Um Modelo de Replicação com Garantia de Consistência em Ambientes P2P com Mobilidade

Izalmo P. da Silva, Vanessa Braganholo, Carlo E. Tolla

Programa de Pós-Graduação em Informática (PPGI)
Universidade Federal do Rio de Janeiro (UFRJ)
Caixa Postal 2324 – 20.010-974 – Rio de Janeiro – RJ – Brasil

izalmo@gmail.com, braganholo@dcc.ufrj.br, carlo@nce.ufrj.br

Relatório Técnico NCE-03/08

Abstract. *In unstructured distributed P2P systems there is no logical structure to control the peers coming and leaving the network, which can occur anytime due to mobility. Thus, data exchange with consistence, and data availability are very important. To favor high data availability and to acquire performance gains in dynamic behavior environments, P2P systems use data replication. However, current replication data models have a single failure point, or require complex conciliation algorithms to acquire consistency. This work introduces a replication data model that simplifies the consistency mechanism.*

Resumo. *Em sistemas P2P não-estruturados não existe uma estrutura lógica que controle a entrada e saída de nós da rede, o que pode ocorrer em qualquer tempo devido à alta mobilidade presente. Assim, é de fundamental importância a disponibilização e a troca de dados entre dispositivos presentes na rede com consistência. Para prover uma alta disponibilidade dos dados e obter ganhos de desempenho em ambientes com comportamento dinâmico, sistemas P2P utilizam-se da replicação de dados. Modelos atuais de replicação de dados possuem um ponto único de falha, ou necessitam de algoritmos muito complexos para garantir consistência. Esse trabalho apresenta um modelo de replicação que simplifica o gerenciamento de consistência de dados replicados..*

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 (denominado 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 deatualização. 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, 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 indicando ao servidor primário sua atualização. 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 Lifschtiz (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 muito elevado de processamento e de troca de mensagens. Além disso, são necessários algoritmos muito complexos para resolver o problema de consistência.

Nesse trabalho apresentamos um modelo de replicação que visa garantir consistência de forma simples, 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 une características 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.

A principal preocupação deste trabalho está na manutenção dos itens de dados mais atuais 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.

O restante deste artigo está dividido em seis seções. Nas Seções 2 e 3 são apresentados os principais conceitos de replicação de dados e sistemas P2P, respectivamente. A estratégia de replicação proposta é apresentada na Seção 4 e as simulações sobre o tempo de execução do modelo são apresentadas na Seção 5. Finalmente, a Seção 6 apresenta as considerações finais e direções futuras.

2 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 determinado instante de tempo.

Há duas razões principais para replicar dados: confiabilidade e desempenho. Em primeiro lugar, dados são replicados para aumentar a confiabilidade de um sistema. Se os dados de um sistema foram replicados, é possível continuar trabalhando após a queda de uma réplica simplesmente alternando para uma das outras réplicas. Outra razão é o aumento de desempenho, que é importante quando um sistema distribuído sofre 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]. Neste caso, cada servidor contém uma cópia completa do conjunto de dados do sistema.

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 vazão 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 e 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, as 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*. Nas seções a seguir essas características são explicadas em maiores detalhes.

2.1 Modelo Único Mestre x Múltiplos Mestres

Modelo de replicação de mestre único (Figura 1), também chamado de replicação mestre/escravo, guarda apenas uma cópia primária de cada objeto. As demais cópias são chamadas cópias secundárias. 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), 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.

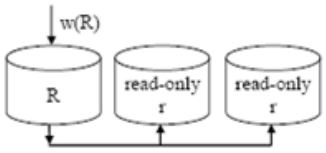

Figura 1 - *Replicação de Mestre Único* [Vidal, Pacitti, Valduriez 2007]*; R é a cópia primária e r a cópia secundária*

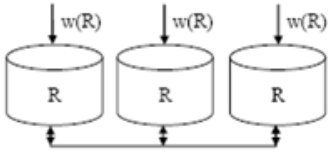

Figura 2 - *Replicação de Múltiplos Mestres* [Vidal, Pacitti, Valduriez 2007]*; R é a cópia primária do mesmo objeto*

2.2 Propagação Síncrona x Propagação Assíncrona

Estratégias de propagação podem ser síncronas ou assíncronas. Na estratégia de propagação síncrona (Figura 3) 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 4), 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.

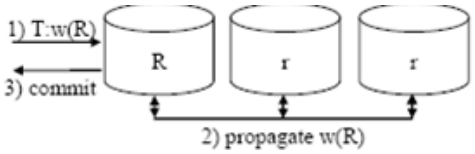

Figura 3 - Princípio de Propagação Síncrona [Vidal, Pacitti, Valduriez 2007].

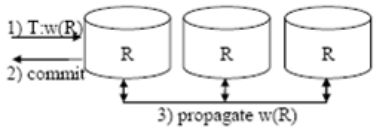

Figura 4 - Princípio de Propagação Assíncrona [Vidal, Pacitti, Valduriez 2007].

2.3 Replicação Total x Replicação Parcial

Replicação total (Figura 5) consiste em armazenar uma cópia de todos os objetos compartilhados em todos os nós participantes do sistema, enquanto que na replicação parcial (Figura 6) 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.

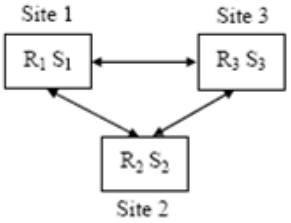

Figura 5 - Modelo de Replicação Total com dois objetos R e S [Vidal, Pacitti, Valduriez 2007].

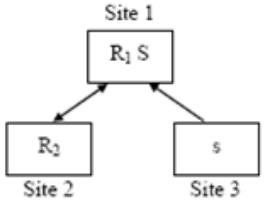

Figura 6 - Modelo de Replicação Parcial com dois objetos R e S [Vidal, Pacitti, Valduriez 2007].

3 Sistemas Ponto-a-Ponto (P2P)

Sistemas ponto-a-ponto 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 2007] ou compartilhamento de dados [Gnutella 2006; Kazaa 2007]. O sucesso dos sistemas P2P se deve ao 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, sistemas P2P podem ser muito úteis no contexto de computação móvel ou computação pervasiva [Satyanarayanan 2001]. 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 que causam problemas de desempenho. Muitas pesquisas têm focado em soluções para 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 da 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 sendo 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.

3.1 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*) [Genç 2005] 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 7 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].

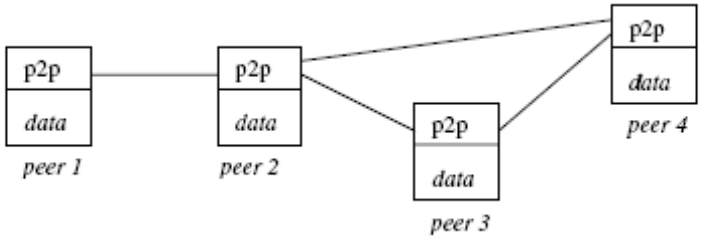

Figura 7 – Rede ponto-a-ponto não estruturada. [Valduriez, Pacitti 2004]

3.2 Sistemas Estruturados

Sistemas 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 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].

3.3 Sistemas com Super-Nós

Redes 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 metadados. 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* envia a requisição, que 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 de 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].

3.4 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 da seção 3.

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

4 Modelo de Replicação Múltiplos Únicos Mestres

O ambiente no qual a proposta foi desenvolvida baseia-se em uma rede P2P não estruturada com mobilidade dos nós, onde cada nó possui uma visão parcial da rede. A visão parcial deve-se ao fato de que cada nó contém uma lista com referências para parte de seus nós vizinhos (nós próximos a ele na rede). De preferência, cada item dessa lista deve representar um nó vizinho vivo na rede escolhido aleatoriamente do conjunto de nós vigentes da rede. Para manter essa lista atualizada, cada nó, em instantes de

tempo periódicos, troca sua visão parcial com seus vizinhos com o objetivo de atualizar sua própria lista e de eliminar os nós que se ausentarem da rede [Tanenbaum e Steen 2007]. Outra característica do nosso ambiente é que apesar da mobilidade dos nós e da possibilidade de falhas de vários nós, não é vislumbrada a possibilidade de particionamento da rede, assim, assumimos que os nós da rede formam sempre um grafo não particionado.

O modelo proposto neste trabalho prevê uma visualização diferente dos modelos de único mestre e múltiplos mestres. Ele se diferencia 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. Permitir atualizações concorrentes através da adaptação do modelo de único mestre, evitando problemas referentes a essas atualizações, mas mantendo simples o controle de concorrência;

b. Possuir 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. Permitir a escolha de um novo mestre em caso de falha do nó mestre do objeto, evitando assim os pontos únicos de falha os quais 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.

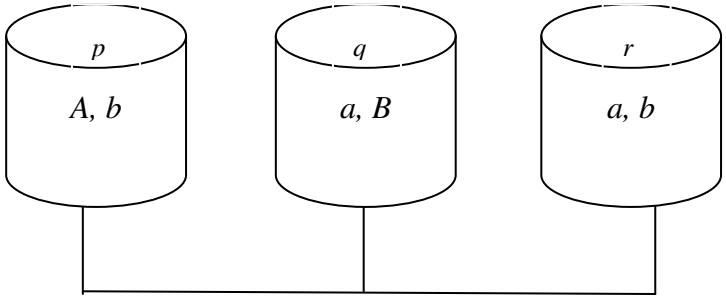

Figura 8 - Representação do modelo múltiplos únicos mestres;*p, q e r* são nós que armazenam objetos *a* e *b*, onde *A* é cópia principal, *a* é réplica e *B* é cópia principal enquanto *b* é a réplica.

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.

A Figura 8 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 feita (esses identificadores são explicados na seção seguinte);
d. Após a alteração, o nó deve retornar ao mestre o objeto alterado, solicitando a ele que finalize a operação;
e. Nesse momento o mestre valida se é possível alterar ou não (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

[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.

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 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. 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 Garantia de 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 garantir 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).

Definindo *N* como o conjunto de nós e *O* como o conjunto de cópias principais de um sistema, podemos afirmar que:

a. Todo objeto criado por *n*, com $n \in N$, deve possuir um *C(V, T, M)* atrelado a ele, onde *M* é o identificador de *n*;
b. Ao criar um objeto, os valores referentes a *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;
c. A cada solicitação de alteração o identificador de transação é incrementado em uma unidade no objeto *o*, onde $o \in O$, 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 a atualização simultânea do objeto sem a necessidade de bloqueá-lo para somente um solicitante.

d. 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, o qual 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..

e. Quando um nó *n*, onde $n \in N$, 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 Sincronização 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.

4.3.1 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 um dos seus vizinhos (selecionado aleatoriamente) todos os objetos dos quais ele é mestre e indica a este para que o mesmo se torne mestre das cópias principais. O nó *n* também replica as informações apontando para o novo mestre.

Se o nó mestre *p* falha, essa falha só será sentida por um nó que desejar atualizar 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 o objeto desatualizado assuma como mestre do mesmo e permaneça por muito tempo com este desatualizado (mais detalhes na seção 4.4), porém 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.

4.3.2 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.

4.3.3 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 algoritmo seta o valor do novo mestre do objeto e retorna um valor booleano indicando se a busca foi bem sucedida (*Verdadeiro*) ou não (*Falso*).

A Figura 9 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]. Uma vez terminada a execução da função, seu valor de retorno é armazenado na variável *valC*.

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.

```
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 9 - Algoritmo para encontrar Novo Mestre

4.3.4 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* $(Vp=Vn \text{ e } Tp>Tn)$.

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 10 apresenta o algoritmo.

Figura 10 - Algoritmo para Nomeação de Novo Mestre

```
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
```

A probabilidade mínima de sucesso na busca por um novo mestre (P_s) pode ser expressa da seguinte forma. 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 afirma que:

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

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.

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 permitindo copiar apenas o que for desejado e à medida que seja necessário.

Para evitar os problemas mencionados acima 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 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.

As principais características da *função de propagação de réplicas* são:

a. É uma função baseada em um cálculo matemático e em alguns parâmetros;

b. 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;

c. Os nós vizinhos que receberão a réplica da cópia original alterada serão sempre diferentes do nó que solicitou a alteração;

d. 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 11 mostra a rede e o objeto presente nos nós após a atualização.

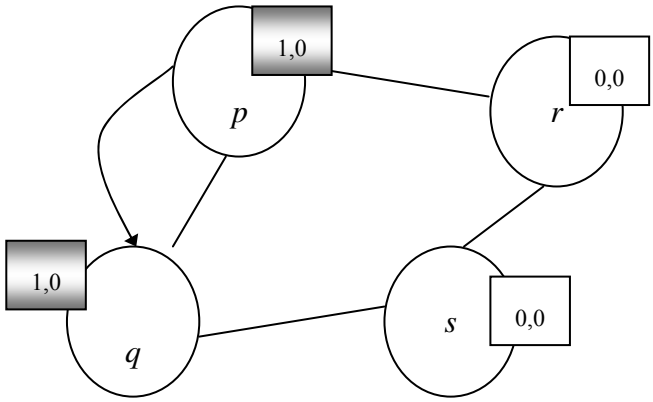

Figura 11 - Exemplo de rede onde a função de replicação é aplicada

Na Figura 11, *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 a informação 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 de a rede ficar sem a informação atual é de 75 %, enquanto com a função cai para 50 % na primeira atualização. A partir da segunda atualização a probabilidade é de 25% e na terceira atualização 0%.

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 afirmar que:

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

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*:

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

Nesta função, quanto maior o número de atualizações, maior será o percentual de nós que conterão o dado mais atual. O gráfico da Figura 12 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.

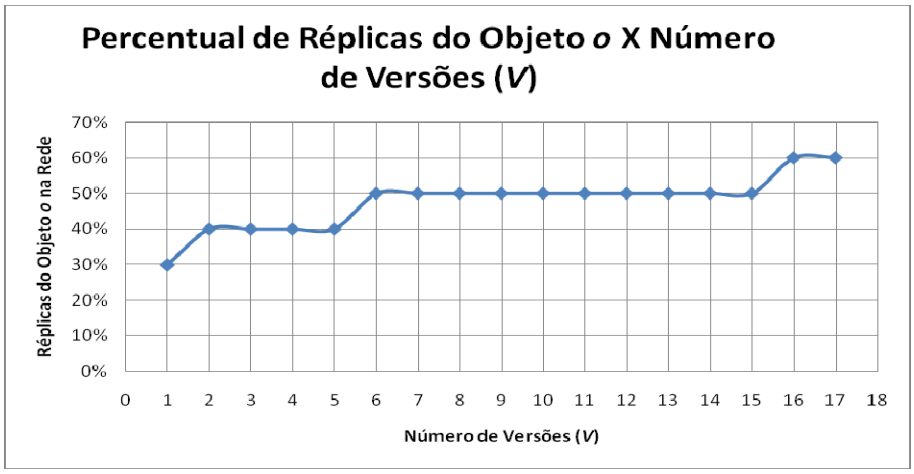

Figura 12 – Percentual de vizinhos que terão a réplica mais atual de um objeto *o* à medida que o número de atualizações aumenta.

4.5 Comparativo

A Tabela 1 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 ele identifique aumentar a disponibilidade dos dados. 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 mesma.

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 na 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 1: 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
Nossa abordagem	*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 trabalhar uma consistência seqüencial em um ambiente P2P não estruturado, sem ter que bloquear as operações aplicadas sobre ele. Sendo assim, a proposta deste trabalho tem forma de garantia de consistência mais simples 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.

5 Simulação

A avaliação da proposta foi feita mediante simulação e leva em consideração o tempo necessário 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 e a conexão entre os nós é composta de forma que cada nó tenha no máximo 20 vizinhos. Apenas um dado será

inserido na rede o qual será replicado para os demais nós. Além disso, o tempo gasto para comunicação entre dois nós é de 1 segundo.

As métricas utilizadas na simulação serão:

- **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.

A primeira 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. O número de nós que possuíam a cópia atual do objeto também foi gerado aleatoriamente. Para o segundo parâmetro, o número mínimo de nós que possuíam a cópia atual do objeto era 0 (zero) e o máximo, 10 (dez). 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. A Figura 13 apresenta os resultados obtidos. Note que a simulação começa com uma rede de 300 nós. O tempo aumenta à medida que mais nós são acrescentados à simulação, já que pode ser necessário percorrer vários nós até encontrar o que contém a informação mais atual.

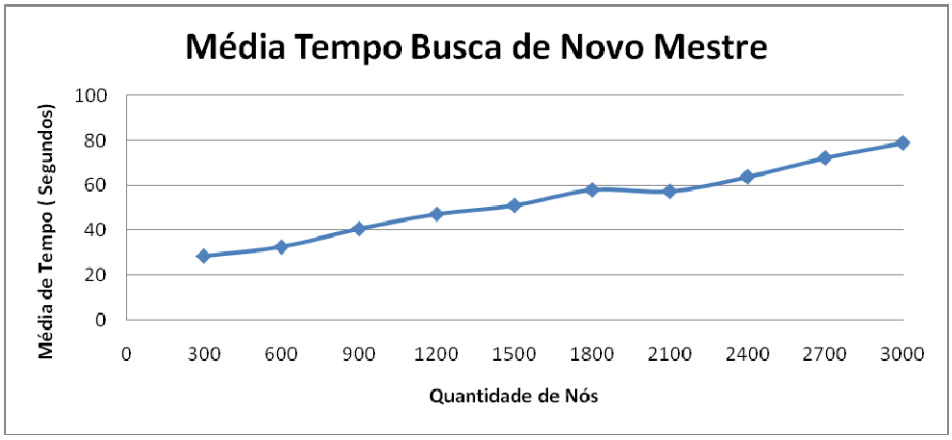

Figura 13 - Tempo médio para execução do algoritmo de localização de novo mestre.

A segunda avaliação é bem parecida com a primeira, 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 13) é possível se ter um percentual bom de novos mestres encontrados.

Para essa simulação, foi escolhida uma rede com 300 nós e 500 simulações foram feitas para cada tempo de simulação apresentado na Figura 14.

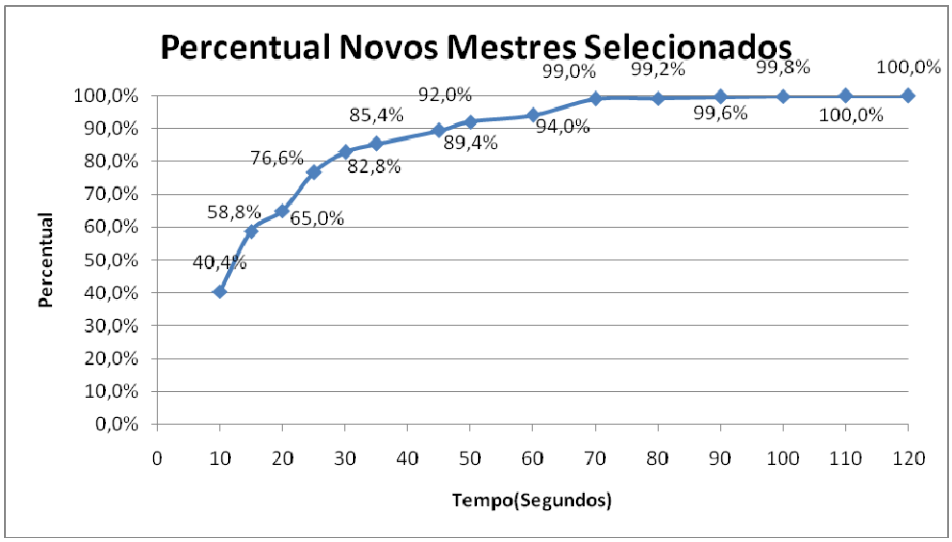

Figura 14 – Percentual de Novos Mestres selecionados X Tempo de Simulação

Os tempos de simulação apresentados nos gráficos acima foram altos, mas ainda assim satisfatórios se levarmos em consideração o dinamismo da rede, a grande quantidade de nós e ainda a necessidade de pesquisar todos os nós para o caso da busca de um novo mestre.

Após analisarmos os resultados, verificamos uma possibilidade de melhoria do algoritmo de busca de novo mestre. Esta melhoria visa 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 a informação mais atual sempre existente na rede. Assim, a cada atualização, o nó mestre seleciona alguns nós para os quais serão enviadas as cópias atuais do objeto principal. Na solução atual, esse nós são selecionados de forma aleatória a cada atualização. A proposta de alteração estaria em não mais realizar uma seleção aleatória dos nós, mas sim uma seleção fixa desses nós, desde que os mesmo se encontrem na rede. Dessa forma, sempre os mesmos nós seriam atualizados com o valor mais atual enquanto eles permanecerem ativos na rede. Quando um desses se ausentar da rede, ele é 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. Abaixo estão os resultados referentes as métricas da simulação: Latência de Consistência e Taxa de Sucesso, utilizando os mesmo parâmetros apresentados na primeira e segunda avaliação.

Aplicando os mesmos parâmetros da primeira avaliação (Latência de Consistência) percebe-se pela Figura 15 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 pesquisa ser feita em apenas alguns nós e não em toda rede, como proposto originalmente.

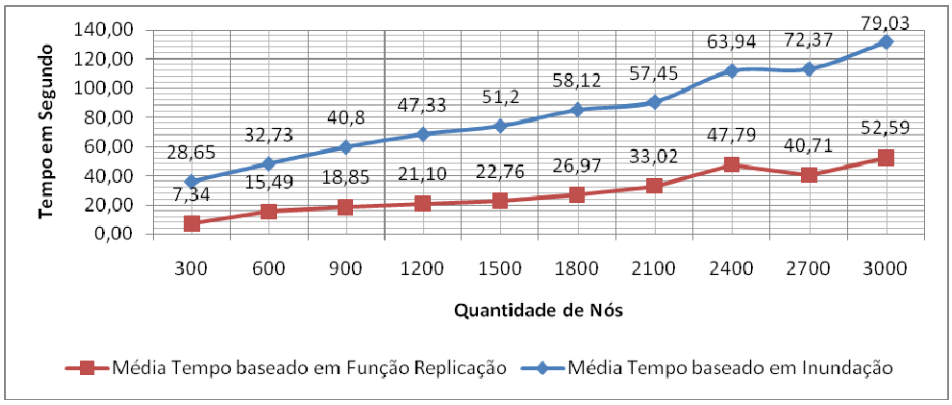

Figura 15 – Comparativo de Tempo entre os algoritmos de busca de novo mestre.

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 atingir esses resultados, sendo necessário recorrer ao algoritmo atual, ou seja, buscar através de inundação. Isso causaria um aumento no tempo de busca, mas neste caso, o tempo seria no máximo igual ao do algoritmo baseado em inundação (que é o utilizado atualmente).

Outra verificação feita refere-se à taxa de sucesso (segunda avaliação) na busca por um novo mestre. Na Figura 16 é feita a comparação entre o modelo de busca por novo mestre atualmente utilizado (através de inundação) e a melhoria proposta baseada na função de replicação.

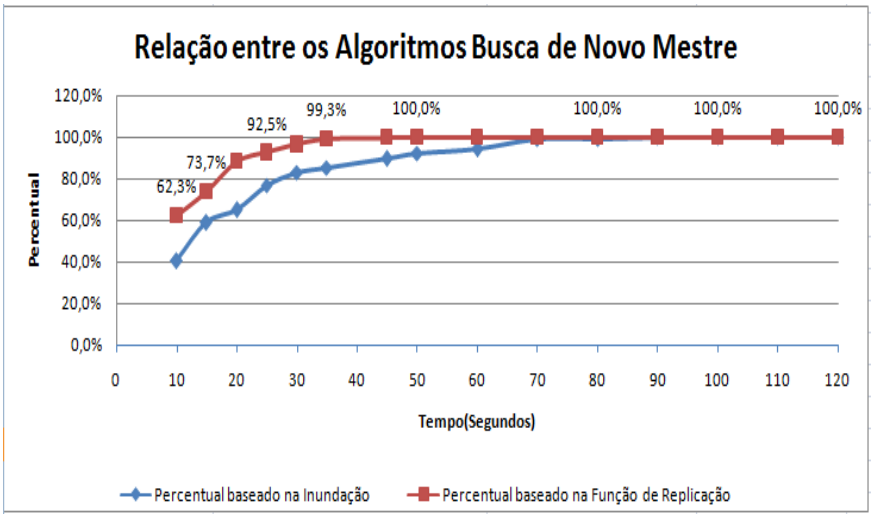

Figura 16 - Comparativo de Percentual de Novos Mestres Selecionados X Tempo de Simulação.

No comparativo da Figura 16 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 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.

6 Considerações Finais

Este artigo apresentou uma proposta de modelo de replicação para ambientes P2P não-estruturados. Este modelo garante a consistência dos dados de forma simples 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.

Como trabalhos futuros propomos o tratamento de problemas referentes a 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? Perguntas como essas e outras encontram-se atualmente em estudo.

7 Referências

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.

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.

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.

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.

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

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.

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.

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

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.

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.