessivas de lances em que o valor pode ser reduzido, determina-se o vencedor ou vencedores.

Cada serviço/recurso possui um valor mínimo e máximo de venda, que será levado em consideração para resolução da compra. Outras características também podem ser levadas em consideração para tal.

Chegado a um acordo, os nós efetuam a compra/venda de acordo com as condições estabelecidas na publicação do pedido do recurso/serviço.

5.2.3 Aplicação do Teseus ao Estudo de Caso

Para auxiliar o Economic Cnossos em seu propósito, o Teseus será utilizado. Sua principal função será manter disponíveis os descritores referentes aos recursos/serviços de forma atualizada e com garantia de consistência.

Esses descritores conterão as informações sobre os recursos/serviços, bem como seus valores mínimos e máximos de venda e uma conta onde serão armazenados os valores a cada vez que um nó comprar um serviço.

Assim, quando um nó se cadastra no Economic Cnossos, ele deve informar os serviços/recursos a serem disponibilizados. Esses serviços/recursos serão encapsulados na forma de serviços P2P e seus descritores serão disponibilizados no Teseus. Tanto o Teseus quanto o Economic Cnossos estarão locais na máquina.

Quando um nó desejar comprar um serviço, ele se conecta ao Economic Cnossos que por sua vez fará a busca pelo descritor do serviço no Teseus. Uma vez encontrado, o Economic fará o processo de compra/venda e finalizada a tarefa transferirá as DEDALECAS

referentes à compra para o descritor do serviço selecionado. Nesse momento será exigido do modelo de garantia de consistência do Teseus que as informações referentes à compra não fiquem desatualizadas.

Cada serviço possuirá uma informação referente ao proprietário do serviço. Isso permitirá que o Economic Cnossos possa a qualquer tempo buscar no Teseus os serviços aos quais ele é proprietário e reunir suas DEDALECAS em sua conta no Economic Cnossos. Uma vantagem de se utilizar o Teseus é permitir que o serviço possa ser vendido mesmo que o nó esteja fora do ar. Isso é importante quando possuímos uma rede muito heterogênea, onde a quantidade de dispositivos móveis é grande e a possibilidade de desconexão é inerente.

5.2.3.1 Definindo a estrutura

Para facilitar a implementação do modelo é necessário que todos os nós que desejem compartilhar (vender) seus serviços/recursos assumam um modelo único para publicação dos seus serviços. A Tabela 3 apresenta a lista de parâmetros necessários para à implementação do modelo proposto.

Tabela 3 – Informações presentes no descritor do serviço publicado no Teseus.

ATRIBUTO	DESCRIÇÃO
UID	Identificador Único do Serviço
Tags	Identificadores para consulta do Serviço
Descrição	Descreve o propósito do Serviço
Preço Mínio	Indica um valor mínimo de negociação do Serviço
Preço Máximo	Indica um valor máximo de negociação do Serviço
Status	Indica se o serviço esta On ou Off . Ele estará Off quando o nó cliente sofre uma falha e desconecta-se da rede.
Proprietário	Indica o dono do serviço.
Conta	Acumula as compras do serviço
Disponibilidade	Indica que o serviço está disponível ou não (mesmo que o serviço esteja on-line ele pode ser indisponibilizado pelo seu proprietário)

Quando um usuário deseja vender um serviço, o Economic publica este serviço no Teseus através do descritor (apresentado na Tabela 3). Uma vez inserido no Teseus, o modelo

de replicação múltiplos únicos mestres se incumbe de replicar os descritores e resolver a consistência durante a presença do serviço e após sua saída (quer seja por falha ou por livre escolha), a informação de venda do serviço continuará disponível. O descritor poderá ser removido pelo proprietário do serviço em algum momento, o que ainda está em fase de definição pelo Economic Cnossos.

Para exemplificar o funcionamento, suponha uma rede com três nós onde dois deles compartilham serviços e recursos para compra/venda e o terceiro apenas serviços. Um dos nós é um supercomputador com grande poder de processamento de 4GHz Dual Core (identificado como *p*) enquanto o segundo possui uma capacidade de processamento de 1.6 GHz Dual Core (identificado como *q*). O terceiro nó é um celular com recursos limitados (identificado por *r*). A Figura 5.6 apresenta a rede.

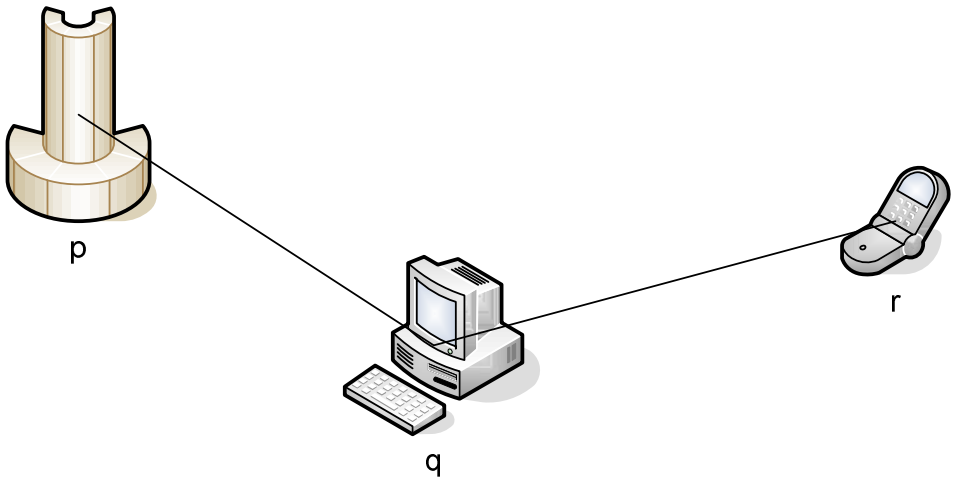

Figura 5.6 – Arquitetura da rede.

Cada um dos nós possui o Teseus e o Economic Cnossos instalados localmente. Para disponibilizar os serviços, eles se registram no Economic que insere no Teseus os descritores dos serviços. Para exemplificar usaremos apenas os descritores dos serviços de *p* e *q*, os quais são apresentados conforme a Tabela 4 abaixo.

Quando *r* deseja comprar um serviço de processamento acima de 1 GHz e que seja dual core, ele abre um processo de compra no Economic procurando por: tag=<processamento >= 1GHz dual>. O Economic repassa a consulta para o Teseus que retornará ambos os descritores dos nós *p* e *q* como resposta.

Nesse momento o Economic Cnossos executa um processo para definir quem será o "vencedor" na venda. Digamos que seja o nó *q* e que a venda tenha acontecido a um valor de 300 DEDALECAS. Assim, o valor será removido da conta do nó *r* no Economic Cnossos e publicado no descritor do serviço *q* no Teseus. É importante notar que o Teseus garantirá, através do seu modelo de replicação, a consistência dessas informações. Assim, o descritor do nó *q* referente ao serviço de processamento passará a ter as informações conforme a Tabela 5 abaixo.

Tabela 4 – Descritores de *p* e *q* publicados no Teseus.

ATRIBUTO	DESCRIÇÃO
UID	P1
Tags	<processamento= 4GHz dual>
Descrição	Processamento a 4 GHZ Dual Core
Preço Mínio	100
Preço Máximo	500
Status	On
Proprietário	P
Conta	0
Disponibilidade	On

ATRIBUTO	DESCRIÇÃO
UID	Q1
Tags	<processamento= 1.6GHz dual>
Descrição	Processamento a 1.6 GHZ Dual Core
Preço Mínio	50
Preço Máximo	400
Status	On
Proprietário	Q
Conta	0
Disponibilidade	On

A cada nó que comprar este serviço, o valor do atributo *conta* deverá ser acumulado. Num determinado instante o nó *q*, através do Economic Cnossos irá retirar os valores do serviço Q1, existente no atributo *conta* e acumulará em sua conta, para permitir que o mesmo possa agora, então, comprar novos serviços.

Tabela 5 - Descritor do nó *q* presente no Teseus. Este sofreu uma atualização do atributo conta, gerenciado pelo modelo de Replicação do Teseus.

ATRIBUTO	DESCRIÇÃO
UID	Q1
Tags	<processamento=1.6GHz dual>
Descrição	Processamento a 1.6 GHZ Dual Core
Preço Mínio	50
Preço Máximo	400
Status	On
Proprietário	Q
Conta	**300**
Disponibilidade	On

O fato do acúmulo da compra do serviço no descritor do serviço traz algumas vantagens, entre elas: a possibilidade da compra, mesmo que o nó esteja fora do ar; a possibilidade de se vender um serviço mesmo que seja para utilização posterior e ainda, a possibilidade do agendamento de um serviço para um determinado horário entre outros. Assim, o Teseus auxilia o Economic Cnossos em suas tarefas, sem se atentar aos detalhes da compra/venda, mas sendo um suporte fundamental para a efetuação do mesmo.

5.3 Simulação do Funcionamento

Nessa seção apresentaremos a simulação de uma rede utilizando o Economic Cnossos e o Teseus com o objetivo de implementar a compra/venda de serviços/recursos. Nas próximas seções apresentaremos os seguintes passos: a rede utilizada para simulação; os descritores dos serviços disponibilizados pelos nós e suas contas; a operação de compra de um Serviço; a aplicação de Garantia de Consistência na ausência de um nó e a necessidade de compra de um serviço; e, por fim, a retirada das DEDALECAS referentes aos serviços comprados durante a ausência do nó provedor do mesmo.

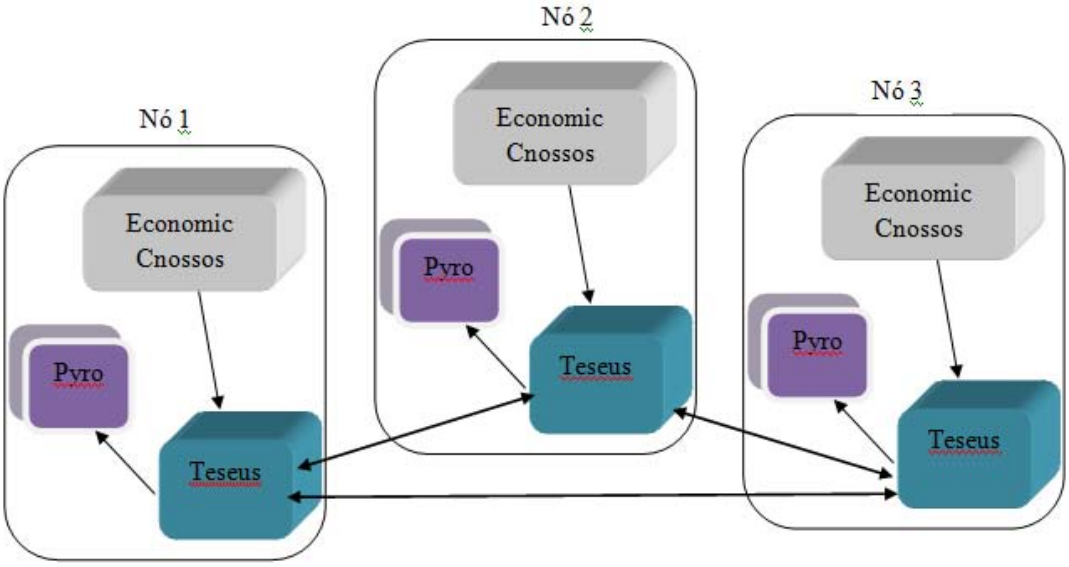

Figura 5.7 – Integração entre as aplicações dentro de um mesmo nó para publicação de Serviços no Teseus. E integração entre os nós com Teseus para Replicação da Informação.

A aplicação foi desenvolvida utilizando a linguagem Python (Python, 2008). Os serviços são representados através de objetos na linguagem python que são publicados no *Python Remote Objects* (Pyro) (Pyro, 2008), uma aplicação para publicação de serviços que podem ser acessados via *Remote Method Invocation* (RMI). Ao publicar um serviço no pyro, este gera um *Uniform Resource Identifier* (URI), que permite ao solicitante do serviço invocá-lo remotamente.

Para poder utilizar o serviço, o URI é adicionada ao descritor do serviço, o qual é publicado no Teseus. Dessa forma, ao encontrar um serviço no Teseus é possível executá-lo utilizando o URI através do Pyro.

A Figura 5.7 apresenta a integração entre as aplicações para publicação de um serviço. O Economic pede ao Teseus que publique o serviço, o Teseus gera o URI no pyro e publica o serviço, repassando o mesmo para os demais Teseus.

5.3.1.1 Definição do Ambiente

A rede para nosso teste será formada por quatro alunos compartilhando seus serviços enquanto assistem aula numa sala. Cada um deles possui um dispositivo móvel, sendo os quatro notebooks.

Cada dispositivo possui o Teseus, o Economic Cnossos e o Pyro instalados. Os notebooks se comunicam entre si através do Teseus. Os notebooks serão chamados de N1, N2, N3 e N4. O Teseus está configurado com uma função de propagação de réplicas (FPR), onde FPR= Log_2 (1+V), assim a cada número impar de atualizações uma cópia atualizada será encaminhada a algum outro nó.

Para darmos início à simulação, suponha que a professora de Matemática Financeira passou um exercício para os quatro alunos os quais deveriam utilizar uma calculadora HP 12C. Para o que aluno com o notebook N1 resolvesse o exercício, ele comprou um software que é uma calculadora HP 12C e resolveu fornecê-lo como um serviço para os demais colegas da escola cobrando um valor por isso. No momento o aluno do note N1 se encontra na biblioteca terminando o exercício de matemática financeira. Enquanto isso o aluno do note N2 criou uma aplicação que executa as mesmas funções da calculadora HP 12C para resolver os exercícios da aula passada e decidiu que também forneceria como serviço para seus colegas de escola. Nesse momento ele está assistindo a aula na sala. N3 e N4 por sua vez, não possuem essa calculadora e estão na sala de aula tentando resolver o exercício. Ambos os alunos fazem parte de uma mesma turma e assistem à aula de matemática financeira juntos. Abaixo a Figura 5.8 representa a sala e a biblioteca da escola.

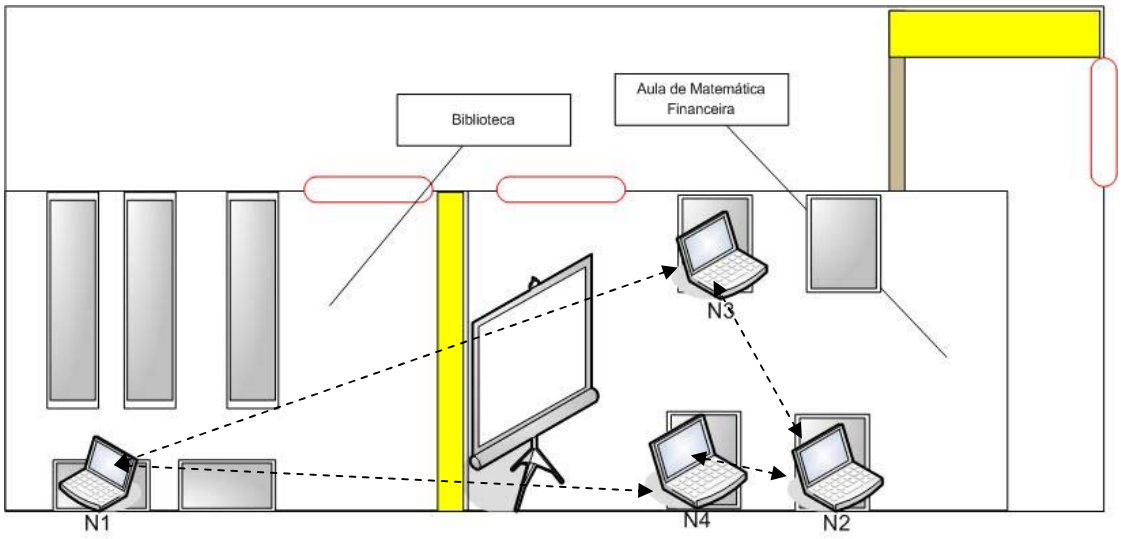

Figura 5.8 - Sala e Biblioteca da Escola.

Cada nó iniciará a simulação com uma conta de 100 DEDALECAS, que podem ser entendidas como advindas de outras transações de compra/venda de outros serviços.

5.3.1.2 Descritores dos Serviços nos nós

Os nós N1 e N2 possuem os serviços necessitados por N3 e N4. Todos eles estão cadastrados no Economic Cnossos e N3 e N4 querem comprar os serviços. Para que N3 e N4 possam adquirir os serviços, os descritores dos mesmos devem ser publicados. A Tabela 6 mostra os descritores dos serviços de N1, com o UID N1calc e de N2 com o UID N2calc.

O modelo de replicação múltiplos únicos mestres, utilizado pelo Teseus, permite que os descritores dos serviços de N1 e N2 sejam replicados para todos os outros nós e que eles possam assim conhecer, consultar e comprar o serviço mesmo que o nó provedor deste esteja desconectado da rede no momento.

Tabela 6 - Descritores do serviço de calculadora HP 12C publicado no Teseus pelos Notebooks N1 e N2.

ATRIBUTO	DESCRIÇÃO
UID	N1calc
Tags	<calculadora> <HP12C>
Descrição	Software comprado da empresa XYZ
Preço Mínimo	10
Preço Máximo	25
Status	On
Proprietário	N1
Conta	0
Disponibilidade	On

ATRIBUTO	DESCRIÇÃO
UID	N2calc
Tags	<calculadora> <HP12C>
Descrição	Software feito por mim
Preço Mínimo	5
Preço Máximo	15
Status	On
Proprietário	N2
Conta	0
Disponibilidade	On

A Figura 5.8 apresenta os nós compartilhando os dados referentes a seus descritores com os seus vizinhos. Esses dados são enviados para os demais nós vizinhos. Todos agora têm a informação dos demais nós.

Ao iniciar, o nó N3 se conecta ao nó N2 para comprar o serviço da calculadora HP. A Figura 5.9 apresenta a inicialização do nó N3, sua busca pelo serviço, a compra feita pelo Economic Cnossos e o valor da compra.

5.3.1.3 Compra de um serviço

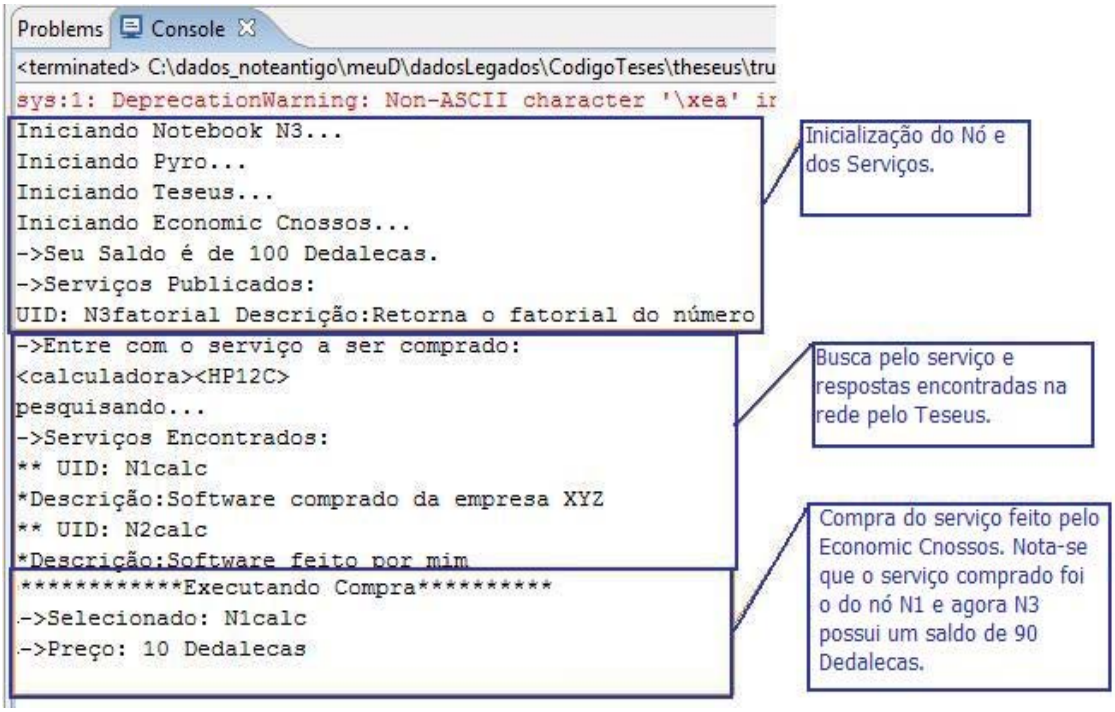

Figura 5.9 - Execução completa do nó N3 para compra do serviço da calculadora.

5.3.1.4 Garantia de Consistência na Ausência do Nó durante a compra de um Serviço

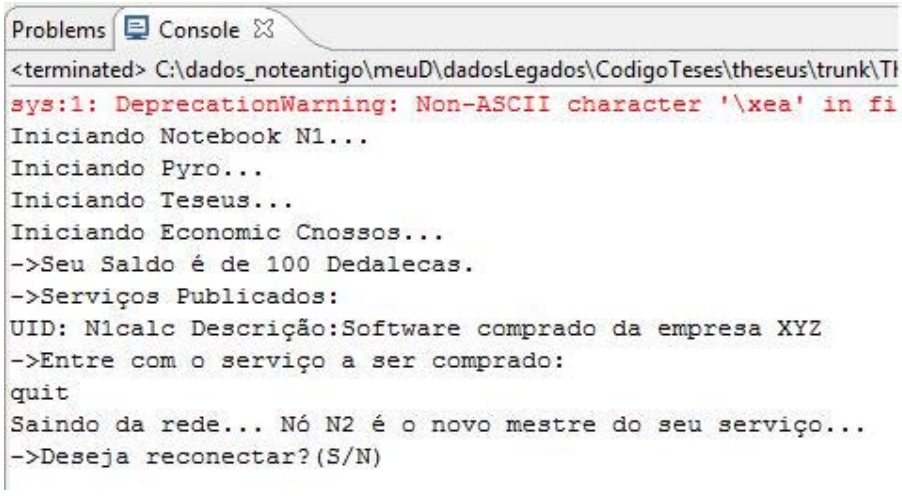

Figura 5.10 Nó N1 desconecta-se da rede para mudar de sala

Após alguns minutos, o aluno do Notebook N1 resolve ir para sala de aula trocar informações com N2 sobre o exercício, nesse momento ele sai da rede e como foi uma saída e não uma falha, o Teseus se incumbe de escolher um novo mestre para o serviço

disponibilizado pelo nó N1. A Figura 5.10 apresenta a saída dos nós. O comando "*quit*" faz com que o nó se desconecte da rede, mas os serviços continuem funcionando.

Nesse momento, enquanto N1 está mudando de uma sala para outra o aluno com o notebook N4 deseja comprar também o serviço de calculadora. Podemos verificar que nos serviços selecionados para compra, aparece também o serviço de N1 mesmo que o nó esteja fora do ar. A Figura 5.11 apresenta N4 executando sua compra. Nela, N4 compra o serviço de N1 por 12 Dedalecas, mas é apresentado que o serviço está indisponível.

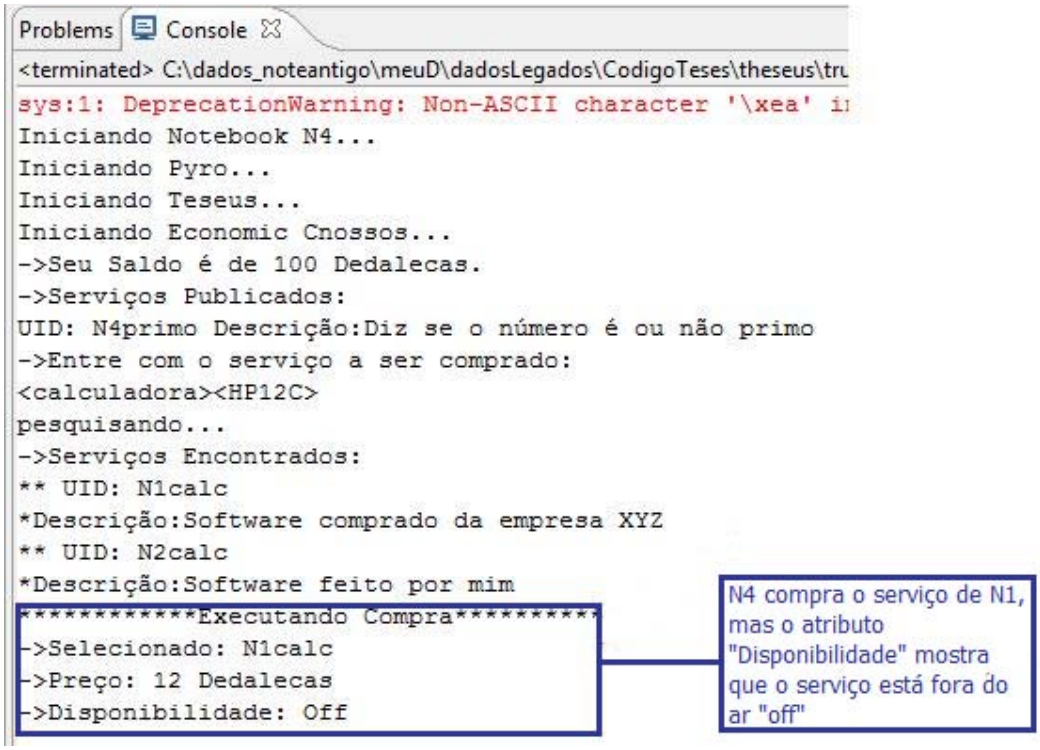

Figura 5.11 - N4 compra o serviço de N1, mesmo o nó estando fora do ar

5.3.1.5 Retirada das informações dos serviços para acúmulo em conta

Após algum tempo, N1 está na sala e resolve disponibilizar novamente o serviço. Nesse momento, N1 pede para atualizar sua conta através do comando "*update*". Assim, uma

mensagem é enviada para o mestre do serviço de N1 (no caso, N2) e pede a ele que envie o valor contido na conta. A Figura 5.12 apresenta o resultado do comando.

```
->Deseja reconectar?(S/N)
S
->Descobrindo o mestre do serviço N1calc...
Novo mestre: N2
->Entre com o serviço a ser comprado:
update
Atualizando o valor da conta...
->Seu Saldo é de 122 Dedalecas.
```

Figura 5.12 Resultado da atualização da conta em N1

Podemos notar que o saldo de N1 é de 122 Dedalecas. Isso se deve ao fato de N1 iniciar com 100 Dedalecas e também por ter vendido seu serviço para os nós N3 e N4 por 10 e 12 Dedalecas, respectivamente.

5.4 Conclusão

Neste capitulo foram apresentados a avaliação do algoritmo de replicação e garantia de consistência, bem como o estudo de caso, que objetivou apresentar a utilização da proposta em uma possível aplicação para os dias atuais.

A avaliação teve como foco principal o tempo de execução para disponibilização da informação entre os nós e o tempo para localização de um novo mestre. Baseados nesses parâmetros podemos constatar um tempo alto, principalmente na localização de um novo mestre. Assim, no intuito de diminuir o tempo de localização, foi apresentado um algoritmo alternativo, onde se manteriam fixos os nós atualizados pela função de propagação de réplicas e os resultados apresentados foram bem melhores do que os do algoritmo atualmente utilizado. Não foram apresentados com profundidade os impactos referentes à substituição do algoritmo atual (o qual se baseia em inundação) pelo baseado na função de propagação de réplicas, sendo indicado para análise e como um trabalho de melhoria futura do modelo atual.

No estudo de caso apresentado, o modelo proposto por este trabalho suplantou as necessidades de resolução dos problemas intrínsecos ao projeto Economic Cnossos (Moura, 2007) e ao tipo de ambiente no qual ele se propõe à atuar (que necessitam de mobilidade e a busca de informação a qualquer momento e lugar). O ambiente escolhido foi uma escola onde o objetivo era apresentar uma aplicação prática deste trabalho num cenário para compra e venda de serviços. Os serviços eram publicados e replicados. À medida que os clientes necessitavam de serviços, estes eram pesquisados e uma vez escolhidos dava-se inicio a uma tarefa de compra.

O Teseus foi utilizado no auxilio para as fases de publicação, replicação e escolha dos serviços, passando informações que servem de insumo para o Economic Cnossos e para a execução da compra/venda dos serviços. Outra função do Teseus era armazenar e garantir a consistência dos valores de compra feitos pelos clientes aos serviços, mesmo que os nós provedores do serviço estivessem fora do ar. Esse estudo apresentou o projeto Teseus e garantiu a aplicabilidade do modelo proposto em uma situação possível no mundo real. Vale ressaltar ainda que, pelo fato de ser usado em uma aplicação voltada a serviços, o modelo de replicação possui um diferencial de não ser apenas aplicável a dados.

No próximo capítulo serão apresentados os trabalhos futuros e as considerações finais.

6 Conclusão e Trabalhos Futuros

Esse trabalho apresentou um modelo de replicação em ambientes P2P com mobilidade. Este modelo resolve a consistência dos dados de forma seqüencial permitindo várias solicitações de atualização simultâneas e evita trocas excessivas de mensagens após uma replicação. O modelo prevê que cada informação tenha um nó mestre responsável por ela. Na ausência ou falha deste mestre, um novo mestre deve ser escolhido. Neste caso, para garantir consistência, basta que apenas um nó na rede tenha o dado mais atual (e que este não falhe).

A função de propagação de réplica trabalha para auxiliar a manutenção do estado atual de um objeto *o* de forma a garantir que, a cada atualização, um conjunto de vizinhos receba a cópia mais atual de *o*. A função proposta garante que no pior caso, 30% dos vizinhos terão a informação mais atual (considerando uma rede onde cada nó tenha 10 vizinhos). Além disso, à medida que mais atualizações ocorrem, mais importante aquele objeto se torna para o sistema e maior quantidade de vizinhos são atualizados. Essa estratégia aumenta a probabilidade de garantia de consistência e evita o tráfego excessivo de dados replicados a cada atualização.

Alguns modelos foram apresentados e serviram como base para a prospecção do modelo apresentado., o qual não tem como objetivo resolver os problemas de todos os modelos estudados, mas ser uma alternativa em ambientes P2P, onde não existe a presença de um servidor e os nós trabalham excessivamente com troca de informações. Permitir a troca de dados nesse ambiente heterogêneo resolvendo a consistência é um grande aspecto de estudo para apresentação desse trabalho.

Como contribuição, apresentamos uma proposta de alteração do algoritmo de único mestre, a qual foi denominada de modelo de múltiplos únicos mestres. Nele, o nó que disponibiliza a informação para replicação torna-se mestre dela. Dessa forma, vários mestres estarão presentes para diferentes objetos diminuindo os grandes problemas do modelo único mestre: o gargalo no acesso e o problema da impossibilidade de atualização no caso da falha do nó mestre. Essa contribuição gerou um relatório técnico (Silva, Braganholo & Tolla, 2008) e um artigo submetido ao SBBD 2008.

O primeiro é resolvido, pois a quantidade de mestres presentes na rede é proporcional ao número de nós provedores de dados. O segundo é resolvido pela característica do modelo no qual a ausência de um mestre em um determinado instante não impossibilita o acesso aos dados cujo mestre seja outro nó. Outra característica ainda é o fato da seleção de um novo mestre durante a ausência do atual e por fim a execução da função de propagação de réplicas aplicada sobre as atualizações, o que permite que a informação mais atual e dos dados que sofrem o maior número de atualizações estejam sempre vivas na rede.

Para permitir que nós com pouca capacidade de armazenamento fossem beneficiados pelo modelo de replicação, um modelo de espaço de tuplas foi criado implementando o modelo de replicação. Esse modelo de espaço de tuplas foi denominado Teseus. Ele está presente em todos os nós da rede, mas somente os nós com capacidade de armazenamento recebiam informações replicadas, enquanto os nós sem capacidade de armazenamento (como celulares, por exemplo) se conectavam a esses nós para poder compartilhar seus dados e localizar dados compartilhados por outros nós da rede.

Por fim, foi feita uma avaliação do modelo de replicação e um estudo de caso utilizando o modelo de replicação aplicado ao Teseus. A avaliação teve como ponto focal o tempo necessário para replicar os dados entre os nós e principalmente a avaliação do

algoritmo para escolha de novo mestre. Analisado o tempo, foi proposta uma alteração no algoritmo com o objetivo de diminuí-lo. A proposta de alteração no algoritmo foi bem sucedida, mas detalhes relacionados à mudança não foram estudados a fundo. O objetivo principal dessa alteração era mostrar que o algoritmo pode ser melhorado, ficando para trabalhos futuros a melhoria da solução proposta.

O estudo de caso teve como objetivo apresentar a utilização da solução como um todo e verificar o comportamento do modelo de replicação em uma solução voltada para o mundo real. Um fato importante da aplicação apresentado através do estudo foi verificado pela utilização da solução em um modelo voltado para serviços. Soluções para replicação de dados são normalmente utilizados apenas com dados. A aplicação da solução em um problema relacionado a serviços mostrou a pluralidade de problemas nos quais a solução pode ser utilizada.

A próxima seção visa apresentar novos passos que precisam ser dados em relação a melhorias na proposta atual.

6.1 Trabalhos Futuros

O modelo proposto possui algumas possibilidades de melhoria, desde o modelo de replicação à implementação da aplicação.

Dentre as melhorias no algoritmo, propomos o tratamento de problemas referentes à escolha de um novo mestre em caso de particionamento da rede. Propomos também a alteração do algoritmo de busca de novo mestre, que atualmente utiliza inundação como forma de comunicação. A alteração seria baseada no uso de um conjunto fixo de nós, que sempre receberiam a versão mais atual de um objeto. Como demonstrado via simulação, essa mudança causa uma diminuição drástica no tempo necessário para localização de um novo

mestre. No entanto, cabe ressaltar que é necessário um estudo aprofundado do impacto dessa alteração dentro do funcionamento do algoritmo como um todo, pois algumas questões precisam estar esclarecidas antes de sua aplicação à solução atual. Dentre elas destacamos duas: Como fazer para saber se o nó que recebeu o dado atualizado, após uma falha, possui o dado ainda atualizado? E como tratar esse problema? Pela limitação do tempo, perguntas como essas e outras não puderam ser respondidas.

A avaliação do modelo de replicação pode ser melhorada para indicar o quanto a solução é eficiente, em relação ao tempo de resposta, na busca de novo mestre e na disseminação dos dados. Para tal torna-se necessário compará-la com outras abordagens e utilizar um simulador em comum para tal aferição. Tal avaliação não pôde ser feita devido a restrições de tempo.

Outra melhoria proposta está na criação de uma interface gráfica para interação com o usuário e uma instalação e atualização facilitadas para o usuário final.

Referências

Adar, E., Huberman, B. A. (2000) *Free Riding on Gnutella*. First Monday,5(10), Agostos, 2000.

Akbarinia, R., Pacitti, E., & Valduriez, P. (2007). *Data Currency in Replicated DHTs*. In: International Conference on Management of Data, SIGMOD, pp. 211-222. New York: ACM.

Barbará, D. (1999). Mobile Computing and Databases- A Survey. *IEEE Transactions on Knowledge and Data Engineering , 11* (1), pp. 108-117.

Benevenuto, F., Júnior, J. I., & Almeida, J. (2005). *Avaliação de mecanismos avançados de recuperação de conteúdo em sistemas p2p.* In: Simpósio Brasileiro de Redes de Computadores, Fortaleza, Brasil.

Bjornson, R. D. (1992). *Linda on Distributed Memory Multiprocessors.* Universidade de Yale: Tese de doutorado,1992.

Budhiraja, N., Marzullo, K., Schneider, F., & Toueg, S. (1993). The Primary-Backup Approach. In: Mullender, S. (Org). *Distributed Systems*, 2a. edição. New York: ACM Press/Addison-Wesley.

Casanova, M. A. (1981). *The Concurrency Problem of Database Systems*. In:Lectures Notes in Computer Science , p. 116, 1981.

Coulouris, G., Dollimore, J., & Kindberg, T. (2001). *Distributed Systems: Concepts and Design.* 3ª edição, Addison-Wesley, 2001.

Daswani, N., Molina, H. G., & Yang, B. (2003). *Open problems in data-sharing peer-to-peer systems.* In: International Conference on Database Theory.

Datta, A., & Aberer, K. (2004). *Autonomous Gossiping: Self-Organizing Epidemic Algorithm for Selective Information Dissemination in Wireless Mobile Ad-Hoc Network.* In: Semantics of a Networked World, pp. 126-132. Paris, França: Springer.

Genç, Z. (2005). *SFG: Flooding Smart by Gossiping.* In: International Conference On Emerging Networking Experiments And Technologies, Toulouse, França. New York: ACM.

Gnutella. (2006). *Genutelliums*.Disponível em <http://www.gnutelliums.com/>. Acesso em 26 de Outubro de 2007.

Gnutellahost (2008).Disponível em < http://www.gnutellahosts.com/>. Acesso em 27 de Maio de 2008.

Gonçalves , F. B., Oliveira, C. E. T., Silva, I. P. , Moura, L. G. L. (2007) *An Architectural Model for Applications Based on Mobile Services*. In: International Multi-Conference on Computing in the Global Information Technology, 2007, Guadaloupe. nternational Multi-Conference on Computing in the Global Information Technology, 2007.

Gonçalves, F. B. , Oliveira, C. E. T.; Silva, I. P. , Moura, L. G. L , França, F. M. G. (2007). *A Software Architecture for Provisioning of Mobile Services in Peer-to-Peer Environments*. In: The Second International Conference on Internet and Web Applications and Services, 2007, Mauritius. The Second International Conference on Internet and Web Applications and Services, 2007.

Gray, J., Helland, P., O'Neil, P. E., & Shasha, D. (1996). *The dangers of replication and a solution*. In: International Conference on Management of Data, SIGMOD, pp. 173-182. New York: ACM.

Icq. (2007). Disponível em<http://www.icq.com>. Acesso em 26 de Outubro de 2007.

Jajodia, S. & Mutchler, D. (1990). Dynamic voting algorithms for maintaining the consistency of a replicated database. *ACM Transactions on. Database Systems,* 15(2), (Jun. 1990), pp. 230-280.

Jovanovic, M. (2000). *Modelling large-scale peer-to-peer networks and a case study os Gnutella.* Department of Electrical and Computer engineering and Computer Science, University of Cincinnati, Cincinnati, Ohio, 2000.

Jovanovic, M., Annexstein, F., & Berman, K. (2001). *Scalability issues in large peer-to-peer networks:a case study of Gnutella.* University of Cincinnati, Cincinnati, Ohio, 2001.

Kazaa. (2007). Disponível em <http://www.kazaa.com>. Acesso em 26 de Outubro de 2007.

Lv, Q., Cao, P., Cohen, E., Li, K., Shenker, S. (2002). *Search and replication in unstrctured peer-to-peer networks.* In: ACM Int. Conf. on Supercomputing(ICS) pp. 84-85, New York, 2002.

Lynch, N. (1996). *Distributed Algorithms*, Morgan Kaufmann Publishers, 1996.

Mondiano, E., Shah, D., & Zussman, G. (2006). Maximizing Throughput in Wireless Network Via Gossiping. *ACM SIGMETRICS Performance Evaluation Review, 34(1),* pp. 27-38. New York: ACM.

Monteiro, J. M., Brayner, A., & Lifschitz, S. (2007). *A mechanism for replicated data consistency in mobile computing environments*. In: ACM Symposium on Applied Computing, pp. 914-919.

Moura, L. G. L. (2007). *Economic Cnossos.* Universidade Federal do Rio de Janeiro - COPPE: Tese de doutorado,2007 – Em Andamento.

Nandy, S., Carter, L., & Ferrante, J. (2005). *GUARD: Gossip Used for Autonomous Resource Detection*. In: Parallel and Distributed Processing Symposium, p. 58-58. [S.l.]:IEEE.

Napster. (2007). Disponível em < http://www.napster.com> . Acesso em 26 Outubro de 2007.

Paris J. F. & Long D. D. E (1988). *Efficient Dynamic Voting Algorithms*. In: International Conference on Data Engineering, pp. 268-275, Los Angeles, California.

Pyro. (2008). Disponível em < http://pyro.sourceforge.net/ > . Acesso em 27 Maio de 2008.

Python. (2008). Disponível em < http://www.python.org> . Acesso em 27 Maio de 2008.

Ratnasamy, S., Francis, P., Handley, M., Karp, R., & Shenker, S. (2001). *A Scalable Content-Addressable Network*. In: Conference on Applications, Technologies, Architectures and Protocols for Computer Communications, pp. 161-172. San Diego, California. New York:ACM.

Satyanarayanan, M. (2001). Pervasive Computing: vision and challenges. *IEEE Wireless Communications , 8* (4), pp. 10-17.

Seti. (2007). SETI@Home. Disponível em<http://www.setiathome.ssl.berkeley.edu/>. Acesso em 26 de Outubro de 2007.

Silva, I.P., Braganholo, V., Tolla, C. E. (2008). *Um Modelo de Replicação com Garantia de Consistência em Ambiente P2P com Mobilidade.* Relatório de Pesquisa, NCE-04/2008.

Silva, I.P., Braganholo, V., Tolla, C. E. (2008b). *Teseus : Uma arquitetura para manutenção de informações em ambientes P2P.* Relatório de Pesquisa, disponível em: www.dcc.ufrj.br/~braganholo/artigos/RT-Teseus.pdf.

Skouteli, S., Samaras, G., Pitoura, E. (2005), *Concept-based Discovery in Mobile Services*. In: Proceedings of the 6th international conference on Mobile data management, Ayia Napa, Cyprus, 2005.

Stoica, I., Morris, R., Karger, D. R., Kaashoek, M. F., & Balakrishman, H. (2001). *Chord: A Scalable peer-to-peer lookup service for internet applications*. In: Conference on Applications, Technologies, Architectures and Protocols for Computer Communications, pp. 149-160. San Diego, California. New York:ACM.

Tanenbaum, A. S., & Steen, M. V. (2007). *Sistemas Distribuídos - Princípios e Paradigmas.* 2ª. edição. São Paulo: Pearson.

Terry, D., Theimer, M., Petersen, K., Demers, A., & Spreitzer, M. (1997). *Flexible update propagation for weakly consistent replication*. In: Symposium on Operating Systems Principles, pp. 288-301. ACM.

Terry, D., Theimer, M., Petersen, K., Demers, A., Spreitzer, M., & Hauser, C. (1995). *Managing update conflicts in Bayou, a weakly connected replicated storage system*. In: Symposium on Operating Systems Principles, pp. 172-183. ACM.

Thomas, R. (1979). A Majority Consensus Approach to Concurrency Control for Multiple Copy Databases. *ACM Transaction Database Systems , 4* (2), pp. 180-209.

Valduriez, P., & Pacitti, E. (2004). *Data Management in Large-scale P2P Systems*. In: International Conference on High Performance Computing for Computational Science-VECPAR, pp. 104-110. Valencia, Espanha. Springer.

Vidal, M., Pacitti, E., Valduriez, P. (2007). *Survey of Data Replication in P2P Systems.* Relatório de Pesquisa, RR-6083, versão 2.

Wilson, B.J. (2007). *Project Jxta book*. Disponível em < http://www.brendonwilson.com/projects/jxta/pdf/JXTA.pdf>. Acesso em 26 de Outubro de 2007.