GD^2: Um Serviço de Diretórios P2P para Gerenciamento de Informações em Redes Heterogêneas

Tarciana Dias da Silva[1], Carlos Kamienski[2], Stênio Fernandes[3], Djamel Sadok[3]

[1]Escola Politécnica, Engenharia da Computação – Universidade de Pernambuco (UPE)
tarciana.dias@ecomp.poli.br

[2]Universidade Federal do ABC (UFABC)
cak@ufabc.edu.br

[3]Universidade Federal de Pernambuco (UFPE)
{stenio, jamel}@gprt.ufpe.br

Abstract. *Information management is a key feature for the successful deployment of service architectures that involve highly distributed, dynamic, collaborative, and heterogeneous networks. Current solutions fail to meet important requirements of those networks since they have limited support for dynamicity of networks, nodes and information, or flexible information retrieval mechanisms for satisfying user´s needs. In this paper, we propose a Global Directory Service based on Distributed Hash Tables (DHT) and Hilbert Space Filling Curves that provides distribution and flexibility for information retrieval. Performance analyses reveal that proposed mechanisms are scalable with the number of networks, nodes, and information.*

Resumo. *O gerenciamento de informação é uma característica essencial para o sucesso na implantação de arquiteturas de serviços envolvendo redes distribuídas, dinâmicas, colaborativas e heterogêneas. As soluções atuais não conseguem tratar adequadamente os requisitos dessas redes porque tem suporte limitado à dinamicidade das redes, nós e informação, além de não terem mecanismos flexíveis para recuperação de informação que satisfaçam as necessidades dos usuários. Para isso, esse artigo propõe um Serviço de Diretórios baseado em DHT e na teoria de curvas de preenchimento de espaço de Hilbert. Uma avaliação de desempenho revela escalabilidade nos mecanismos propostos considerando a quantidade de redes, nós e informação.*

1. Introdução

Nas últimas décadas, mudanças significativas vêm ocorrendo na infra-estrutura de comunicação que evoluiu de redes homogêneas de interconexão de computadores estacionários para uma grande gama de redes de acesso com altos níveis de complexidade, heterogeneidade e dinamicidade. Particularmente, é observada uma grande proliferação de tecnologias de redes sem fio e serviços que exploram mobilidade e recursos multimídia para dispositivos como *smartphones* e *tablets*. Essas novas demandas geram impactos em vários aspectos do gerenciamento das redes, especialmente na obtenção, agregação, distribuição e busca de informações.

O gerenciamento de informação é uma característica essencial para o sucesso na implantação de arquiteturas de serviços envolvendo redes distribuídas, dinâmicas, colaborativas e heterogêneas. Para uma implantação exitosa de arquiteturas de gerenciamento de serviços futuros que necessitam de grande interoperabilidade, um

elemento decisivo é uma Infraestrutura de Serviços de Informação [1], que oferece acesso ubíquo e relativo ao contexto a vários componentes de gerenciamento.

As soluções atuais para gerenciamento de informações em redes heterogêneas não conseguem tratar adequadamente os novos requisitos devido ao suporte limitado à dinamicidade das redes, nós e informação, além de não terem mecanismos flexíveis para recuperação de informação que satisfaçam as necessidades dos usuários. Abordagens típicas são baseadas num componente centralizado que periodicamente obtém informações dos elementos distribuídos, processa e redistribui para a rede inteira ou para grupos de dispositivos e/ou usuários interessados. A desvantagem mais clara da abordagem centralizada está relacionada com a existência de um ponto único de falha, que obviamente pode ser remediada com condições de contorno existentes na literatura. Ainda, a escalabilidade é uma desvantagem adicional principalmente em ambientes altamente dinâmicos onde tipicamente trocas de informações são muito freqüentes. Por outro lado, soluções distribuídas e escaláveis, como *Distributed Hash Tables* (DHT) [2], possuem limitações na realização de consultas flexíveis (ex.: consultas a chaves não exatas ou por faixas não são permitidas tipicamente).

Esse artigo propõe e avalia um serviço de diretórios distribuído inovador para redes heterogêneas, onde os participantes (usuários ou provedores) podem registrar informações globais de modo que os usuários possam obter informações sobre recursos e serviços disponíveis na rede através de consultas flexíveis e eficientes. Essas informações são representadas através de um modelo global de informações, que pode ser facilmente estendido dinamicamente à medida que novos recursos globais entram no sistema e precisam ser representados no diretório. Devido à grande escalabilidade e inerente suporte à dinamicidade, um mecanismo de rede Peer-to-Peer (P2P) baseado em DHT é usado. Para tratar das limitações de consultas flexíveis de DHT, é proposto um novo mecanismo para armazenar e consultar dados baseado em curvas de preenchimento de espaço de Hilbert (HSFC) [3][4]. São apresentadas descrições completas dos algoritmos de indexação usados para inserir e buscar os elementos de dados no DHT. Vários funcionais de um serviço de diretórios global são tratados nesse trabalho, como: a) suporte para mobilidade freqüente e atualização de informações; b) suporte a dinamismo da informação devido à colaboração entre redes para evitar replicações e inconsistências; c) tratamento eficiente e flexível de consultas; d) escalabilidade, tolerância a falhas e robustez em ambientes distribuídos.

Algumas contribuições são trazidas pelos resultados apresentados nesse artigo tanto para a área de serviços de diretórios globais para ambientes móveis e heterogêneos. Primeiro, as decisões de projeto usadas para o diretório global reconhecem o *tradeoff* existente entre duas abordagens opostas: adaptar um serviço de diretórios existente para suporte a alta mobilidade, escalabilidade e dinamicidade ou adaptar uma estrutura altamente distribuída para adicionar consultas flexíveis. A segunda abordagem foi escolhida, usando DHT com o conceito das curvas de Hilbert. Segundo, o processo de geração de chaves para DHT foi significativamente alterado para oferecer localidade para consultas em faixa ao mesmo tempo em que se preserva a escalabilidade. Terceiro, foi construído um simulador em cima do arcabouço OMNeT++ com a implementação das principais características de um serviço de diretórios. Por último, foi realizada uma avaliação de desempenho baseada em simulação que revelou que os novos mecanismos são escaláveis com o número de redes, nós DHT e quantidade de informação.

Na seqüência desse trabalho, a seção 2 apresenta a fundamentação teórica e os trabalhos relacionados. A seção 3 apresenta o diretório Global Distribuído para Gerenciamento de Informação de Redes e a seção 4 apresenta detalhes da sua implementação; a seção 5, a metodologia, e a seção 6, os principais resultados. Por fim, a seção 7 apresenta as conclusões e expectativas de trabalhos futuros.

2. Fundamentação Teórica e Trabalhos Relacionados

Redes heterogêneas e dinâmicas requerem interoperabilidade, mobilidade transparente, colaboração e composição, como motor para sua evolução e desenvolvimento. Requisitos de mobilidade de usuários e serviços em tais ambientes implicam que constantes atualizações de informação (e.g., conteúdo e estrutura) devem ser suportadas. Mecanismos de gerenciamento responsáveis pela mobilidade de usuários e serviços estão fora do escopo deste trabalho e assumimos que eles existem e estão disponíveis. Em [1], propõe-se uma Infraestrutura de Serviço de Informação (ISI) capaz de coletar, filtrar e correlacionar eventos, e prover informações de contexto para aplicações. Apesar de seu propósito ter sido no contexto de gerenciamento de mobilidade, o objetivo foi fornecer serviços para suportar coleta de informações de rede e tomada de decisões.

Serviços de diretórios como X.500 [5], LDAP [6] não são adequados para ambientes heterogêneos por uma série de razões. Primeiramente, eles não são escaláveis com o número de registros devido à sua arquitetura centralizada. Depois, eles são altamente otimizados para operações de leitura, enquanto atualização e inclusão de novas informações são mais comuns em ambientes dinâmicos. Ainda, a sua estrutura está organizada como um espaço de nomes estático e hierárquico, logo mobilidade e atualizações constantes não são diretamente suportados.

UDDI [7] é uma alternativa que vem de *web services* que foi projetado para operar em ambientes centralizados. Logo, compromete a escalabilidade, tolerância a falhas e robustez para um diretório para ambientes móveis e heterogêneos. Há variações como em [8], objetivando uma solução escalável, porém, como a arquitetura original foi projetada para atuar em ambientes centralizados, tem-se os desafios inerentes a tornar este tipo de arquitetura escalável. Por outro lado, tecnologias Peer-to-Peer (P2P), particularmente a de tabelas hash distribuídas (*DHT*) [9], são comprovadamente escaláveis e eficientes. Buscas são realizadas em *O(log N)* saltos, onde N é o número de nós na rede *overlay*. No entanto, o uso de DHT traz novos desafios, como a organização semântica dos dados armazenados na rede. Nesta, resultados da busca somente são retornados quando a chave de busca é exatamente igual à usada para inserir o dado.

Curvas de Hilbert (HSFC) [3][4] vêm sendo usadas para indexação multidimensional em áreas como banco de dados tradicionais, compressão de imagens e sistemas de informação geográficos. Sua principal característica é fazer um mapeamento de um espaço n-dimensional para um unidimensional, ao mesmo tempo em que a localidade é preservada. Ou seja, quando índices unidimensionais possuem valores aproximados, assim também o será para o espaço multidimensional deles correspondente. Dois conceitos têm uma grande importância no contexto de curvas Hilbert: *derived-keys* e *n-points*. Uma *derived-key* é o resultado unidimensional obtido de um conjunto de dimensões. Por exemplo, se há um espaço bidimensional de terceira ordem, como é o mostrado na Figura 1 (a), o ponto D da figura representa a *derived-key* cujas dimensões correspondem a (000, 011). Este par (000,011) é chamado de *n-point* da *derived-key*.

Terceira ordem porque há 3 bits para representar cada coordenada de cada dimensão. Quanto mais bits para representar a dimensão, maior é a ordem, e mais precisa a representação do ponto no espaço. A Figura 1 de (b) a (d) representam, respectivamente, a curva na primeira, segunda e terceira ordem, num espaço bidimensional. Observa-se que a curva é evoluída de forma recursiva e o número de *derived-keys* aumenta de acordo com a ordem da curva. Temos 4 *derived-keys*: (00, 01, 10, 11) em (b), 16: (0000 a 1111) em (c) e 64: (000000 to 111111) em (d).

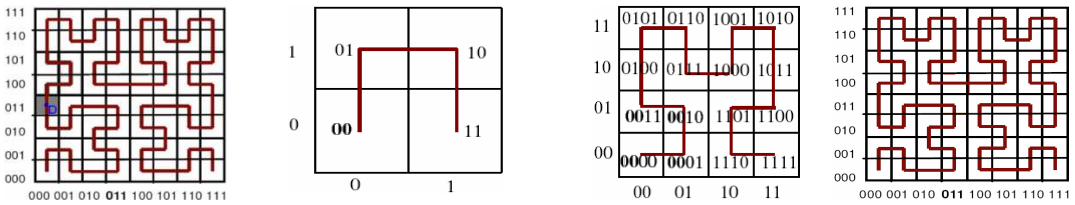


Figura 1. Curvas de preenchimento de espaço de Hilbert (HSFC).

Em [13], aplica-se HSFC em *web services* e num contexto P2P. As informações são armazenadas via Chord e podem ser consultadas de forma flexível. Porém, não é detalhado como os índices são gerados e como os dados que satisfazem são resgatados. O mecanismo de otimização da busca descrito se resume à estratégia inerente ao uso de HSFC. Neste trabalho, o foco é dado no domínio de redes dinâmicas, além de prover otimizações aos mecanismos de indexação e busca, que são explicitamente detalhados, e ainda, é apresentada uma avaliação de resultados bem mais completa.

3. Um diretório Distribuído para Gerenciamento de Informação de Redes

3.1 Modelo de Informação e Arquitetura lógica do GD^2

A Figura 2 (a) apresenta um modelo de informações simplificado para o GD^2. Podemos observar quatro entidades principais: serviço, usuário, rede e nó. Assumimos que cada serviço é identificado através de uma definição de serviço bem conhecida, identificável por todas as redes existentes. Embora existam muitos esforços para padronizar serviços, uma análise detalhada sobre isso está fora do escopo deste trabalho. Este modelo de informação também pode ser estendido para suportar o compartilhamento de informações e descoberta de serviços em ambientes de virtualização de rede.

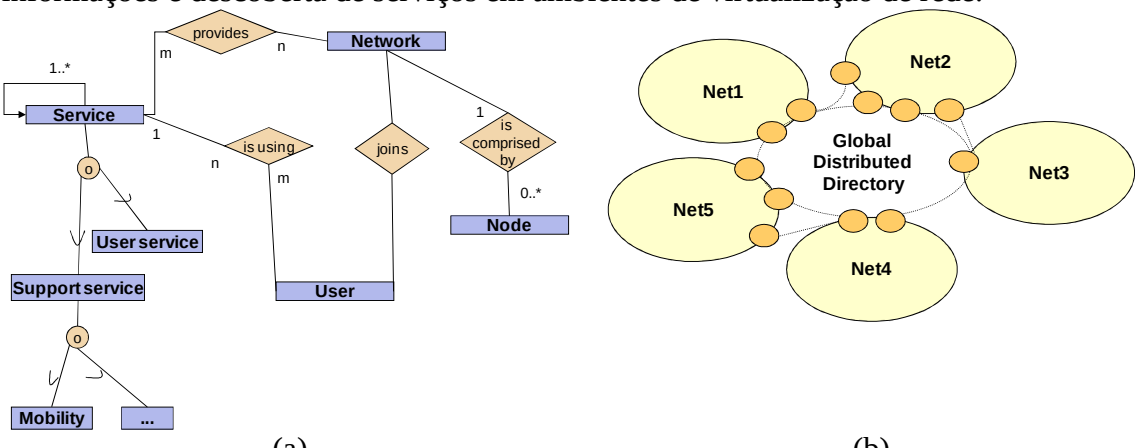


Figura 2. (a) Modelo de informação e (b) visão da arquitetura lógica

Vale ressaltar que um serviço pode conter um conjunto de outros serviços (conforme auto-relacionamento em Figura 2 (a)). Além disso, a entidade de serviço pode ser especializada em *User Service* (ex.: aplicações de voz) ou *Support Service* que, por sua vez, pode ser ainda mais especializado em mobilidade, segurança etc. Não há padronização de um modelo global de informação para representar redes heterogêneas e este modelo é considerado como um estudo de caso. Portanto, ele pode ser aperfeiçoado à medida que novas informações de rede ou os recursos vão sendo descobertos.

O GD^2 foi projetado para ter uma estrutura distribuída, composto de um ou mais nós de cada rede. Esses nós formam uma estrutura de *overlay*, como mostrado na Figura 2 (b). Cada rede deve publicar as suas informações globais de acordo com o modelo de informação. Além disso, uma rede pode acessar através de outra rede GD^2 participante, que pode ser realizado por um acordo de colaboração anterior entre elas.

3.3 Estruturas de Inserção (GD^2 indexes)

Nós derivamos índices para representar o modelo de informação e, em particular, as entidades e os relacionamentos entre eles, como mostrado em Figura 3 (a). O NetID representa o identificador da rede, enquanto o índice USER deve ser usado para armazenar a rede que o usuário está atualmente conectado. As instâncias dos índices (índex instances) representam as informações armazenadas no diretório.

O índice SERVICE_USER armazena informações derivadas da relação entre as entidades SERVICE e USER. Este índice é útil, por exemplo, quando se quer descobrir se um usuário chamado Anne está usando um serviço de TV em sua rede. O índice PROTOCOLOS visa armazenar os protocolos atuais disponíveis numa rede específica. Finalmente, o índice SERVICE aceita consultas sobre quais redes oferecem um serviço específico de um subtipo específico, enquanto o índice FULL_MODEL armazena todas as informações disponíveis associado ao identificador de rede.

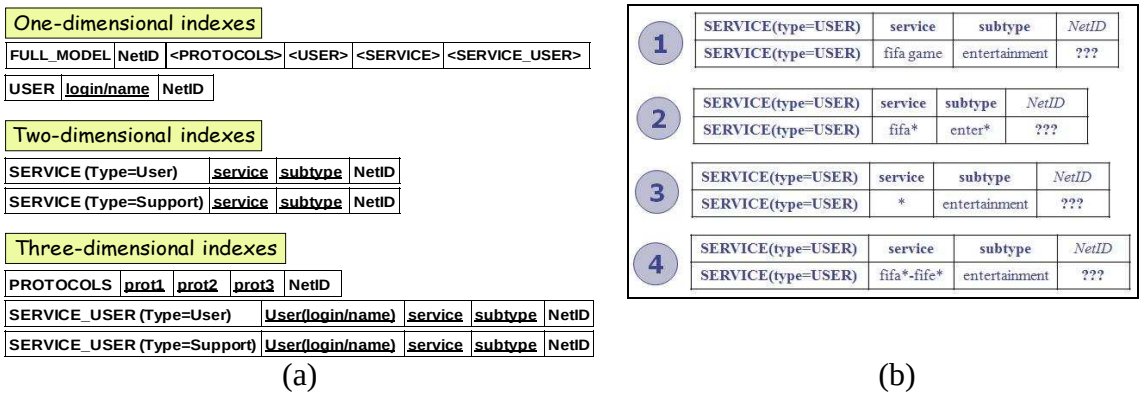


Figura 3. (a) Exemplos de índices no GD2 (b) exemplos de consultas

Os índices GD^2 são classificados de acordo com seu número de dimensões ou metadados (são os campos na Figura 3 (a) que estão sublinhados e em negrito). Estas dimensões são associadas com um valor, que é, exceto para o índice FULL_MODEL, o identificador de rede (representada pelo campo *NetID*). Quando várias redes apresentam instâncias de índice em comum, o campo NetID consiste de um conjunto de identificadores de rede. Como exemplo, considere duas redes identificadas por *NetID* 54 e 76, e um *user service* chamado "fifa", cujo subtipo é entretenimento. Se este serviço só é fornecido por essas redes, o conteúdo do campo *NetID* pode ser "54, 76".

A Figura 3 (b) mostra exemplos de consultas relacionadas ao índice SERVICE(tipo = USER). O GD^2 aceita requisições que podem não conter todos os metadados preenchidos, pois os requerentes não sabem ou não precisam de todas as palavras-chave associados aos índices envolvidos, mas apenas parte deles. Assim, pode haver consultas parciais, contendo apenas um subconjunto de metadados que compõem o índice.

Na primeira consulta, o usuário do GD^2 envia uma solicitação para descobrir redes (*NetID*) com a informação exata especificada no serviço campos e subtipo. Em outras palavras, esta consulta é para uma rede que oferece um serviço chamado "jogo fifa", no qual subtipo é entretenimento. Na segunda consulta, o usuário solicita um serviço que começa com "fifa", e cujo subtipo começa com "enter". A terceira consulta especifica qualquer serviço cujo sub-tipo é "entretenimento". Neste caso, apenas uma dimensão foi especificada num índice bidimensional. Podemos também especificar intervalos para dimensões, como mostrado pela quarta consulta.

4. Implementação da arquitetura GD^2

Esta seção descreve a implementação do GD^2. Como consultas podem não ter chaves exatas, o primeiro passo é aperfeiçoar o mecanismo DHT para suportar consultas flexíveis. Para isso, o GD^2 utiliza mecanismo para preparar os dados quando da sua inserção, via algoritmo de indexação. Além disso, os dados são resgatados pelo algoritmo de busca, também apresentado nesta seção.

4.1. Algoritmo de Indexação

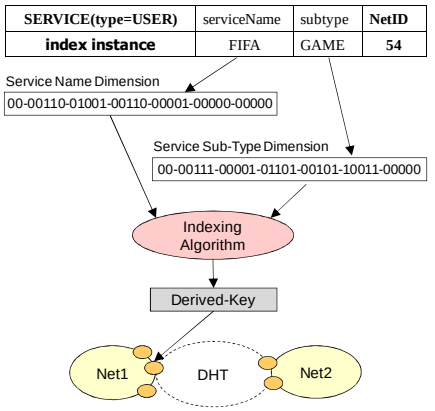
Nosso algoritmo de indexação traduz dimensões (apresentadas em Figura 4 (a)) numa *derived-key* da curva de Hilbert, que por sua vez será usada na estrutura DHT. No caso dos índices unidimensionais FULL_MODEL e USER, a chave DHT será, respectivamente, os campos NetID e login/name. Nos demais índices, o campo NetID representa o valor de cada chave na DHT, e este valor corresponde às redes que possuem os recursos específicos ou relacionamentos especificados pelas dimensões.

Antes de executar o algoritmo de indexação, os valores das dimensões, em nomes, são transformados em valores das dimensões, em bits, que por sua vez serão os *n-points* (seção 2). Neste trabalho, utilizamos uma curva de Hilbert na trigésima segunda ordem para um espaço bidimensional, que justifica o uso da máquina de estados representada na Figura 5(a). Para cada espaço dimensional, há uma máquina de estados específica.

Campos do índice foram limitados a acomodar no máximo 6 caracteres onde cada caractere será representado por 5 bits, somando no total 30 bits. Os dois bits restantes são deixados para uso futuro. A letra ‘a’ representa 00001 (1), letra ‘b’ 00010 (2) e assim por diante até a letra ‘z’, que corresponde a 11010 (26). A correspondência entre as letras e os bits é feita do terceiro bit mais significativo conforme mostra a Figura 4 (a). Tanto o algoritmo de indexação quanto o de busca podem ser facilmente estendidos para dimensões e ordens maiores da curva, seguindo a mesma lógica.

Depois de ter os valores das dimensões representados por bits, o algoritmo de indexação é executado à medida que os bits do *n-point* são lidos. Como exemplo, vamos considerar o n-point <110,100>. O estado inicial é sempre o estado 0, correspondente à raiz da árvore de Hilbert como mostrado pela Figura 5(a). Os primeiros bits lidos do n-point são 1 para a dimensão *x* e 1 para a *y* [<**1**10, **1**00>]. De acordo com o estado 0 da Figura 5(a), este *n-point* corresponde à *derived-key* 10, e o próximo estado deste *n-point*

é o próprio estado 0. A *derived-key* é então deslocada para a esquerda em duas posições (linha 6, Figura 4 (b)). Os próximos bits a serem lidos são: 1 para a dimensão x e 0 para a dimensão y [<11**1**0, 1**0**0>]. No estado 0, observa-se que a *derived-key* correspondente ao *n-point* 10 (segundo lido) é 11 e o próximo estado é o estado 2 (Figura 5(a)). A *derived-key* então passa a ser 1011. O. O próximo n-point é 00 [<11**0**, 10**0**>] cuja derived-key é 10 considerando o estado atual, 2. Logo, a derived-key final correspondente ao n-point [<110, 100>] é 101110.



(a)

Indexing Algorithm

INPUT: Dimension values in bits
OUTPUT: Derived-key

```
1 select the state machine to be used
2 derived-key <- blank
3 while (∃ unread bits in each dimension)
  {
4    check current state of state machine
5    key <- get derived-key for n-point
6    derived-key <- derived-key << 2
7    concatenate key into derived-key
8 }
```

(b)

Figura 4. Algoritmo de indexação. (a) inserção de uma instância de índice em GD2. (b) Pseudo-código.

O algoritmo de indexação é usado para todos os índices do GD^2 exceto para o FULL MODEL, que associa o campo *NetID* com todas as instâncias de índices relacionados a uma rede específica. Assim, é feita a ligação entre o identificador da rede e seu conjunto de informações completo. Neste caso, não há necessidade um mecanismo especial já que a chave a ser inserida na DHT é o identificador da rede, enquanto que o valor é a informação contida nos índices PROTOCOLS, USER, SERVICE e SERVICE_USER.

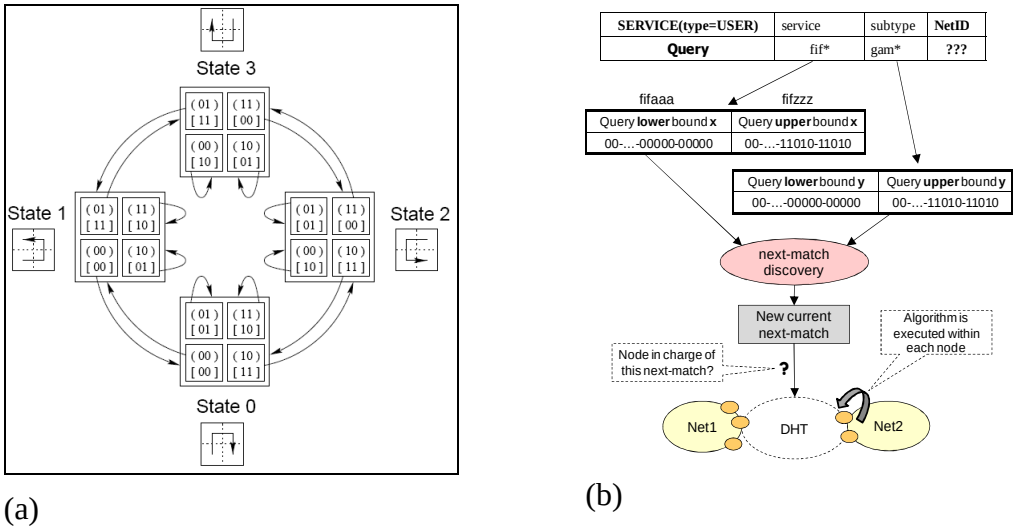
Nosso mecanismo opera com uma diferença em relação ao mecanismo padrão da DHT: a função *hash* não é usada para gerar as chaves. Isto acarretaria o "espalhamento" ou distribuição uniforme do conteúdo para os nós existentes e a idéia aqui é preservar localidade entre os dados semanticamente relacionados. As *derived-keys* obtidas para cada instância de índice são diretamente usadas para inserir informação na rede overlay.

4.2. Algoritmo de Busca

Numa DHT, uma informação que satisfaça uma consulta como "fif*" pode, no pior cenário, ser espalhada sobre a rede inteira. Assim, achar todos os dados correlacionados a esta consulta seria impraticável. Por isso, o algoritmo de busca desenvolvido para o GD^2 suporta consultas flexíveis, enquanto mantém os benefícios da escalabilidade.

O algoritmo de busca do GD^2 é dividido em duas partes. Primeiro, há a descoberta dos nós DHT que podem conter elementos de dados (ou *derived-keys*) que satisfazem a consulta. Uma vez que o(s) nó(s) é(são) descoberto(s), é preciso achar as chaves, se houver, dentre os elementos que o nó DHT é responsável por armazenar. Para isso, é utilizado o algoritmo de *next-match* [3]. O algoritmo de *next-match* objetiva encontrar um ponto minimamente maior (ou igual) ao *next-match* atual e dentro do espaço

definido pela consulta, mas que não necessariamente existe na rede DHT. Portanto, um *next-match* é um ponto na curva de Hilbert que satisfaz a consulta.



(a) (b)

Figura 5. (a) Máquina de estados da curva de Hilbert bidimensional. (b) algoritmo de busca.

Primeiro, o algoritmo de *next-match*, usando 0 (zero) como parâmetro de *next-match* atual é aplicado, a fim de se descobrir o primeiro ponto na curva de *Hilbert* que pertença à região da consulta. Como exemplo, considere uma consulta do tipo *range* composta das dimensões 'fif*' e 'gam*'. Podemos deduzir seus limites (*query bounds*) como 'fifaaa…fifzzz' e 'gamaaa…gamzzz', respectivamente, onde 'fifaaa' e 'gamaaa' são os *lower query bounds* enquanto 'fifzzz' e 'gamzzz' são os *upper*. Depois, os *query bounds* especificados com nomes são passados para *query bounds* especificados com bits, fazendo-se a mesma correspondência discutida na seção 4.1. Então, o novo *next-match* atual para a consulta é gerado, e o próximo passo é consultar a rede *overlay* para descobrir qual nó DHT é responsável por esse novo *next-match* encontrado. Isto representa o início do nosso algoritmo de busca na rede DHT.

Quando o nó responsável pelo *next-match* é encontrado, o próximo passo seria consultar o dado dentro do armazenamento do nó para verificar se este *next-match* é um dado real (existe no nó). Logo, o mecanismo de busca do GD^2 envolve dois procedimentos: a busca de um nó DHT responsável por um *next-match*, e uma vez que este nó seja encontrado, a busca de todos os *matches* dentro deste nó que satisfazem a consulta.

Elementos de dados são armazenados em ordem ascendente dentro do armazenamento de dados do nó. O *next match* atual é comparado a cada uma das chaves armazenadas no nó da DHT – linhas 3, 6 e 19 da Figura 6. Se igual, um elemento que pertence à região de consulta foi encontrado e é um dado existente na DHT. Se não, e se o *next-match* atual é menor que o elemento armazenado, é preciso calcular um novo *next-match*, passando o elemento armazenado como *next-match* atual para o algoritmo de descoberta do *next-match* (linhas 7 e 8). Se o cálculo do *next-match* retornar zero como valor (linha 10), isto significa que o nó e a rede não possuem qualquer elemento de dado que satisfaça a consulta, pois os possíveis elementos que satisfariam seriam aqueles entre o *next-match* anterior e o elemento armazenado no nó indicado pelo índice *i*. Caso o novo *next-match* encontrado não seja zero (linha 13), isto significa que este *next-match* é minimamente maior que o *next-match* anterior e também está na região de consulta.

Logo, compara-se este novo *next-match* com o mesmo elemento armazenado pelo nó (linha 14). Se idênticos, encontrou-se um elemento da DHT armazenado pelo nó que satisfaz a região de consulta. Caso contrário, continua-se e repete-se o mesmo procedimento com o próximo elemento armazenado pelo nó.

Se a chave dentro do nó DHT é menor que o *next match* atual (linha 19), verifica-se se o *next-match* atual é ainda maior que o último elemento armazenado (linha 20), a fim de otimizar o tempo de processamento. Se for, então a busca dentro neste nó é finalizada e logo é preciso encontrar o próximo nó da DHT que tenha mais *matches* satisfazendo a consulta. Se não for, aplicamos um mecanismo de otimização para encontrar a chave imediatamente menor do que o *next-match* atual (linha 25). Isto é feito porque uma busca linear com todos os elementos armazenados pelo nó para fazer as devidas comparações com o *next-match* atual é extremamente custoso.

Em resumo, há três possibilidades durante a busca: (a) Encontrar a chave imediatamente menor (via otimização) do que o *next-match* atual e continuar a busca conforme descrito acima. (b) encontrar a chave dentro da estrutura de armazenamento do nó igual ao *next match* atual. Isto representa uma chave dentro da DHT que é uma *derived-key* contida na região de consulta. (c) Achar a chave maior do que o *next-match* atual. Neste caso, é necessário recalcular um novo *next-match* onde a chave maior passa a ser o parâmetro do algoritmo de descoberta do *next-match*. (c.1) Em caso de o algoritmo de descoberta fornecer zero como resultado, a busca pode ser finalizada conforme já explicado.

Search Algorithm

INPUT: Query and initial current next-match
OUTPUT: Data elements (keys)

```
1  while(i < dataStorageSize) {
2    k <- dataStorage->getKeyAtPos(i);
3    if(k==current_next_match){
4      Add key/value pair to result set
5      i++;
6    }else if(current_next_match < k ){
7      current_next_match <- k
8      Calculate next-match;
9      Update current_next_match value
10     if(next_match==0){
11       Search is finished
12       break;
13     }else{
14       if(k==current_next_match){
15         Add key/value pair to result set
16         i++;
17       }
18     }
19   }else{ //current_next_match > k
20     if(lastKey < current_next_match){
21       Next-match greater than last
22           element in node storage
23       break;
24     }else{
25     Define first key < current_next_match
26       ...
27     }
28   }
29 }
```

Figura 6. Algoritmo de busca dentro de um nó DHT

Depois de finalizar a busca no nó, retorna-se para o nó que solicitou a consulta todas as chaves encontradas. Em caso de *next-match*=0 não ter sido alcançado, isto significa que a busca precisa continuar já que pode haver mais elementos de dados pertencendo à região de consulta em outros nós da rede. Numa rede DHT, todos os nós são organizados em ordem ascendente de acordo com seus identificadores. Um nó é responsável pelo armazenamento de todas as chaves no intervalo entre o identificador do seu predecessor e seu identificador. Logo, o novo *next-match* é calculado, passando-se o identificador do nó mais 1 (um) como parâmetro para o algoritmo de descoberta do *next-match*. Se depois disso o novo *next-match* encontrado for zero, então não há mais elementos de dados existentes na rede overlay que satisfaçam a consulta e as métricas para aquela consulta podem ser coletadas. Por outro lado, caso o novo *next-match*

encontrado não seja zero, o nó na DHT responsável por armazenar este *next-match* tem que ser consultado (seguindo o mesmo procedimento já descrito) e pode estar armazenando mais elementos que satisfazem a consulta.

5. Resultados

Foi realizada uma avaliação de desempenho baseada em simulação para validar o GD^2 usando o simulador *OverSim* [10] que suporta protocolos P2P como Chord [11] e Pastry [12]. Foi modificado o módulo DHT do simulador e foi desenvolvida uma biblioteca que implementa o mecanismo GD^2 (descrito na seção 4) e o algoritmo de *next-match*. Cada rede é vista do ponto de vista de seu modelo de informação, e por isso sua topologia e outras características não são levadas em consideração. GD^2 suporta armazenamento de informação pelos nós da rede bem como consultas (inclusive as flexíveis). Consultas são geradas da seguinte maneira: para cada índice inserido, é gerada uma consulta exata (Q0), 2 inválidas (Q1) e 4 consultas do tipo *range* (Q2). Consultas inválidas são importantes porque representam uma possível ação do usuário, ex.: uma informação solicitada que não está disponível no diretório. O conjunto de consultas geradas é armazenado na estrutura de dados dentro do módulo *Global Observer* do simulador ao mesmo tempo em que os valores dos índices são inseridos na DHT. No tempo de simulação, consultas são randomicamente escolhidas na estrutura de dados onde as consultas são armazenadas via distribuição uniforme.

As seguintes métricas são usadas para avaliar o desempenho do mecanismo do GD^2: Número de elementos de dados (1), número de nós de dados (2), número de nós de processamento (3), número de mensagens trocadas no *overlay* (4), número de mensagens por consulta (5) e tempo de consulta (6). Em (1), tem-se o número total de elementos de dado encontrados por cada tipo de consulta, onde um elemento de dado corresponde a uma *derived-key* armazenada na rede *overlay* DHT. Em (2), o número de nós onde elementos de dados para uma consulta específica são encontrados. O melhor resultado é quando a taxa representada pelo número de nós de processamento dividido pelo número de nós de dados é igual a 1, logo sempre que um algoritmo é executado no nó, elementos de dados são encontrados. Em (3), o número de nós cujos elementos armazenados estão sendo comparados durante a consulta, ou seja, número de nós que efetivamente participam da consulta. Já os nós de dados efetivamente armazenam os dados. Em (4), o número de mensagens enviadas ao *overlay* a cada consulta, aquelas que buscam pelo nó responsável por uma chave. Em consultas exatas ou inválidas, há apenas uma mensagem enviada para o *overlay*. Em (5), o número total de mensagens para processar uma consulta específica. Pelo mecanismo utilizado pelo simulador, no mínimo 6 (seis) mensagens são precisas quando se tem uma busca com chave exata. Para uma consulta do tipo *range* os nós podem precisar trocar um número elevado de mensagens. Por fim, (6) representa o tempo para cada consulta ser finalizada.

Para a realização dos experimentos foram variados o número de instâncias de índices (em 10, 20, 40, 70 e 100) e o número de redes (em 100, 250, 500, 750, 1000), com o objetivo de aferir a escalabilidade da solução. A taxa de chegada de consultas (em 1/50, 1/35, 1/20) segue a distribuição de *Poisson*.

5.1. Efeito do número de instâncias de índices

Nos primeiros experimentos, variou-se a quantidade de informação armazenada no diretório. O número de instâncias de índices assume 5 níveis diferentes (10, 20, 40, 70 e

100), a taxa de chegada da consulta é de 1/35 segundos. Nós consideramos 100 redes e 100 nós DHT. A duração de cada experimento é de 1000 segundos.

À medida que aumentamos o número de instâncias de índices, há também um aumento dos elementos de dados encontrados para a consulta Q2 (Figura 7 (a)), o que é um resultado esperado pois há mais informações armazenadas. Consultas Q0 e Q1 ficam com 1 e 0 elementos de dados, respectivamente, considerando não haver replicação na DHT. Os valores das métricas para consultas exatas e inválidas são os mesmos para todos os experimentos – o que é um comportamento esperado – exceto a métrica do tempo de consulta. Tem-se o mesmo número de nós de processamento (um), mesmo número de mensagens trocadas (seis) e mensagens enviadas ao *overlay* (uma). Por isso, todos os comentários feitos são para as consultas do tipo *range* (Q2). Porém Q0 e Q1 são ilustradas nas figures para facilitar a comparação com as do tipo *range* (Q2).

Os gráficos mostram uma métrica pela sua média de acordo com a variação dos respectivos fatores. O intervalo de confiança de 99% foi calculado, mas as barras verticais não são mostradas pois não há sobreposição entre as curvas.

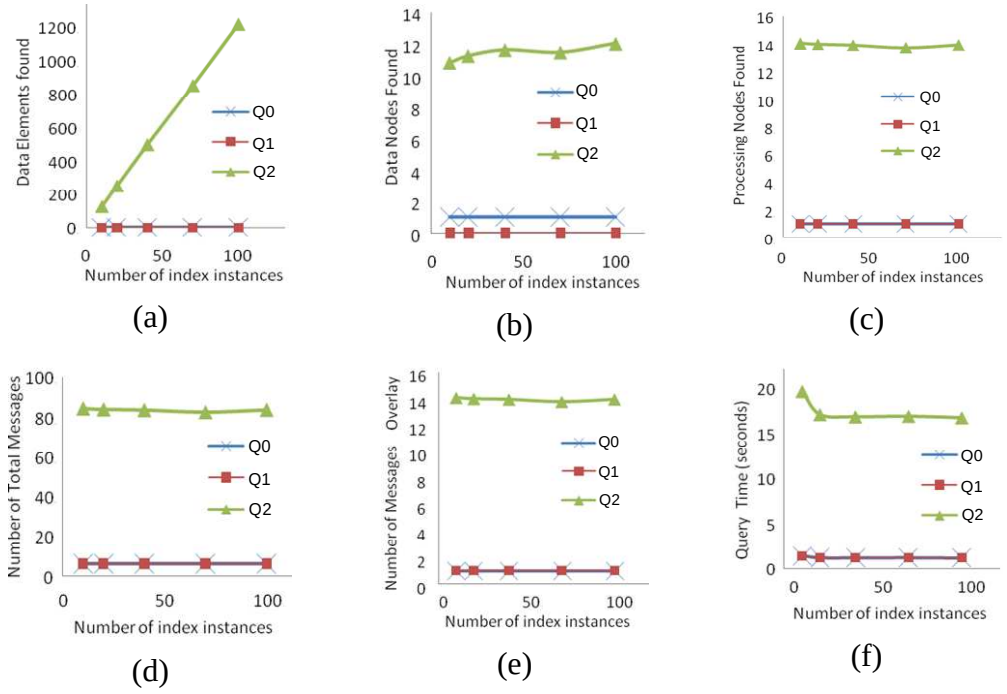


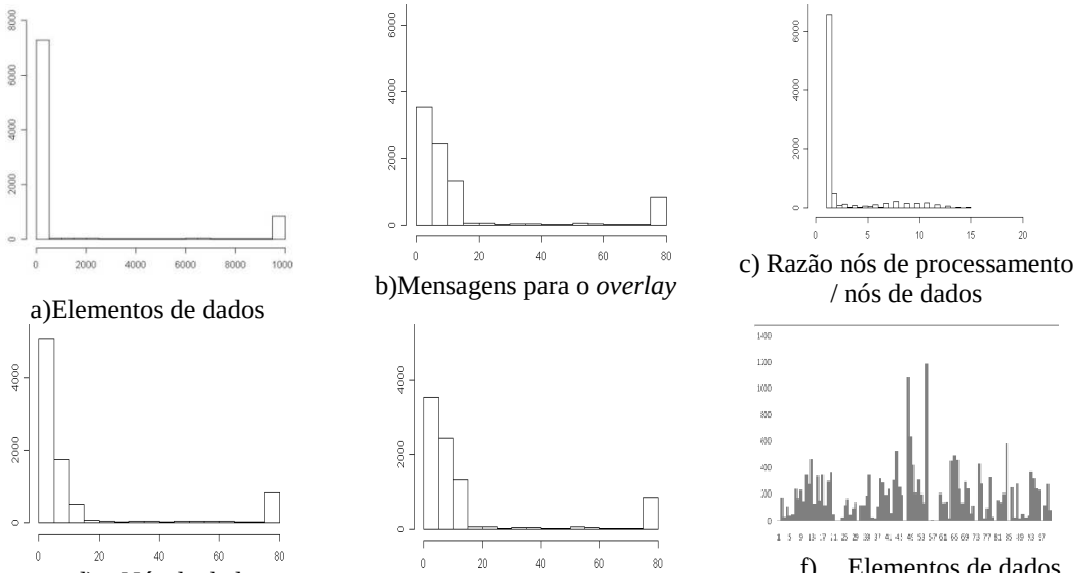
Figura 7. Efeito da variação do número de instâncias por índice

Na Figura 7 (b), observamos um suave aumento no número de nós de dados quando o número de instâncias de índices de cada índice varia de 10 a 40. A métrica permanece quase a mesma quando o aumento é de 40 a 70 e aumenta novamente de 70 a 100. Este resultado é positivo pois mostra que os dados tendem a se espalhar no sistema GD^2 à medida que mais informação existe. A Figura 8(f) ilustra a distribuição dos elementos de dados entre os nós. O eixo *x* representa os nós e *y* indica o número de elementos de dados, ou seja, quase todos os nós armazenam informação.

Na Figura 7(c), observa-se que o número de nós de processamento permanece quase constante quando o número de instâncias de índices aumenta. Isto também é esperado já que o número de nós na rede não é alterado e, portanto, os mesmos nós na rede são responsáveis pelos mesmos intervalos. O número de mensagens trocadas em consultas

do tipo *range* é 80 (oitenta) vezes maior do que as trocadas em consultas exatas ou inválidas - Figura 7(d) – enquanto as mensagens enviadas para o *overlay* é 14 vezes maior - Figura 7(e). Embora este resultado pareça estranho, se observarmos o histograma apresentado em Figura 8 (b), veremos que o número de mensagens enviadas para o *overlay* fica entre 0 e 5 para a maioria das consultas. Isto depende da consulta que é realizada. Por exemplo, uma consulta como "fif*, game*" tende a ser muito mais rápida do que a consulta "f*, g*". O tempo de processamento para consultas do tipo *range* é também maior - Figura 7(f) – mas a mesma explicação dada para o número de mensagens se aplica aqui. Para quase 4000 consultas, o tempo fica em torno de 5 (cinco) segundos para consultas Q2 e 1.3 (um ponto três) segundos para consultas Q0 ou Q1 - Figura 8 (e). Na Figura 8 (a), observa-se que entre 0 e 500 elementos de dados foram encontrados na maioria das consultas. No entanto, para consultas como "a*,b*" foram encontrados mais do que 9500 elementos.

Um fato interessante é a diminuição do tempo da consulta à medida que temos mais informações no GD^2 - Figura 7(f). Isto ocorre pois com uma maior concentração dos dados nos nós, como o algoritmo de *next-match* passa o elemento de dado armazenado no nó como parâmetro para o algoritmo, mais o algoritmo é executado, e portanto maior é a probabilidade de achar um *next-match* zero. Isto também evita uma solicitação adicional por um novo *next-match* na camada *Tier2*, quando o resultado do processamento no nó retorna para o nó solicitante da consulta, o que contribui mais ainda para a diminuição do tempo. Isto pode também ser associado com a diminuição de outras métricas como nós de processamento, número de total de mensagens e de mensagens enviadas ao *overlay*.



a)Elementos de dados

b)Mensagens para o *overlay*

c) Razão nós de processamento / nós de dados

d) Nós de dados

e) Nós de processamento

f) Elementos de dados

Figura 8. Histogramas para 100 instâncias por índice de rede em GD2

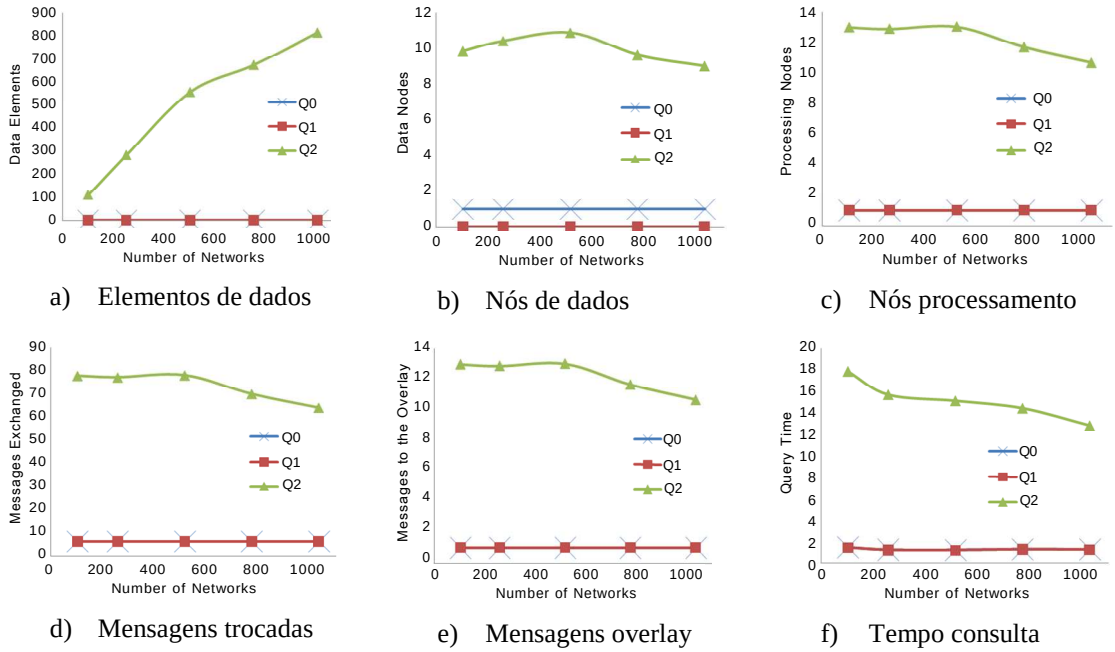
Embora HSFC seja conhecido como um mecanismo que preenche somente uma pequena parte do espaço, formando clusters, no nosso caso os dados ficaram bem distribuídos. Isto ocorre, pois usamos um mecanismo simples de balanceamento de carga, qual seja: como nossos elementos de dados armazenados foram transformados em chaves de 64 bits, este limite (64) foi usado para limitar os identificadores dos nós

DHT. Dessa forma, o intervalo de chaves pelas quais os nós são responsáveis é limitado a chaves de 64 bits, o que ajuda consideravelmente na distribuição natural dos dados.

Também houve a medição da relação entre os nós de processamento e os nós de dados encontrados durante as consultas. Situação perfeita seria se esta divisão fosse sempre 1, o que quer dizer que toda hora em que o algoritmo dentro do nó é executado, elementos de dados são encontrados. Mas isso nem sempre ocorre e muitas vezes é necessário passar por um nó sem ter encontrado elementos pertencentes à consulta. GD^2 apresentou resultados adequados na Figura 8 (c), já que para a maioria das consultas, a razão entre nós de processamento e nós de dados é em torno de 1.

6.3. Efeito da variação do número de redes

Figura 9 apresenta os resultados das métricas selecionadas para 100, 250, 500, 750 e 1000 redes, 10 instâncias de índices para cada rede e 100 nós DHT. Na Figura 9 (a) podemos observar que o número de elementos de dados encontrados por consulta aumenta quando o número de redes aumenta, o que é um resultado esperado. A Figura 9 (b) mostra um aumento estável no número de nós de dados à medida que o número de redes aumenta, o que significa que num primeiro momento os dados tendem a se espalhar na rede. Porém, o número de nós de dados para consultas Q2 tem um pico para 500 redes, enquanto há uma diminuição para 750 e 1000 redes. Isto é relacionado com o conceito de HSFC, que tende a concentrar o dado. Portanto, poucos nós de dados são retornados por consulta quando há mais informação no GD^2.



a) Elementos de dados b) Nós de dados c) Nós processamento

d) Mensagens trocadas e) Mensagens overlay f) Tempo consulta

Figura 9. Efeito da variação do número de redes

Na Figura 9(c), o número de nós de processamento diminui quando aumentamos de 500 para 1000 redes. Isto ocorre pois quanto mais concentrado estão os dados nos nós, mais rápido o *next-match* zero é alcançado e logo, menos nós têm que ser consultados e processados para completar uma consulta particular. As outras métricas na Figura 9(d), (e) e (f) seguem o mesmo comportamento justificado pelo mesmo motivo.

7. Conclusões e Trabalhos Futuros

Neste artigo, é proposto o GD^2, diretório distribuído para lidar com o desafio do gerenciamento de informação em redes colaborativas, dinâmicas, heterogêneas e móveis, em que foi adicionada uma camada para suporte a consultas flexíveis a um ambiente altamente distribuído (a DHT). Constatou-se que há uma excelente sinergia no uso de DHT para distribuição e dinamicidade junto com o princípio das HSFC.

Também foi construído um novo simulador implementando as principais características do GD^2. A análise de desempenho revelou que GD^2 preenche diferentes requisitos de distribuição e escalabilidade, típico para redes futuras. A escalabilidade foi avaliada de acordo com o número de redes e informação. Resultados mostraram que o número de nós de dados e de processamento, o número de mensagens e o tempo de resposta das consultas não apresentaram variações significativas à medida que se tinha mais redes e elementos de dados. Os requisitos traçados para o GD^2 e apresentados na seção 3 foram atendidos e sua validação realizadas através de simulação.

Como trabalhos futuros, pretende-se estender a arquitetura GD^2 e sua avaliação com técnicas mais robustas para balanceamento de carga, incrementá-lo para funcionar com dimensões superiores, oferecer suporte a diferentes tecnologias de redes e incluir mecanismos de segurança relacionados ao controle de acesso para a informação.

Referências

[1] Pentikousis, K., Galis, A., Agüero, R., "Information Management and Sharing for Ambient Multiaccess Networks", IEEE GIIS 2009.

[2] Balakrishnan, H., et al., "Looking Up Data in P2P Systems", Communications of the ACM, Fevereiro 2003

[3] Lawder, J. K., King, P. J. H., "Using Space-Filling Curves for Multi-dimensional Indexing". 17th BNCOD 17, vol 1832, Lecture Notes in Computer Science, pages 20--35, July 2000.

[4] Moon, B. Jagadish, H. V. , Faloutsos, C., Saultz, J. H., "Analysis of the Clustering Properties of the Hilbert Space-Filling Curve", IEEE Transactions on Knowledge and Data Engineering, March 1996.

[5] ITU-T, "Recommendation X.500 - The Directory: Overview of concepts, models and services", v. 08/2005.

[6] Sermersheim, J. (Ed.), "Lightweight Directory Access Protocol (LDAP): The Protocol", RFC 4511, June 2006.

[7] OASIS, "Universal Description Discovery and Integration", OASIS UDDI Version 3.0.2, October 2004, July 2010.

[8] Dustdar, S., Treiber, M. "A View Based Analysis on Web Service Registries", Distributed and Parallel Databases, 18, 147–171, 2005.

[9] Stoica, I. et al., "Chord: a scalable Peer-to-Peer, lookup protocol for Internet applications", IEEE/ACM Transaction on Networking, 2003.

[10] Baumgart, I., Heep, B., Krause, S., "OverSim: A Flexible Overlay Network Simulation Framework", 10th IEEE Global Internet Symposium, May 2007.

[11] Stoica, I. et al., "Chord: a scalable Peer-to-Peer, lookup protocol for Internet applications", IEEE/ACM Transaction on Networking, 2003.

[12] Rowstron, A., Druschel, P. "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems", Middleware, November 2001.

[13] Schmidt, C. e Parashar, M., "A Peer-to-Peer Approach to Web Service Discovery," World Wide Web: Internet and Web Information Systems, 7, 211-229, 2004.