Um SGBD com Armazenamento Distribuído de Dados Baseado em DHT

Eduardo Augusto Ribas[1], **Roney Uba**[1], **Ana Paula Dias**[1], **Arion de Campos Jr.**[1,2], **Davi Arnaut**[1], **Carmem Hara**[1]

[1] Departamento de Informática – Universidade Federal do Paraná
Caixa Postal 19.081 – 81.531-980 – Curitiba – PR

[2]Departamento de Informática – Universidade Estadual de Ponta Grossa
Avenida General Carlos Cavalcanti, 4748 – Ponta Grossa – PR

{ear07, rciu06, apd06, arioncj, davi, carmem}@inf.ufpr.br

Abstract. *This paper investigates the development of a DHT-based storage engine for a database management system (DBMS). The storage engine is responsible for implementing the interface between an SQL query processor and a DHT, by translating operations based on relations to DHT standard operations. By combining DHTs to DBMSs we achieve scalability, decentralization, and fault tolerance, due to a DHT-based relational storage, and also a general high level language for querying data stored on DHTs. Our experimental study shows some initial results on two issues. The first determines the impact of developing an indexing structure on top of a DHT for processing range queries. The second investigates two approaches for mapping relations to DHT's key-value pairs: vertical and horizontal partition of relations.*

Resumo. *Este artigo apresenta o desenvolvimento de um módulo de armazenamento baseado em Tabelas de Espalhamento Distribuídas (DHT) para um Sistema Gerenciador de Banco de Dados (SGBD). O módulo integra a máquina SQL com a DHT, traduzindo para o formato de utilização da DHT as operações de inclusão, leitura, atualização e remoção de dados estruturados em forma de tabelas. Unindo as propriedades da DHT às propriedades do banco de dados, é possível desenvolver um sistema altamente escalável, descentralizado, tolerante a falhas e que seja de fácil utilização para o armazenamento de dados no modelo relacional. São apresentados resultados de dois experimentos. O primeiro tem por objetivo determinar o impacto de uma estrutura de indexação sobre a DHT para o processamento de consultas por intervalo de valores. O segundo investiga duas formas de realizar o mapeamento entre o modelo relacional e o modelo chave-valor, utilizado pelas DHT: a fragmentação horizontal e a fragmentação vertical de tabelas.*

1. Introdução

Redes ponto-a-ponto (P2P) são redes distribuídas não hierárquicas que surgiram com o intuito de facilitar o compartilhamento de recursos entre seus usuários. Uma das propriedades das redes P2P é o seu caráter descentralizado, sem a figura de um responsável central pela gerência e manutenção de sua estrutura. Esta descentralização minimiza a ocorrência de gargalos e filas de espera tão comuns em arquiteturas cliente-servidor. Com

relação ao armazenamento de dados, é frequente a utilização de tabelas de espalhamento distribuídas (*distributed hash tables* – DHT) para gerenciar os dados em uma rede P2P.

As DHT armazenam dados no formato `(chave, valor)` e a interface é constituída pelas operações `get`, que retorna o `valor` associado à `chave`, `put`, que armazena a informação dada, e `rem`, que remove o par que contém uma determinada chave. Desta forma, redes P2P baseadas em DHT são amplamente usadas para o compartilhamento de arquivos e dão suporte a uma funcionalidade de busca simplista norteada pela busca de um arquivo identificado por uma determinada `chave`.

Tendo em vista que em muitas aplicações a recuperação dos dados não se limita a buscas por valores exatos de chave, é desejável a possibilidade de utilizar uma linguagem de alto nível para expressar quais são os dados que se deseja obter. Neste artigo é proposto o desenvolvimento de um sistema de gestão de dados, através da integração de uma máquina de buscas SQL sobre uma base de dados armazenada em uma rede P2P. O objetivo do sistema é combinar características típicas de SGBD, como uma linguagem de consulta de alto nível, com o armazenamento distribuído de dados oferecido pela DHT, que possui propriedades fortemente desejáveis, tais como durabilidade, disponibilidade, desempenho, escalabilidade e descentralização. A integração destes dois componentes é responsabilidade do módulo de armazenamento, que realiza o mapeamento dos dados entre os dois modelos e também sua distribuição. Desta forma, a interface SQL oferecida pelo SGBD é mantida, contendo todas as funcionalidades tradicionais e a distribuição do armazenamento é realizada de forma transparente para o usuário. A idéia central ao se utilizar uma DHT é criar um ambiente distribuído que potencialmente comporta enormes quantidades de dados.

A integração entre SGBD e DHT envolve diversas questões. Uma delas é o suporte a buscas por intervalo, ou seja, a busca de linhas de uma tabela que possuem atributos com valores entre um limite inferior e superior. Estruturas baseadas puramente em DHT não são capazes de processar este tipo de consulta de forma eficiente, uma vez que sua interface padrão é baseada apenas em buscas exatas por chave. A segunda questão diz respeito ao mapeamento entre o modelo chave-valor, utilizado pelas DHT, e o modelo relacional. Neste artigo, são apresentados resultados iniciais de experimentos que abordam estas questões. Assim, os objetivos específicos deste trabalho são:

- proposta de uma arquitetura de integração de SGBD e DHT;
- desenvolvimento e estudo experimental de uma estrutura de indexação sobre a DHT para permitir o processamento de consultas por intervalo;
- desenvolvimento e estudo experimental de diferentes formas de mapeamento entre os modelos de dados chave-valor e relacional.

O restante do trabalho está organizado da seguinte forma. A Seção 2 apresenta trabalhos relacionados, enquanto a proposta da arquitetura e sua implementação são descritas na Seção 3. A análise sobre consultas de intervalo é abordada na Seção 4, e diferentes mapeamentos entre modelos são apresentados e analisados na Seção 5. A Seção 6 finaliza o artigo apresentando alguns trabalhos futuros.

2. Trabalhos Relacionados

Existem atualmente diversos sistemas e arquiteturas que tem como objetivo dar suporte ao armazenamento de um grande volume de dados, tais como os bancos de dados em *cluster*

e os sistemas *noSQL*. Os bancos de dados em *cluster* [Mattoso 2009] foram propostos para prover repositórios relacionais de larga escala. Nestes sistemas, uma camada é desenvolvida sobre um *cluster* de SGBD sendo executados sobre sistemas computacionais convencionais. Esta camada é responsável pelo gerenciamento de meta-dados, replicação e processamento de consultas. A arquitetura proposta neste artigo difere dos bancos de dados em *clusters* por manter a mesma arquitetura em três níveis dos SGBD tradicionais, porém utilizando uma DHT em sua camada física. Enquanto a maioria dos bancos de dados em *cluster* tem como objetivo paralelizar a execução de consultas OLAP, este artigo propõe uma arquitetura voltada para o compartilhamento e distribuição do armazenamento dos dados, cujos acessos são simples e transacionais, com propósitos similares àqueles propostos em [Curino et al. 2010]. Os sistemas *noSQL*, tais como o SimpleDB [Amazon 2007] e Google App Engine [Google 2008] também possuem propósitos similares, além de darem suporte a uma linguagem similar ao SQL. Porém, diversas propriedades inerentes aos SGBD, tais como suporte a metadados e independência lógica e física não são satisfeitas. Por outro lado, este artigo propõe a integração de um SGBD, com todas as facilidades que são tradicionalmente oferecidas, ao armazenamento distribuído em uma rede P2P baseada em DHT.

Uma das dificuldades para fazer tal integração é o suporte a buscas por intervalo, que não são processadas de forma eficiente por uma DHT. Para contornar tal limitação, diversas propostas são encontradas na literatura. Estas soluções enquadram-se, principalmente, em uma das seguintes alternativas: modificar a própria estrutura da DHT e construir uma camada sobreposta a uma DHT existente. Relacionada à primeira solução, pode-se citar o sistema Mercury[Bharambe et al. 2004], que altera a função de espalhamento e inclui algoritmos para o roteamento de mensagens que dá suporte à pesquisa por intervalos e balanceamento de carga. Quanto à segunda alternativa, podem ser mencionados a DST [Zheng et al. 2006] e PHT [Chawathe et al. 2005]. A PHT é uma árvore de busca na qual todo nodo corresponde a um prefixo distinto do domínio da chave. Já a DST implementa uma árvore de segmentos.

A atividade de distribuição de dados compreende duas fases que são cruciais na determinação do desempenho do sistema: fase da fragmentação e fase da distribuição dos fragmentos nos nodos. Como uma DHT distribui os dados baseado no valor de uma função aplicada sobre uma chave, a fase de distribuição de dados envolve a definição de uma chave para a DHT. A fragmentação pode ser feita no nível de tabela, linha ou coluna. Em [Baiao et al. 1998] é enfatizado que o tipo de fragmentação adotado é um importante aspecto a ser considerado no desempenho do sistema e é proposto um algoritmo que auxilia na escolha do tipo de fragmentação a ser empregado.

3. SGBD e DHT: Uma Abordagem de Integração

A integração das tecnologias de SGBD e DHT proposta neste trabalho é baseada no desenvolvimento de um módulo de armazenamento. Um módulo de armazenamento é o componente subjacente do SGBD utilizado para incluir, ler, atualizar e remover itens de uma base de dados. Em um SGBD relacional, este módulo provê o armazenamento, manipulação e recuperação de dados estruturados na forma de tabelas, nas quais cada linha contém um mesmo conjunto de colunas e representa uma tupla. A escolha de uma DHT para o armazenamento dos dados se deve às suas propriedades de durabilidade, disponibilidade, desempenho e descentralização. A idéia central ao se utilizar uma DHT

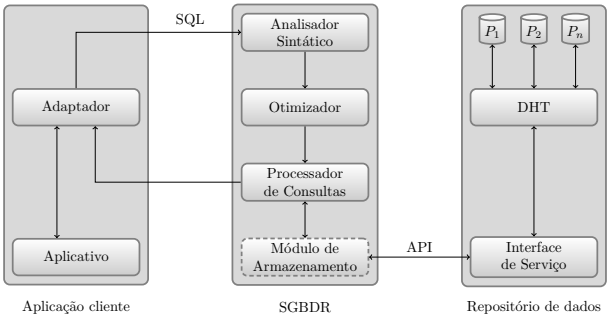


Figura 1. Arquitetura do Sistema

não é paralelizar a execução de uma única transação, mas criar um ambiente distribuído que potencialmente comporta enormes quantidades de dados e que dá suporte a diversas transações (de consulta e inserção) em paralelo.

A Figura 1 apresenta a arquitetura do sistema proposto, ilustrando a função do módulo de armazenamento, como uma interface entre os componentes do SGBD e um repositório de dados baseado em DHT. Ou seja, este módulo é responsável por estruturar o modelo de dados lógico do banco de dados relacional em um modelo físico, que especifica como os dados são armazenados. O módulo recebe requisições do *processador de consultas* do SGBD na forma de uma interface relacional, e transforma estas requisições em comandos para a *interface de serviço* do repositório de dados. Neste caso, esta interface consiste nas funções de manipulação que são padrão em uma DHT: `put(chave, valor)`, `get(chave)` e `rem(chave)`. Uma característica importante da arquitetura proposta é que ela é independente da DHT subjacente, desde que a interface de serviço padrão seja mantida. A recuperação e gerência dos dados armazenados passam a ser problemas tratados pela DHT, que provê um novo ambiente para o processamento de comandos relacionais. É importante notar que, como a arquitetura proposta não prevê alterações no processador de consultas do SGBD, todas as funcionalidades da linguagem SQL permanecem inalteradas.

Para armazenar os dados na DHT é preciso estabelecer estratégias para realizar as quatro operações básicas (inserção, leitura, atualização e remoção) que o módulo de armazenamento de um SGBD deve implementar. Como premissa de projeto, essas operações são implementadas utilizando-se apenas o conjunto de primitivas que a interface da DHT suporta. Para inserir uma linha em uma tabela é preciso empacotar os dados em um par `(chave, valor)`, que é enviado para a DHT através da operação `put`. A forma como os dados em uma tabela são particionados para serem armazenados em pares `(chave, valor)` é denominada de *fragmentação*. Na fragmentação horizontal, o `valor` relacionado a uma determinada `chave` é um vetor unidimensional no qual são empacotados os valores das colunas da linha correspondente. Uma possível `chave` é composta pelo nome do aplicativo e pelo nome completo da tabela. Caso haja índices na tabela, um componente extra pode ser adicionado aos campos anteriormente descritos, concatenando os nomes, lexicograficamente ordenados, das colunas que participam do índice, como por exemplo `/aplicação/banco/tabela/coluna1/coluna2`. No caso da fragmentação vertical, a tabela é fragmentada na forma de colunas. Similarmente à fragmentação horizontal, o `valor` relacionado a uma `chave` corresponde a um vetor unidimensional no qual são empacotados os valores de uma determinada coluna da

tabela. As demais operações sobre tabelas (leitura, remoção e atualização), são transformadas em requisições à DHT de forma similar, gerando-se uma `chave` de busca idêntica àquela utilizada na inserção dos dados.

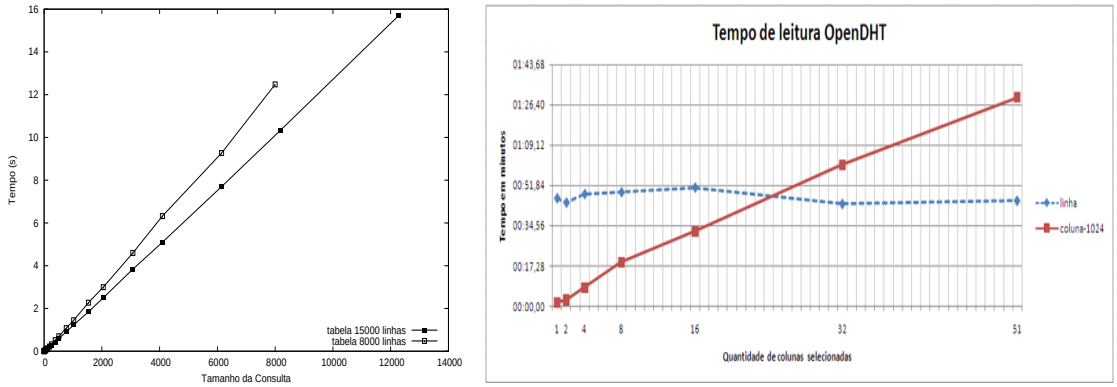
Desenvolvimento
Para o desenvolvimento do sistema proposto foram escolhidos o SGBD MySQL e a OpenDHT [Rhea et al. 2005] como repositório de dados. O *Bamboo* [Rhea 2009] é uma implementação da OpenDHT utilizada no desenvolvimento do sistema proposto. Tal implementação é uma modificação do protocolo Pastry, principalmente na maneira como os vizinhos na DHT são gerenciados, garantindo um melhor comportamento em redes com largura de banda limitada. A DHT pode ser acessada através do protocolo Sun RPC sobre TCP ou XML RPC sobre HTTP, e possui restrições quanto ao tamanho das chaves e valores: chaves são limitadas a 40 bytes e valores em 1024 bytes. O SGBD MySQL possui uma arquitetura na qual os componentes possuem interfaces bem definidas, o que motivou a sua escolha no desenvolvimento do sistema. Esta característica facilita o desenvolvimento de novos componentes, como o módulo de armazenamento. Assim, para cada funcionalidade prevista para este módulo, é definida uma interface que necessariamente o novo componente deve implementar. Exemplificando, para implementar a funcionalidade de inserção em uma tabela, é necessário o desenvolvimento da função `write_row(uchar* buf)`. Essa função recebe a linha que deve ser inserida de uma maneira compactada. Por meio de um conjunto de funções existentes no MySQL é possível desmembrar a linha para obter o valor dos seus atributos. De forma similar, para fazer uma varredura completa, o MySQL utiliza duas funções: `rnd_init()` para iniciar a varredura e `rnd_next(uchar *buf)` para obter as linhas.

Nas próximas seções são descritos experimentos realizados para abordar duas questões pertinentes à integração das tecnologias de SGBD e DHT sobre a arquitetura proposta: o processamento de consultas por intervalo e o mapeamento entre o modelo relacional e o modelo chave-valor.

4. Consulta por Intervalos

Para permitir consultas por intervalos sem a necessidade de recuperar todas as linhas de uma tabela, foi desenvolvida uma estrutura de indexação baseada em DST [Zheng et al. 2006], sobreposta à DHT. A DST foi escolhida tendo em vista o bom desempenho reportado na literatura. Ela é uma árvore de segmentos, na qual cada nó corresponde a um intervalo e os nós terminais equivalem a intervalos atômicos de acordo com uma ordem da esquerda para a direita. Um nó interno u corresponde à união dos intervalos correspondentes às folhas da sub-árvore com raiz em u. Ou seja, cada nó interno é equivalente à união de todos os segmentos dos nós terminais que o descendem.

A DST foi criada sobre a chave primária de cada uma das tabelas. Assim, os valores existentes da chave são divididos em intervalos $[s, t]$, e cada intervalo é atribuído ao nó associado à chave resultante da aplicação da função de espalhamento sobre o valor $[s, t]$. Assim, todos os nós colaborativamente formam uma árvore com relacionamentos pai-filho indicados pela árvore de segmento. As tuplas que pertencem ao intervalo $[s, t]$ são armazenadas na própria estrutura de indexação. Assim, cada nodo da DST tem como referência o seu intervalo de chave e guarda dentro dele tuplas inteiras da tabela.



(a) Consulta por intervalo (b) Fragmentação Vertical e Horizontal

Figura 2. Tempo de consulta dos experimentos

Para o experimento, foi utilizada uma rede local com 10 computadores de configurações semelhantes: processadores *dual* ou *quad core* com memória primária variando de 2GB a 4GB. Os computadores foram interligados em uma rede local de 100MBits, cada um com 5 nodos da DHT, totalizando 50 nodos. Foram utilizadas duas tabelas, r_1 e r_2, com 8.000 e 15.000 tuplas, e 1.1 e 1.5 MB, respectivamente. Para realizar os testes, foram feitas consultas por intervalo nas chaves de cada tabela, levando em consideração intervalos que resultam em uma quantidade fixa de elementos sobre o intervalo. Para medir o desempenho das consultas de maneira precisa, foram geradas 40 consultas diferentes para cada tamanho de intervalo e cada consulta foi executada 2 vezes. Os tempos de execução das consultas são apresentados na Figura 2(a), na qual o eixo vertical representa o tempo em segundos e o eixo horizontal a quantidade de linhas no resultado da busca.

Pode ser observado que o tempo de resposta cresce linearmente com a quantidade de tuplas no intervalo. Para recuperar todas as linhas das tabelas (última marcação nas linhas do gráfico), foram necessários aproximadamente 12 segundos para r_1 e 16 segundos para r_2. Sem a utilização de uma estrutura não clusterizada como a DST em uma DHT, este seria um limite superior do tempo de resposta para qualquer consulta por intervalo, independente do tamanho do resultado, uma vez que seria necessário recuperar todas as tuplas para posteriormente filtrá-las. Como a DST limita o espaço de busca, para recuperar 2000 tuplas foram necessários menos de 3 segundos em ambas as tabelas, o que mostra que a estratégia de indexação sobreposta à DHT melhora significativamente o tempo de resposta deste tipo de consulta. Uma observação importante é que o tempo de resposta de r_1 cresce mais rapidamente que o de r_2 com o tamanho do resultado. Isso se deve ao fato do número de segmentos gerados na DST em ambas as tabelas ter sido o mesmo. Ou seja, cada segmento de r_1 contém menos tuplas que as de r_2. Assim, para recuperar o mesmo número de tuplas no resultado, são necessárias mais requisições à DHT para r_1 que para r_2. Isso mostra que o fator determinante no desempenho das consultas é o número de requisições feitas à DHT.

5. Fragmentação Horizontal e Vertical

Para determinar o impacto de diferentes formas de mapeamento entre o modelo relacional e o modelo `(chave, valor)`, foram desenvolvidos dois tipos de fragmentação de tabelas: horizontal e vertical. Na fragmentação horizontal, cada par `(chave, valor)` armazenado na DHT corresponde a uma única linha da tabela. A `chave` da DHT cor-

responde à concatenação do nome da tabela com o valor da chave primária, enquanto o `valor` contém os valores de todos os seus atributos. Neste tipo de fragmentação, a inserção de uma linha na tabela corresponde à execução de um único comando `put` na DHT. Para a fragmentação vertical, cada fragmento contém um conjunto de valores de uma mesma coluna na tabela. Ou seja, a tabela é dividida por colunas e cada coluna, por sua vez, dividida em blocos. Assim, é gerado um par `(chave, valor)` para cada bloco, sendo que a `chave` corresponde à concatenação do nome da tabela, com o nome da coluna e também um índice de bloco. Se uma coluna é dividida em n blocos, então o índice do bloco está dentro do intervalo $[1..n]$. O `valor` do par `(chave, valor)` corresponde a um conjunto de pares `(chave da tabela, valor do atributo)`. Assim, a inserção de uma linha na tabela resulta na execução de diversos comandos `put` na DHT, um para cada bloco contendo um conjunto de seus atributos.

Foram realizados experimentos com os dois modelos de fragmentação. Eles foram executados em uma rede local com 7 computadores com processadores *dual* ou *quad* core com memória primária variando de 6 a 32 GB. Os computadores estão interligados em uma rede Ethernet de 1 Gbps e em cada computador foram gerados dez nodos para compor a DHT mais um nodo gateway para receber as mensagens enviadas pelo MySQL, totalizando uma rede composta por setenta e um nodos. Os testes foram realizados com uma tabela de 2000 linhas, composta por 51 atributos de 20 bytes cada, totalizando 2,04 MB. Nestes experimentos, foram gerados pares `(chave, valor)` com `valor` de 1024 bytes, que é o maior tamanho permitido pela DHT Bamboo. Assim, na fragmentação horizontal, cada par corresponde a uma linha na tabela, enquanto na fragmentação vertical cada bloco contém 42 valores de uma mesma coluna.

Na fragmentação por linhas, o tempo de inserção de toda a tabela foi de 1 minuto e 39,87 segundos e, na de colunas, o tempo foi de 1 minuto e 59 segundos. Para testar os tempos de busca, foram executadas consultas SQL para obter todas as linhas da tabela, mas variando-se a quantidade de atributos sobre os quais foi realizada uma projeção. Os tempos de execução das consultas são apresentadas na Figura 2(b). O eixo vertical apresenta o tempo da consulta e o eixo horizontal a quantidade de atributos no resultado.

Observa-se que o tempo de consulta utilizando a fragmentação horizontal é praticamente linear, pois em todas as consultas todas as linhas da tabela são retornadas, mesmo que a projeção seja feita sobre um pequeno conjunto de colunas. Já na fragmentação vertical, há um crescimento linear. Para a obtenção de todos as colunas da tabela o tempo gasto utilizando a fragmentação horizontal foi de 45,24 segundos, enquanto que na fragmentação vertical foi de 1 minuto e 29,50 segundos. A diferença no tempo da fragmentação vertical deve-se a três fatores: ao volume maior de dados armazenado na DHT, já que cada coluna é armazenada com a chave primária, à quantidade maior de requisições à DHT e também ao tempo de reconstrução da linha, com a junção de todos os atributos que a compõe. A fragmentação vertical mostrou-se interessante quando a quantidade de atributos retornados é pequeno. Neste caso, o tempo de busca para as duas formas de fragmentação é exatamente na metade da quantidade total de atributos.

Foram realizados também experimentos com a fragmentação vertical, na qual os blocos continham uma quantidade menor de valores. No caso extremo, cada bloco consiste apenas de um par `(chave primária, valor do atributo)`. Neste caso, o tempo de execução das consultas foi extremamente alto, chegando a 1 minuto e 17,91

segundos para consultas retornando apenas 1 atributo e aproximadamente uma hora para obter todos os atributos. Estes resultados confirmam os resultados do experimento descrito na Seção 4 de que a quantidade de requisições à DHT é um fator determinante para o desempenho do sistema.

6. Conclusão

Neste trabalho é descrito um sistema de gestão de dados para indexação e recuperação de dados relacionais armazenados em uma DHT. A organização e arquitetura do sistema seguem uma abordagem estratificada cujo alicerce é a DHT. A DHT é responsável por distribuir e armazenar os dados entre os vários nós da rede, enquanto o módulo de armazenamento é responsável por realizar o mapeamento entre os modelos relacional e pares `(chave, valor)`. A camada de indexação, justaposta à DHT, tem por objetivo aprimorar o tempo de acesso às informações em consultas por intervalo. Como contribuições deste trabalho podem ser destacados o estudo de viabilidade da arquitetura proposta e a facilidade para a integração das tecnologias de banco de dados e DHT. Os experimentos revelaram que o desempenho do sistema é fortemente relacionado ao desempenho da DHT subjacente. Assim, o projeto físico do SGBD, que inclui a criação de índices, bem como a definição de formas de clusterização e fragmentação que gerem um volume menor de requisições à DHT são pontos a serem investigados no futuro. Além disso, experimentos adicionais são necessários para avaliar o impacto de outros fatores sobre o desempenho do sistema, além da sua viabilidade e escalabilidade em um ambiente maciçamente distribuído. Pode-se concluir que com a combinação das consultas sofisticadas dos SGBD com a escalabilidade e robustez de uma DHT, é aberta uma ampla área para novas pesquisas, dentre as quais podem ser citadas o suporte a transações e otimização de consultas neste tipo de arquitetura.

Referências

Amazon (2007). Amazon SimpleDB. `http://aws.amazon.com/simpledb/`.

Baiao, F., Mattoso, M., and Zaverucha, G. (1998). Towards an inductive design of distributed object oriented databases. In *CoopIS*.

Bharambe, A. R., Agrawal, M., and Seshan, S. (2004). Mercury: Supporting scalable multi-attribute range queries. In *SIGCOMM*.

Chawathe, Y., Ramabhadran, S., Ratnasamy, S., Lamarca, A., Shenker, S., and Hellerstein, J. (2005). A case study in building layered dht applications. In *SIGCOMM*.

Curino, C., Jones, E., Zhang, Y., Wu, E., and Madden, S. (2010). Relational cloud: The case for a database service. Technical Report MIT-CSAIL-TR-2010-014, MIT.

Google (2008). App engine datastore.

Mattoso, M. (2009). Database clusters. Encyclopedia of Database Systems 2009.

Rhea, S. (2009). The bamboo distributed hash table. Disponível em `http://bamboo-dht.org/`.

Rhea, S., Godfrey, B., Karp, B., Kubiatowicz, J., Ratnasamy, S., Shenker, S., Stoica, I., and Yu, H. (2005). Opendht: A public dht service and its uses. In *ACM SIGCOMM*.

Zheng, C., Shen, G., Li, S., and Shenker, S. (2006). Distributed segment tree: Support of range query and cover query over dht. In *Int. Workshop on P2P Systems*.