MAC 5701 - Tópicos em Ciência da Computação

Protocolo para compartilhar recursos em uma rede Peer2Peer

Candidato: Vladimir Moreira Rocha
Orientador: Fabio Kon

São Paulo, Junho de 2004

Resumo

Nesse projeto propõe-se um novo protocolo para redes *Peer2Peer* que têm dois objetivos: fazer com que a comunicação entre um *Peer* e seus vizinhos seja eficiênte em termos de latência; localizar nos *Peers* da rede informações dinâmicas, por exemplo recursos disponíveis de um *Peer*.

Para obter uma comunicação eficiente usamos uma estrutura chamada Tabela de *Hash* Distribuída que armazena informações de latência entre *Routers* e *Peers*. Através dessas informações um *Peer* pode descobrir se existem outros *Peers* próximos a ele, utilizando-os na troca de mensagens.

No caso da busca de informações dinâmicas, são propostas duas alternativas: uma baseia-se na busca na vizinhança do *Peer* e a outra na busca através de intervalos indexados. Cada uma das alternativas têm vantagens e desvantagens que serão analisadas neste trabalho.

Este estudo trará fundamentação teórica e prática para posterior utilização no projeto InteGrade.

Sumário

1 Introdução

Dentro dos protocolos de comunicação *Peer-to-Peer* existem diversos aspectos que foram analisados nestes últimos anos, mas basicamente são resumidos a aplicações que descobrem recursos que existem em um outro computador, como documentos, arquivos de musica, etc.

O que acontece quando todos os computadores compartilham a mesma informação? Um exemplo disso é a situação na qual todos têm o mesmo arquivo de música para compartilhar, ou alguma informação como quantidade de memória *RAM* disponível, tamanho do disco rígido, etc.

No caso mencionado anteriormente, não é tão simples obter um computador que cumpra com os requisitos de busca, porque todos têm a informação. É necessário saber se existe algum computador que está perto do meu de forma que eu possa me comunicar rapidamente.

A idéia principal é que a troca de mensagem precisa ser feita de uma maneira rápida. Para isso, a conexão entre os computadores tem que ser baseada nos aspectos físicos e lógicos da rede. Além disso, devemos fazer com que automaticamente os computadores possam conhecer a melhor conexão possível.

O primeiro capítulo apresenta a estrutura que será a base do protocolo que permite uma localização em tempo O(log(N)) de informações espalhadas nos Peers. O segundo e o terceiro capítulo mostram as idéias teóricas do novo protocolo proposto. Finalmente, nos comentários finais, mostram-se os próximos passos da nossa pesquisa.

2 Tabela de Hash Distribuída

Um dos problemas fundamentais nas aplicações *Peer2Peer* é a chamada localização da informação. Em redes de grande tamanho, faz-se necessário prover uma busca eficiente em termos de tempo e mensagens enviadas que evite que, aumentando o número de *Peers* na rede, a eficiência da busca fique prejudicada.

No caso do tempo, vê-se claramente que uma busca em todos os nós da rede é impossível, basicamente por questões de escalabilidade. Se a informação, no pior caso está no último nó, teremos que percorrer todos antes de atingir a resposta, o que ocasionará uma espera muito longa para o nó que fez a requisição.

Já para as mensagens enviadas, como no caso anterior, se fazemos uma busca em um nó que não tem a informação procurada, existirão muitas mensagens viajando na rede e que acabarão por se perder. Isso implicará em uma diminuição na largura de banda.

Para solucionar os problemas antes mencionados, existem sistemas onde a localização de dados é um dos objetivos primordiais. Podem-se citar como exemplos sistemas *Peer2Peer* com controle centralizado (arquiteturas cliente-servidor) ou sistemas de organização hierárquica (*DNS*).

Para redes *Peer2Peer*, existem diferentes tipos de protocolo de busca que são os responsáveis por localizar dentro dela um dado específico.

Um desses protocolos é a chamada estrutura de dados Tabela de *Hash* Distribuída (DHT em inglês) que tem as seguintes características:

- **Balanço de Carga**

  Ao herdar uma propriedade da Tabela de *Hash*, cada chave é dirigida a um nó de forma equitativa.

- **Descentralização**

  Ao manter a idéia de *Peer2Peer*, não existe um nó com alguma funcionalidade mais importante do que outro, ou seja, todos os nós têm o mesmo comportamento.

- **Escalabilidade**

  Independente do tamanho da rede de *Peers*, a *DHT* mantém o custo da busca. Como vemos na figura 1, os Peers unidos a esta estrutura estão conectados como se fosse

um anel, ou seja, uma lista encadeada de *Peers*. Com isso, podemos perceber que existe uma união de todos os *Peers* e nenhum fica isolado.

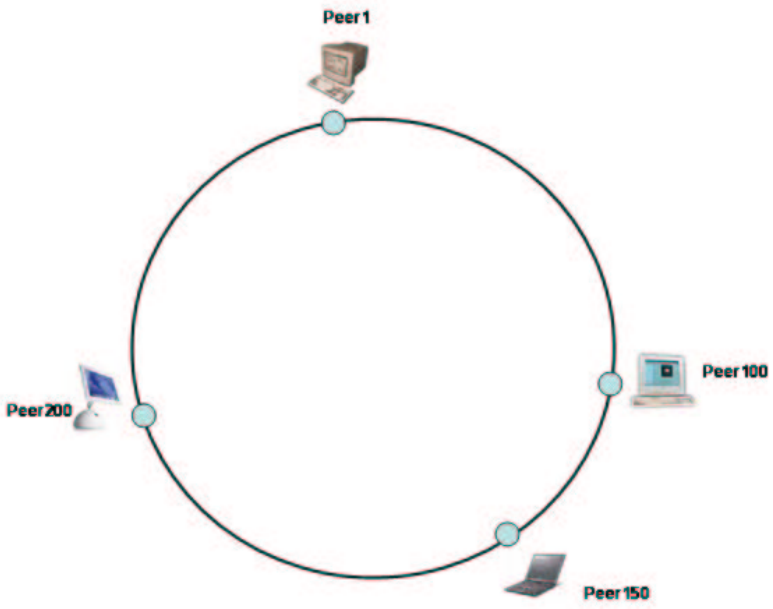

Figura 1: Esquema de um anel para uma estrutura DHT geral.

Toda implementação deste tipo de estrutura especifica dois pontos: como procurar um dado; como novos nós podem unir-se na rede, existindo dentro dele um processo de atualização, explicado mais adiante.

2.1 Busca de um dado

Suponhamos que temos em um dado momento a seguinte estrutura mostrada na figura 2. Como vemos, cada *Peer* é responsável por uma quantidade de chaves. Cada chave é transformada por uma função e mapeada ao identificador do Peer que a contém.

Por sua vez, cada Peer tem um registro indicando seus sucessores, e esse registro está definido da seguinte maneira:

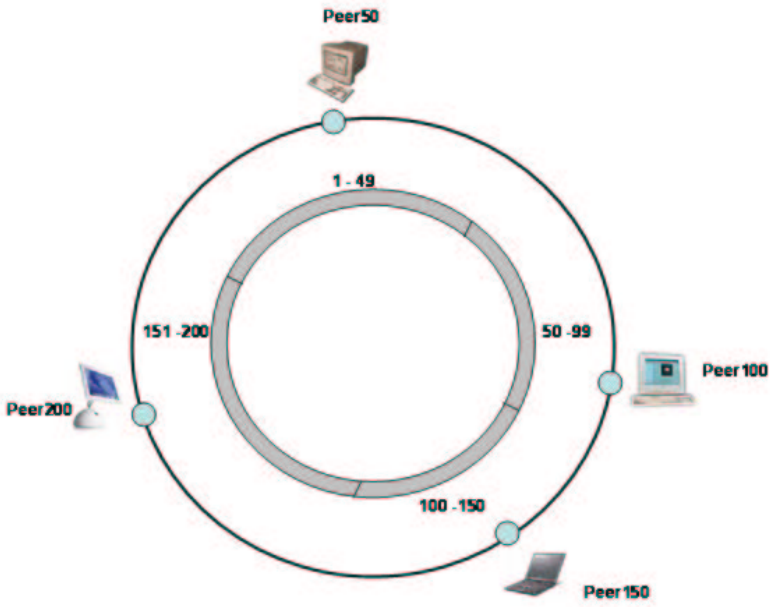

Figura 2: Estrutura das chaves armazenadas pelos Peers.

- Só pode ter log(N) sucessores registrados: com isto evitamos um excesso de informação. Se tivéssemos todos os nós registrados tornaríamos o protocolo pouco escalável. N é a quantidade de *Peers* unidos na rede.
- Cada sucessor registrado será o que está a uma distância potência de 2, até completar o último Peer da Tabela de *Hash* Distribuída.
- Cada sucessor tem a responsabilidade de manter as chaves compreendidas em um intervalo determinado pela estrutura

Para ter uma noção mais clara desses três pontos anteriormente mencionados vejamos a figura 3.

Como vemos, o *Peer* identificado com o número 8 tem a responsabilidade de manter as chaves do intervalo 1 a 8, e em seu registro de sucessores tem os *Peers* 14, 21, 32 e 42.

Agora vejamos a seguinte situação: suponhamos que o *Peer* identificado com o número 8 precisa procurar a chave 54. Os passos seguintes se manifestam na figura 4

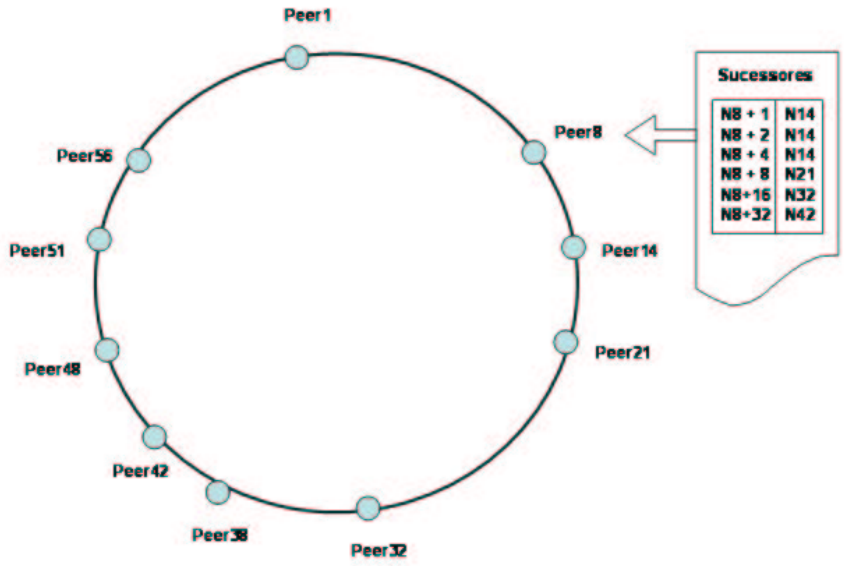

Figura 3: Sucessores do Peer número 8.

O *Peer* número 8 verá em seu registro de sucessores o *Peer* com identificador mais próximo à chave procurada, neste caso o *Peer* número 42. Depois, será feita a consulta pela chave 54 ao *Peer* número 42. Este, por sua vez, verá em seu registro o *Peer* mais próximo à chave procurada, que é o *Peer* com identificador 51. Finalmente, se chegará ao *Peer* com identificador 56, responsável por manter as chaves de número 51 ao 56.

O tempo para executar esta busca é O(log(N)), já que é semelhante a uma busca binária em um vetor ordenado.

2.2 Ingresso de um Peer na DHT

Em um ambiente dinâmico, como as redes *Peers2Peers*, o ingresso e a saída de um *Peer* pode acontecer em qualquer momento. Para manter as propriedades da Tabela de *Hash* Distribuída, faz-se necessário levar em conta as seguintes considerações:

- **Onde alocar o novo *Peer*?**

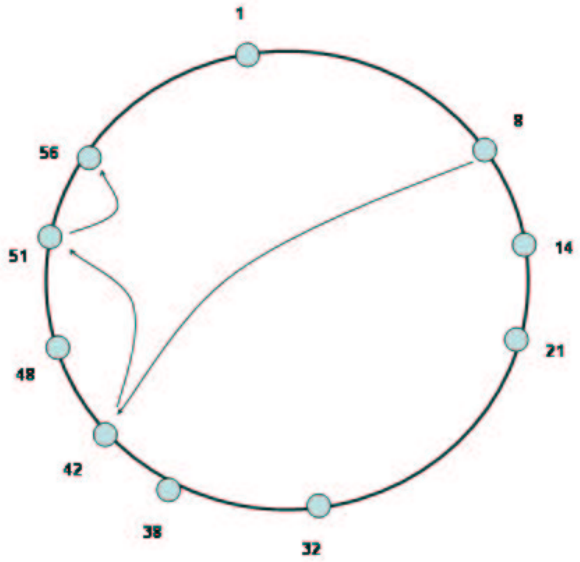

Figura 4: O Peer número 8 procura pela chave 54.

A cada *Peer*, quando ingressa, é dado um identificador único. Esse identificador é gerado com alguma função de *hash* determinada pela aplicação que implementa a *DHT*. Esse identificador, pode ser o IP, alguma credencial, um endereço *CORBA*, etc.

De acordo com o mencionado no parágrafo anterior, teremos um valor numérico não repetido na estrutura que permitirá unir esse novo *Peer*.

Depois, é necessário encontrar o lugar exato onde incluí-lo. Para isto, faz-se uma busca (perguntando a qualquer *Peer* da *DHT*) usando como chave o identificador do novo *Peer*. O caminho traçado por essa busca permitirá saber o nó mais próximo da posição onde teremos que inserir o novo *Peer*. O tempo para executar o alocamento é de O(log(N)), que seria a busca da posição, mais a inserção que leva tempo constante.

- **Atualizar os registros dos outros *Peers* já existentes na *DHT***

É necessário atualizar os registros dos outros *Peers* para que estes tenham conhecimento do novo *Peer* ingressado. Essa atualização esta descrita no artigo *Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications*[7] que prova que isso é executado em O(log(N))

- **Transferência das chaves ao novo *Peer*.**

  Esta última operação move as chaves que agora serão de responsabilidade do novo *Peer*. Somente se precisa extrair, do sucessor imediato do novo *Peer*, as chaves do intervalo compreendido entre o identificador do novo *Peer* até a do identificador do sucessor.

Estes três processos são mostrados na figura 5.

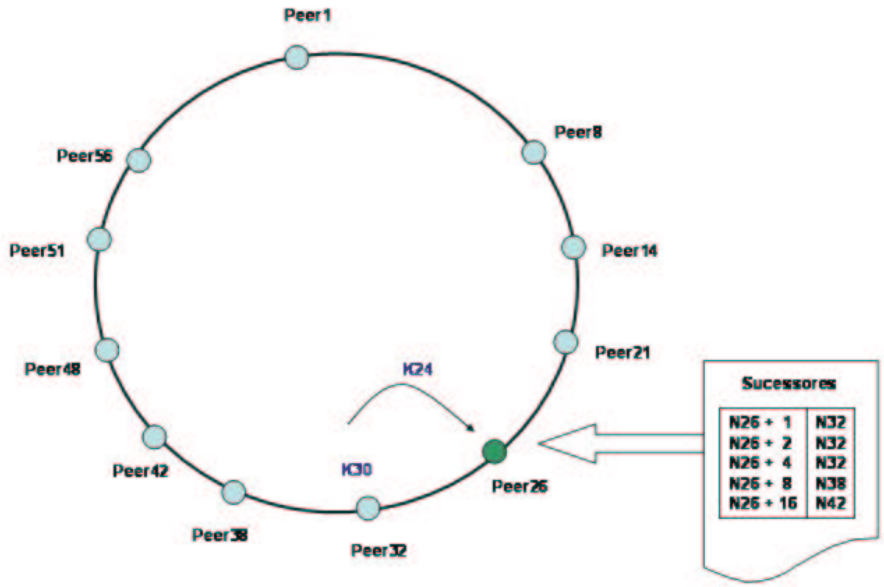

Figura 5: Atualização do Peer número 26 depois que entrou na rede.

2.3 Implementações

Na tabela 1 vemos as diferentes implementações, baseadas nesta estrutura de anel, com suas respectivas linguagens de programação.

Nome DHT	Linguagem
Pastry	Java-Python
Tapestry	Java
Chord	C++
GISP	Java/JXTA
Viceroy	Java/Applet

Tabela 1: Implementações de Tabelas de Hash Distribuídas

3 Protocolo de Comunicação entre Peers

Protocolo, por definição, é um conjunto formal de convenções que manejam o formato e o controle da interação entre unidades funcionais de comunicação.

Na ciência da computação existem diferentes tipos de protocolos que estão dispersos nas diferentes camadas do modelo *OSI*. Como exemplo vejamos a tabela número 2.

Camada ISO-OSI	Protocolos
Camada de Aplicação	HTTP, RPC
Camada de Apresentação	DNS, SNMP
Camada de Sessão	NetBIOS
Camada de Transporte	TCP, UDP
Camada de Rede	IP, ICMP
Camada de Enlace de Dados	ATM, PPP
Camada Física	—

Tabela 2: Camadas Modelo *OSI*.

3.1 Protocolos para redes Peer2Peer

Estes protocolos também se encontram nas diferentes camadas do modelo *OSI*, por exemplo:

- Na camada de Apresentação e de Sessão, os protocolos geralmente são os que se encarregam do envio das mensagens.
- Para a localização de dados utilizam a camada de Rede com o roteamento das mensagens, ou seja, por qual caminho é melhor enviá-los.

3.2 Protocolo para conectar Peers em ambientes colaborativos.

Antes de descrever os detalhes do protocolo, vejamos quais são os principais motivos pelo qual se faz necessário criar um novo protocolo.

Primeiro, nos protocolos *Peer2Peer* existentes na atualidade, temos que a informação não é redundante. No caso de *Gnutella* ou do *Gisp* do *Jxta*, a informação compartilhada são arquivos que estão somente em alguns Peers. Esses protocolos se focam então na busca e disseminação da informação.

Segundo, a estrutura e formação dos *Peers* em relação à vizinhança e ao tempo entre eles não é importante. Isto se deve ao fato de que não se sabe a priori com quem o *Peer* deve conectar-se para fazer as buscas. Devemos recordar que a informação pode estar em qualquer *Peer*.

Vejamos então como podemos solucionar esses problemas anteriormente mencionados.

3.3 Descrição do Protocolo

Ao analisarmos a figura 6, temos uma rede *Peer2Peer* geral, onde podemos observar que a estrutura de conexão dos *Peers* é totalmente aleatória, ou seja, não segue nenhum padrão definido.

Se um computador precisasse comunicar-se com outro computador por meio de mensagens, o mais adequado seria conectar-se com o mais próximo em tempo de latência e

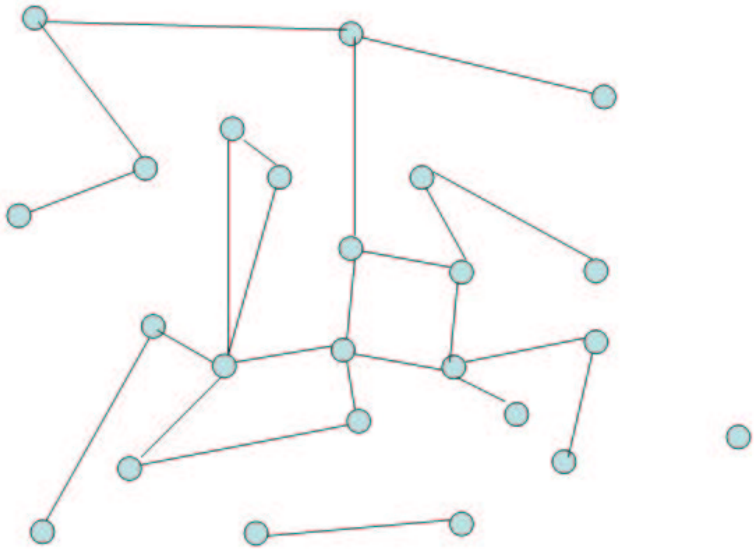

Figura 6: Estrutura de uma rede Peer2Peer geral.

não com outro qualquer. Isso melhoraria a performance no envio e recepção das mensagens.

O protocolo definido a seguir tem por objetivo responder duas perguntas essenciais:

- Como podemos encontrar os *Peers* que estão perto um do outro?
- Como encontrar os *Peers* que satisfaçam uma requisição feita?

3.4 Ingresso de um Peer na Rede

Podemos ver na figura 7 uma estrutura já formada onde um *Peer* deseja ingressar. Nessa estrutura temos um anel de Peers. Isto se deve à formação apresentada no capitulo de Tabela de *Hash* Distribuída.

No caso do ingresso temos duas opções:

- É a primeira vez que o *Peer* vai conectar-se na rede.

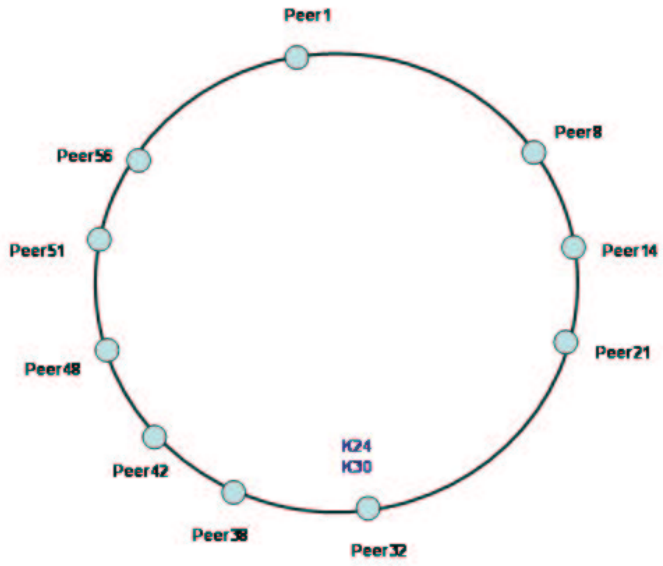

Figura 7: Estrutura ja formada onde um Peer vai ingressar.

- O *Peer* já tinha se conectado antes e deseja voltar

3.4.1 Primeira vez

Na primeira vez que um *Peer* se conecta na Rede, é preciso conhecer outro *Peer* que lhe permita entrar nessa rede.

A solução proposta é a de ter os *Peers* mais estáveis em uma pagina web que todos conheçam. Estes *Peers* têm como característica o fato de a probabilidade de saírem da rede ser muito baixa.

Existe outra possibilidade, que é a busca através do *Broadcast* para ver se alguém atende à requisição, mas isto gera um problema de performance.

Uma das características da página é que possuam uma quantidade pequena de *Peers*, evitando com isso uma demora no processamento dela. Outra característica é que esta página não seja necessariamente única, ou seja, podem ter outras páginas organizadas por

grupos, paises, etc, o que permite escalabilidade. Por exemplo, no diagrama 3 vemos um exemplo de uma possível página que tem o IP e a porta onde o *Peer* escuta as mensagens.

PeerID	Porta
www.austria.eu.net	80
newhome.weblogs.com	156
blo.gs	80
www.freesoft.org	1225
www.topping.ru	678
www.hildrum.com	2674
www.candystand.com	1475
www.ping.be	3442
www.tiscali.ch	2463
www.planetpod.de	2411
www.pinggolf.com	4213
www.ping.de	80
www.dnsstuff.com	3352
www.insecure.org	1113
www.classic-trash.com	80

Tabela 3: Página Web com Id dos *Peers* estáveis

A seguir os passos necessários para ingressar na Rede

1. O processamento da página consiste em obter aleatoriamente uma quantidade de identificadores, *Peers*, com os quais o novo candidato poder-se-ia conectar. O identificador do *Peer* tem que ser único e depende da pessoa que o implementa escolher de que tipo vai ser.

2. Dos identificadores obtidos, escolhemos o *Peer* com menor latência, deixando registrado no Repositório de *Peers* esses tempos. O registro dos Peers no repositório segue certas regras que serão propostas mais adiante.

É importante destacar que existem diversas formas de obter a latência. A mais conhecida é o *ICMP (Internet Control Message Protocol)* com seu comando *Ping* que obtém, do teste de conexão, o tempo para chegar a mensagem.

3. Com o mais próximo dos *Peers* encontrados no passo 1, usamos os recursos físicos de rede. Para isto, temos que obter os *routers* do caminho entre o candidato e o *Peer* mais próximo. Para a obtenção desses routers temos como exemplo a idéia do comando *traceroute* em Linux.

4. Para cada *router* do passo anterior, perguntamos à *DHT* se existe alguma informação sobre esse router. O tipo de informação armazenado no objeto *Router* será explicado a seguir.

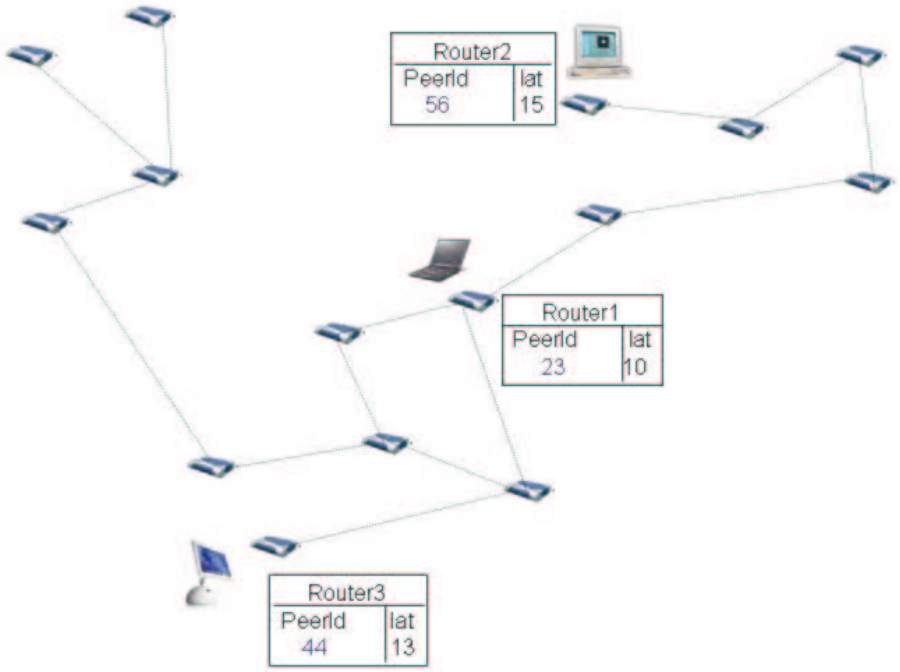

Figura 8: Objetos Routers armazenados nos Peers.

Vendo a figura 8 temos que um *Peer*, designado pela *DHT*, tem responsabilidade por um objeto de tipo *Router* o qual contém informação sobre um *Peer* próximo a ele e a latência entre o *Router* e esse *Peer*.

O protocolo verifica neste passo se o *Peer* conhecido pelo *router* está, em termos de latência, mais próximo que o *Peer* escolhido no passo 2.

Se o *Peer* armazenado no *Router* está mais perto que o escolhido então:

- O candidato conecta-se logicamente com o *Peer* armazenado no objeto *Router*
- Se a latência entre o escolhido e o *router* analisado é menor do que a latência do *Peer* armazenado, então se deve atualizar o *Peer* colocando a informação do escolhido.
- O algoritmo termina e não se analisa os outros *Routers*.

3.4.2 O peer já tinha ingressado antes

Neste caso, não devemos preocupar-nos em procurar os *Peers* na pagina web descrita anteriormente. Agora simplesmente os obtemos do Repositório de *Peers*. Depois, continuamos com o item 2 dos passos para ingressar na Rede como se fosse a primeira vez.

3.5 Saída de um Peer da Rede

Nesse processo o *Peer* se desconecta da Rede. A desconexão pode ser provocada por diversos motivos como:

- O *Peer* sai normalmente
- O *Peer* cai por problemas técnicos

De acordo com a guia de desenho de protocolos robustos para Internet [1], devemos estar cientes que a queda de um *Peer* é possível e com uma alta probabilidade, então, quando um *Peer* sai da rede, o protocolo deve:

- Atualizar a *DHT*. Isso é proporcionado pela implementação da Tabela de *Hash* Distribuída.
- Atualizar as referências que os *Peers* conectados tinham quando o *Peer* saiu.
- Atualizar os objetos *Routers* que tinham armazenado o *Peer* desconectado.

No primeiro caso, a implementação da *DHT* proporciona implicitamente a saída dos *Peers* da rede. Desta forma, o protocolo não precisa preocupar-se com isso. Os dois últimos casos serão analisados a seguir.

3.6 Atualização do Protocolo

Como sabemos até aqui, a *DHT* é que proporciona a base para a localização de objetos *Routers* que guardam por sua vez *Peers* que se encontram na vizinhança. Vejamos a continuação os diferentes tipos de atualizações do protocolo.

3.6.1 Atualização de Referências

A atualização de referências se produz quando um *Peer* entra ou sai da rede. Existirão *Peers* que mantinham referência direta para esse *Peer*. Neste caso, cada *Peer* é responsável por verificar, a cada instante de tempo, se o canal de comunicação está funcionando. Caso não esteja, o protocolo terá que seguir com os passos para conectar-se na rede, já explicado anteriormente.

3.6.2 Atualização dos Objetos Routers

Essa atualização é produzida quando um *Peer* sai da rede. Aqui o protocolo deverá verificar se que o canal de comunicação do Peer armazenado no objeto roteador está funcionando. Caso não esteja, elimina-se o objeto.

3.6.3 Atualização do Repositório de Peers

É possível eliminar as referências que não estejam disponíveis, ou seja, que o canal de comunicação esteja fechado (significa que o *Peer* saiu da Rede). Para isso, o protocolo terá que verificar, com um teste de conexão, se os *Peers* estão vivos ou não. Nesse caso, gera-se um problema de performance ao ter que enviar e receber mensagens de todos os *Peers* do repositório, o que resulta em uma saturação da largura de banda.

A outra atualização é a do Repositório, ou seja, na obtenção de novos *Peers*.

Nesta etapa, pergunta-se a algum dos vizinhos tomados de forma aleatória, se conhecem outros *Peers* que não existam no Repositório e que cumpram com o requisito de

estar entre as primeiras T posições em termos de latência. T é um valor determinado pelo usuário do protocolo. Veremos mais adiante, no capítulo Repositorio de *Peers*, que o Repositório é uma lista ordenada de *Peers* pela latência.

No caso de não encontrar nenhum, definiremos uma variável TTL[5] que será a profundidade em que a mensagem será propagada nos vizinhos para obter novos *Peers*.

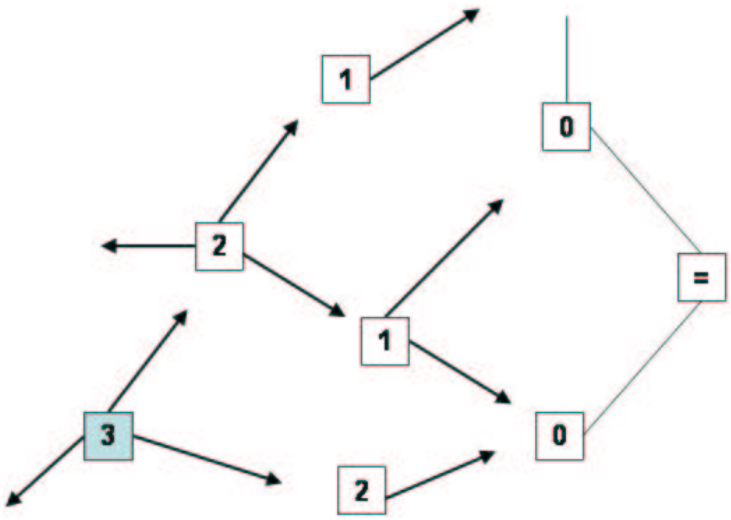

Figura 9: Propragação de uma messagen pelos Peers com o TTL = 3.

Na figura 9 podemos observar o caso quando TTL=3 e sua propagação. Um detalhe importante é como evitar a geração de ciclos entre os *Peers*, ou seja, como evitar que a pergunta seja refeita para um mesmo *Peer*.

3.7 Busca de um requisito

Até agora, solucionamos o primeiro objetivo que era a comunicação e busca dos *Peers* que estão perto um do outro.

O segundo objetivo é encontrar os *Peers* que satisfaçam um certo requisito. Nesse caso, temos duas possibilidades:

- **O requisito é estático**.

  Por exemplo um documento pdf, um programa de multiplicação de matrizes para correr de forma paralela, um arquivo de configuração do protocolo, etc.

- **O requisito é dinâmico**.

  Por exemplo suponhamos que precisamos encontrar 50 *Peers* que tenham capacidade *RAM* disponível no momento de 20 Mb e espaço em Disco Rígido de 5 Mb.

O primeiro caso se faz diretamente tratando com a *DHT*, ou seja, pode-se deixar em *Peers* distribuídos os objetos a compartilhar.

O segundo caso é mais complicado. Como a informação é dinâmica, temos que ter alguma forma de poder acessar essa informação sem sobrecarregar um *Peer*. Um exemplo disso é deixar um servidor que controle a informação dinâmica de todos os *Peers*, ou seja, que responda a uma requisição e devolva uma resposta. Isto se faz impraticável em uma Rede de *Peers* muito grande e portanto não é escalável.

Neste ponto, existem algumas alternativas as quais mostraremos a seguir.

Uma alternativa é perguntar à vizinhança do Peer se o requisito pode ser conseguido. Com isso, asseguramos que a troca de mensagens será feita entre *Peers* com uma latência mínima, o que permite uma boa performance. Tem que se levar em conta as seguintes considerações:

- A vizinhança do *Peer* pode ser obtida do Repositório de Peers, ou perguntando diretamente ao mais próximo. A partir deste vizinho a mensagem de busca pode ser propagada.

- No caso da propagação da mensagem, temos que evitar os ciclos gerados quando se pergunta novamente a um *Peer* já conferido.

Outra alternativa é a de se manter intervalos indexados que permitam uma busca eficiente dos requisitos. Nesse caso a performance cai drasticamente ficando pouco escalável. O exemplo mais claro é quando todos os *Peers* estão dentro de um intervalo.

Neste caso a atualização será feita sobre um só *Peer*, o que gera uma arquitetura cliente-servidor.

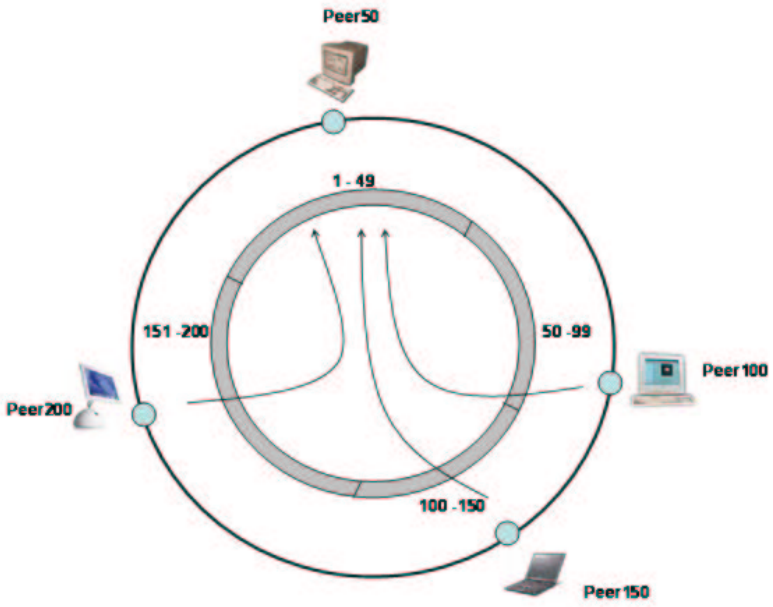

Figura 10: Todos os Peers atualizam num só Peer gerando uma arquitetura Cliente-Servidor.

Na figura 10 podemos observar que um *Peer* é o responsável por um intervalo indexado da memória *RAM* disponível. No caso em que todos os *Peers* fiquem dentro desta margem, teremos o problema do servidor mostrado anteriormente.

Além disso, existe um problema que tem a ver com a vizinhança dos *Peers*. A resposta que teremos em uma busca por intervalos indexados serão *Peers* que não necessariamente estão perto do que fez a pergunta. Com isso, a troca de mensagens terá um custo maior.

Em qualquer um dos dois casos, seja propagação ou indexação, é importante destacar que a análise dos *Peers* necessários para cumprir a requisição será feita localmente.

3.8 Adequar o protocolo a uma instância especifica.

Este protocolo esta sendo desenvolvido para o projeto InteGrade. Na arquitetura mostrada na figura 11, existe uma camada de *GRM (General Resource Manager)* que controla uma quantidade de *LRMs(Local Resource Manager).*

Figura 11: Arquitetura do nós do Integrade.

Além das referências que o *GRM* tem com estes *LRMs*, o *GRM* tem o serviço *Trader* de *CORBA*, muito parecido a um Banco de Dados, que pode devolver referências a *LRMs* que satisfaçam aos critérios de uma certa requisição.

O comportamento dos *GRMs* é parecido com o dos *Peers* mostrado no protocolo, portanto, o protocolo funcionara dentro dos *GRM*.

4 Repositório de Peers

No protocolo analisado, o Repositório é uma característica muito importante já que com ele podemos possuir um conhecimento armazenado dos vizinhos do *Peer*.

Este Repositório corresponde a uma lista ordenada de forma decrescente por latência, armazenada num arquivo, que é atualizada cada certo instante de tempo.

PeerID	Latência
www.austria.eu.net	10.01
newhome.weblogs.com	10.33
www.freesoft.org	12.87
www.topping.ru	12.87
www.hildrum.com	12.98
www.tiscali.ch	13.53
www.planetpod.de	15.08
www.insecure.org	15.48
www.classic-trash.com	17.65

Tabela 4: Exemplo do Repositório de Peers

Na tabela 4 vemos que essa lista proporciona informação para o *Peer* que se desconecto da rede e volta a entrar, como mostrado no protocolo de ingresso de um *Peer* da rede.

5 Comentários finais

Com a base teórica pronta, a primeira etapa será implementar o protocolo em uma linguagem de programação. Como os *GRM* da arquitetura do Integrade estão escritos em *Java* provavelmente essa será a linguagem utilizada.

A segunda etapa consiste em verificar se o protocolo cumpre com fatores importantes de desenvolvimento deste tipo específico de aplicação, tais como: escalabilidade, tempo de resposta, quantidade de mensagens trocadas etc. Para isso, neste momento está se estudando uma ferramenta de simulação de redes de grande envergadura que permite analisar o protocolo e obter resultados dos tipos supracitados.

Caso os resultados sejam satisfatórios, o protocolo deverá ser incluído como o protocolo de comunicação inter-aglomerados do projeto InteGrade.

Referências

[1] Tom Anderson, Scott Shenker, Ion Stoica, and David Wetherall. Design guidelines for robust internet protocols. *SIGCOMM Comput. Commun. Rev.*, 33(1):125–130, 2003.

[2] Fabio Kon Tomonori Yamane Christopher Hess Roy Campbell and M. Dennis Mickunas. Dynamic resource management and automatic configuration of distributed component systems. *USENIX*, 2001.

[3] Miguel Castro, Michael B. Jones, Anne-Marie Kermarrec, Antony Rowstron, Marvin Theimer, Helen Wang, and Alec Wolman. An evaluation of scalable application-level multicast built using peer-to-peer overlays. In *ACM SIGCOMM 2001*, aug 2001.

[4] Ian Foster and Carl Kesselman. *The Grid: Blueprint for a new Computer Infrastructure*. Morgan Kaufmann, 1999.

[5] Bo Leuf. *Peer to Peer*. Addison-Wesley, 2002.

[6] Antony Rowstron and Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Con-*

ference on Distributed Systems Platforms (Middleware), pages 329–350, November 2001.

[7] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Infocom'03*, apr 2003.

[8] Andrew Tanenbaum. *Computer Networks*. Prentice Hall, 1996.

[9] Brendon J. Wilson. *JXTA*. New Riders, 2002.