NodeWiz-R: um sistema para a indexação expressiva, eficiente e auto-gerenciável de recursos distribuídos na Internet

José F. M. Vieira Júnior, Paulo R. M. Gomes, Francisco Brasileiro, Lívia Sampaio

[1]Universidade Federal de Campina Grande
Departamento de Sistemas e Computação
Laboratório de Sistemas Distribuídos
58.109-970, Campina Grande, PB, Brasil

{zflavio, paulo}@lsd.ufcg.edu.br, {fubica, livia}@dsc.ufcg.edu.br

Abstract. *Resource discovery is a crucial service to enable sharing in large scale systems such as the Internet. In such broad and dynamic systems, distribution and self-management are key requirements. Although several solutions have been proposed for distributed and autonomous resource discovery services on the Internet, they all fail to provide at least one of the following important features: i) an expressive way to relate the attributes that annotate resources; ii) efficient ways to index these attributes; and iii) high matching rates in the requests processed. In this paper, we present NodeWiz-R, a distributed and self-managed resource discovery solution that implements a relational model atop of a peer-to-peer substrate to simultaneously provide all these three features. Results obtained by simulations and confirmed by experiments with the system prototype, indicate that NodeWiz-R is more efficient than other flooding based solutions for distributed SQL query processing.*

Resumo. *Um pré-requisito para o compartilhamento de recursos é a existência de um serviço que habilite os usuários a descobrirem os recursos que estejam disponíveis no sistema. Em um sistema com a amplitude e a dinamicidade da Internet, é necessário que o serviço de descoberta seja distribuído e auto-gerenciável. Embora várias soluções para a implementação de serviços de descoberta distribuídos e autônomos tenham sido sugeridas, nenhuma delas consegue ser ao mesmo tempo expressiva, eficiente e ter alta cobertura. Nesse artigo nós propomos NodeWiz-R, uma solução para a descoberta de recursos distribuída e autônoma que implementa uma camada relacional sobre um substrato peer-to-peer para prover simultaneamente todas essas propriedades. Os resultados obtidos através de simulações e experimentos com um protótipo mostram que o NodeWiz-R é mais eficiente que outras soluções que utilizam técnicas de flooding para o processamento de consultas expressas no padrão SQL, apresentando uma alta taxa de retorno de resultados com uma baixa sobrecarga.*

1. Introdução

A popularização da Internet tem propiciado o compartilhamento em larga escala de recursos e conteúdo cujo interesse é comum entre usuários. Esse compartilhamento abrange desde a infra-estrutura de comunicação, passando pelos recursos computacionais que estão em computadores pessoais e servidores, até o grande volume de dados que está armazenado de forma distribuída nesses recursos.

Em ambientes de compartilhamento em larga escala, um serviço básico é aquele que possibilita a descoberta dos recursos (*Resource Discovery* [Harchol-Balter et al. 1999]). O serviço de descoberta, a partir de uma descrição fornecida, retorna um conjunto de provedores que detêm recursos que casam com essa descrição. O serviço implementa a abstração de um catálogo que armazena informações sobre os atributos de todos os recursos disponíveis, usando os valores desses atributos para gerar índices e tornar as consultas mais eficientes.

Uma característica importante de muitos sistemas de compartilhamento de recursos existentes é a autonomia dos participantes. Em *sistemas abertos* de compartilhamento de recursos, os provedores podem entrar e sair do sistema sem que exista um prévio anúncio e sem a existência de uma coordenação central. Essa característica tem duas implicações principais para a manutenção do catálogo: i) as informações sobre os recursos disponíveis precisam ser constantemente revalidadas; e ii) a responsabilidade de manutenção do catálogo deve ser compartilhada entre os provedores que estão efetivamente fazendo parte do sistema em um determinado instante de tempo. Dessa forma, é recomendável que o serviço de descoberta seja distribuído e autogerenciável, de modo a diminuir o esforço administrativo necessário para manter o serviço de descoberta operacional e consistente e evitar que falhas não recuperáveis do mantenedor do catálogo levem o sistema inteiro ao colapso.

Dois exemplos concretos de sistemas abertos que necessitam de serviços de descoberta distribuídos e autogerenciáveis são o SegHidro [Araújo et al. 2005] e o OurGrid [Cirne et al. 2006]. O SegHidro permite o compartilhamento de dados ambientais entre pesquisadores espalhados pelo mundo. Já o OurGrid implementa uma infraestrutura de grade computacional para compartilhar a capacidade de processamento dos computadores ociosos distribuídos pela Internet.

Sistemas de descoberta de recursos para sistemas abertos baseados em arquiteturas *peer-to-peer* (P2P) [Harchol-Balter et al. 1999, Iamnitchi et al. 2002, Basu et al. 2009, Oppenheimer et al. 2005] têm sido propostos, aproveitando as características de descentralização, escalabilidade, autogerencimento e baixo custo dos sistemas P2P. Alguns desses sistemas são extremamente eficientes para realizar buscas em índices organizados como uma única tabela que associa os atributos de um recurso a seus respectivos valores. Porém, com o aumento da complexidade dos atributos que descrevem os recursos e dos requisitos das aplicações que desejam localizar esses recursos, surgiu a necessidade de serviços de descoberta mais expressivos, capazes de lidar com recursos que possuem um modelo de representação mais complexo. Por exemplo, no caso do provedor de dados ambientais, para cada conjunto de dados (*dataset*) o catálogo armazena informações sobre os atributos de localização geográfica e temporal, variáveis de interesse (e.g. umidade e temperatura), além de rótulos (*tags*) que podem ser associados tanto ao conjunto de dados como um todo quanto a cada variável nele contido. Assim, tais recursos não podem ser representados eficientemente por um modelo de representação simplificado, como os baseados em pares de atributo-valor.

Uma possível solução para aumentar a expressividade do serviço de descoberta é a utilização de um modelo relacional para representar os recursos e uma base de dados relacional para implementar o catálogo. Os sistemas de bancos de dados distribuídos (SBDD), em particular, têm se destacado por sua arquitetura descentralizada que possibilita uma melhor distribuição da carga de trabalho, espalhando dados pela rede de forma contro-

lada. Entretanto, tais sistemas não lidam de forma autônoma com a intermitência dos nós que armazenam as informações, podendo causar indisponibilidade e inconsistência dos dados. Além disso, podem demandar esforços administrativos para reconfiguração do sistema, gerando um alto custo de manutenção e requerendo uma centralização da operação do sistema. Finalmente, os sistemas de banco de dados (SBD) P2P têm surgido como alternativa para ligar os dois mundos discutidos acima, oferecendo suporte à modelagem de dados expressiva e armazenamento eficiente, além de garantirem transparência e autogerenciamento [Ng et al. 2003, Boncz and Treijtel 2004, Huebsch et al. 2005].

Neste artigo, nós apresentamos o NodeWiz-R, uma solução para descoberta de recursos de forma distribuída e autônoma que tem as seguintes características: i) modelo expressivo para definição do catálogo; ii) autogerenciamento; iii) indexação eficiente tanto em termos de armazenamento como em termos de latência para a resolução de consultas; e iv) alta taxa de cobertura.

O restante do artigo está estruturado da seguinte forma. Na Seção 2 nós apresentamos o estado-da-arte, sumariando as principais soluções para a descoberta de recursos em sistemas abertos propostas na literatura e discutindo suas limitações. O NodeWiz-R é apresentado na Seção 3, enquanto sua avaliação quantitativa a partir de um modelo simulado do sistema é apresentada na Seção 4. Por fim, na Seção 5 nós apresentamos as principais conclusões e as direções para trabalhos futuros.

2. Trabalhos Relacionados

Grande parte dos sistemas para a descoberta de recursos propostos nos últimos anos, são baseados em arquiteturas P2P apoiadas sobre DHTs. Um exemplo é o PIER [Huebsch et al. 2005], um processador de consultas relacional que permite a localização de recursos na sua rede utilizando vários tipos de consultas, como *exact-match queries*, naturalmente provido pela DHT, e ainda consultas com operações relacionais (como junções). As informações sobre os recursos são distribuídas pela rede usando a implementação de DHT Bamboo [Rhea et al. 2004]. Consultas por múltiplos atributos são possíveis através da sobreposição de várias DHTs, uma para cada atributo. Entretanto, uma operação de atualização pode comprometer a escalabilidade do sistema, quando múltiplos atributos precisam ser atualizados. Além disso, PIER faz *broadcast* para conseguir realizar alguns tipos de consultas mais elaboradas, como as que envolvem operações de junção. O uso de *broadcast* implica na geração de tráfego desnecessário na rede, além de não garantir que todos os nós sejam alcançados, o que não assegura que todas as informações disponíveis na rede sejam retornadas quando requisitadas. PIER também adota uma linguagem de consulta pouco conhecida e pouco amigável (*Unnamed Flow Language - UFL* [Huebsch et al. 2005]), o que dificulta sua utilização.

Existem ainda soluções baseadas em DHTs para prover um armazenamento eficiente de informações sobre esquemas RDF (*Resource Description Framework*), bem como o roteamento de consultas expressivas sobre eles [Sidirourgos et al. 2005, Heine et al. 2005, Liarou et al. 2007]. RDF armazena informações em conjuntos de triplas, compostas por assunto (*subject*), predicado (*predicate*) e um objeto (*object*). A semântica das triplas é definida pelo modelo teórico e por regras, que definem como uma tripla pode ser gerada. RDF permite que informações se relacionem, através de um conceito de classes e entidades. A proposta desses sistemas é que cada tripla seja indexada pela DHT, atribuindo uma chave para cada componente da tripla. Dessa forma, é possível consultar uma tripla

por assunto, predicado ou objeto. O maior problema desse tipo de sistema é o *overhead* de mensagens gerado pelo sistema para publicar as informações, já que uma mesma tripla é enviada para três nós distintos ao mesmo tempo. Além disso, operações de consulta também geram uma carga extra indesejada, uma vez que aquelas formadas por mais de uma tripla são avaliadas fazendo várias combinações com todas as triplas, fazendo com que várias chaves de busca na DHT sejam geradas, uma para cada combinação possível. Por isso, uma operação de consulta pode requerer que uma elevada quantidade de mensagens sejam trocadas pelo sistema.

O uso de DHTs é motivado pela sua eficiência para processamento e roteamento de operações, além de serem escaláveis, mesmo para redes altamente dinâmicas. Por outro lado, como observado nos trabalhos citados acima, estas não permitem que consultas mais elaboradas, incluindo múltiplos atributos e faixas de valores (*range queries*), sejam executadas de forma natural. Além disso, uma DHT não preserva a proximidade semântica dos dados, uma vez que eles são fragmentados pela rede de acordo com as chaves que lhes são atribuídas. Uma alternativa às DHTs, sem perder a eficiência e escalabilidade provida por algumas soluções P2P, é o uso de SBD P2P. Nesse caso, as vantagens da abordagem P2P são aliadas com a expressividade provida pelos modelos de dados suportados pelas tecnologias de bancos de dados.Em outras palavras, os SBDs P2P estendem os sistemas P2P puros, oferecendo suporte a uma modelagem de dados mais expressiva e armazenamento eficiente, além de garantirem transparência nas operações e autogerenciamento, propiciado pelo substrato P2P sobre o qual se apoiam.

PeerDB [Ng et al. 2003] utiliza um SBD relacional sobre uma arquitetura P2P não estruturada, possibilitando o compartilhamento de informações que estão armazenadas nos bancos de dados locais de cada nó. Ele fornece suporte a consultas no padrão SQL (*Structured Query Language*) que são executadas no banco de dados local de cada nó. Porém, por utilizar uma arquitetura P2P não estruturada, PeerDB não garante o retorno de todas as informações disponíveis que casam com determinada consulta. Além disso, o sistema não lida bem com a presença de *churn*, consequentemente, os dados de um nó que deixe a rede espontaneamente ou mediante uma falha, podem tornar-se indisponíveis por tempo indeterminado.

AmbientDB [Boncz and Treijtel 2004] é uma arquitetura para processamento de consultas complexas em redes P2P, que permite a execução de consultas relacionais no padrão SQL. Além disso, possui um esquema de dados bem definido, que implementa conceitos de tabelas locais, tabelas distribuídas e tabelas particionadas. Assim como em PIER, AmbientDB usa várias DHTs para indexar as informações na rede P2P e garantir a escalabilidade. Porém, não possui um mecanismo eficiente para roteamento de consultas complexas. AmbientDB também faz *broadcast* para executar algumas operações, como junções distribuídas, e tais operações exigem altas taxas de transferência de dados para serem completadas. Por não dispor de nenhum mecanismo para balancear a carga dos nós do sistema, os nós podem ainda descartar algumas operações de *INSERT* quando a capacidade máxima do índice local é atingida. Isso cria um índice incompleto, reduzindo o número de informações que podem ser encontrados no sistema.

Outras formas de estruturar sistemas P2P foram propostas para habilitá-los a executar consultas por faixa e por múltiplos atributos eficientemente, a exemplo do NodeWiz (NW) [Basu et al. 2009], um sistema P2P para descoberta de recursos na Internet. No NW é usada uma estratégia para indexação dos dados baseada em árvores k-dimensionais

que são capazes de rotear eficientemente consultas complexas que envolvam múltiplos atributos e faixas de valores. Além disso, o NW oferece uma estratégia inovadora para ajustar a divisão da carga de trabalho entre os nós de acordo com as mudanças ocorridas (chegada de novos dados e consultas recebidas). Isto permite que consultas sejam executadas eficientemente, mantendo uma carga uniforme entre os nós sob uma carga de consultas variante. Contudo, os sistemas de descoberta baseados nas arquiteturas P2P estruturadas em geral não permitem que os atributos associados a uma chave possam ter relacionamentos mais sofisticados, restringindo a expressividade das consultas que podem ser feitas além de impossibilitar que recursos com um modelo de descrição mais complexo possam ser representados pelo sistema.

3. O NodeWiz-R

Um serviço de descoberta de recursos para um sistema aberto deve satisfazer as seguintes características: (i) o sistema precisa ser autônomo, sem a existência de um controle centralizado; (ii) deve ser distribuído, para evitar a existência de um ponto único de falhas e de sobrecarga administrativa nesse ponto; (iii) deve implementar um modelo de dados expressivo, de modo a permitir relacionamentos entre as informações armazenadas e, consequentemente, consultas mais elaboradas; (iv) deve implementar um mecanismo de roteamento que permita, a um baixo custo, alcançar uma alta taxa de cobertura das consultas; e (v) deve implementar um modo de armazenamento que facilite as operações de manutenção da base de dados, e lidar bem com a intermitência e dinamicidade dos provedores e dos recursos.

NodeWiz-R (NW-R) é um sistema P2P relacional com todas essas características, possibilitando a descoberta de recursos distribuídos e que possuem um modelo de representação complexo. Ele é uma extensão do sistema de informação de grade NW [Basu et al. 2009]. Portanto, alguns dos requisitos listados acima são atendidos pelo NW – particularmente os requisitos (i) e (ii), enquanto que outros são atendidos pelas funcionalidades introduzidas pelo NW-R. O NW-R segue a mesma estruturação dos SBDs P2P, com uma camada relacional que atende ao requisito (iii), permitindo que recursos complexos sejam descritos de forma natural, através da modelagem relacional, além de possibilitar que consultas ricas sejam resolvidas. Utilizando uma camada relacional, o NW-R também possibilita que consultas no padrão SQL sejam realizadas. Para atender ao requisito (iv), o NW-R implementa três níveis de indexação que permitem localizar eficientemente até mesmo informações com cópias espalhadas por diversos nós da rede (mais informações na Seção 3.3). Finalmente, para manter a consistência dos dados armazenados no catálogo e satisfazer o requisito (v), utiliza-se a abordagem de atualização de estado fraca (*soft-state*), causando o desaparecimento dos dados do catálogo depois de um período de tempo pré-estabelecido; cabe aos provedores de recursos atualizarem periodicamente o catálogo com as informações mais atuais referentes aos seus recursos.

3.1. Arquitetura

Um sistema NW-R é composto por um conjunto de nós autônomos que conjuntamente provêem as informações sobre os recursos que estão sendo compartilhados. Como pode ser observado na Figura 1, cada nó mantém um banco de dados relacional embarcado para armazenar a parte das informações do catálogo que está sob a sua responsabilidade. O banco de dados permite que as informações sobre os recursos possam ser modeladas usando o modelo relacional, além de possibilitar uma recuperação eficiente dessas informações localmente.

Por ser baseada em uma arquitetura P2P estruturada, cada nó também mantém informações sobre um subconjunto de nós da rede que são responsáveis por outras partes das informações do catálogo. Dessa forma, operações de atualização e de consulta podem ser roteadas de forma eficiente para os nós pertinentes através da camada P2P. Cada nó do NW-R poderá ter várias tabelas de roteamento, também chamadas de índices, para rotear consultas referentes a diferentes tabelas definidas no modelo relacional. Três tipos de índices distribuídos são utilizados: **índice primário**, **índice secundário** e **tabela-índice**.

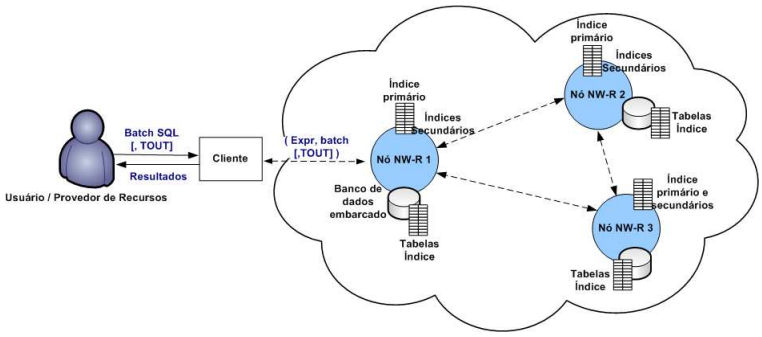


Figura 1. Arquitetura do NW-R em alto nível

Uma aplicação que usa o NW-R para localizar recursos compartilhados em um sistema aberto precisa definir e tornar público o esquema do catálogo usado para armazenar os atributos dos recursos que serão compartilhados. Esse esquema é definido através de um modelo relacional. Além disso, devem ser especificados no esquema, a *tabela principal* e as *tabelas secundárias* que se deseja indexar. A tabela principal normalmente mantém os principais atributos que descrevem um recurso e precisa ser definida em qualquer esquema. Já as tabelas secundárias mantêm informações adicionais sobre os recursos e estão direta ou indiretamente relacionadas com a tabela principal. As tabelas secundárias só devem ser definidas caso se deseje indexar e otimizar as operações sobre as informações mantidas nestas tabelas. Se os atributos de alguma tabela secundária não forem utilizados de forma exclusiva em consultas, não é necessário indexar tal tabela. Tabelas de ligação, por exemplo, não precisam ser indexadas, já que não é comum a realização de consultas utilizando apenas campos dessas tabelas.

O esquema simplificado da aplicação de compartilhamento de dados ambientais do projeto SegHidro é apresentado na Figura 2. A tabela $RESOURCE$ possui os principais atributos que descrevem o recurso e, portanto, é considerada a tabela principal do esquema. A tabela TAG contém rótulos que também auxiliam na identificação de um recurso. Por isso, ela é considerada uma tabela secundária e precisa ser indexada, pois os usuários podem gerar consultas contendo apenas campos dessa tabela e a mesma precisa ser roteada eficientemente pela rede P2P. A tabela $RESOURCE\_TAG$ é considerada apenas uma tabela de ligação e não precisa ser indexada, por isso não é listada no esquema como tabela secundária. A tabela principal tem uma função importante dentro do sistema, que é servir como base para o *particionamento horizontal* das tabelas entre os nós, como será explicado adiante.

O NW-R utiliza a abordagem *soft-state* para facilitar a manutenção das informações no sistema. Nessa abordagem, uma estampilha de tempo ($TOUT$) é associada a cada registro (linha da tabela) armazenado nas tabelas do banco de dados. O valor de $TOUT$ determina por quanto tempo o registro deve permanecer no sistema. Quando esse tempo

```
SCHEMA DEFINITION SEGHIDRO (
   CREATE TABLE RESOURCE (
       URL VARCHAR2 NOT NULL,
       Description VARCHAR2,
       Latitude Double Precision,
       Longitude Double Precision,
       Initial_Date Timestamp,
       Final_Date Timestamp,
      CONSTRAINT c1 PRIMARY KEY (URL)   );

   CREATE TABLE TAG  (
      ID NUMERIC,
      TAG_NAME VARCHAR2,
      CONSTRAINT c2 PRIMARY KEY (ID)   );

   CREATE TABLE RESOURCE_TAG  (
      RESOURCE_ID VARCHAR2,
      TAG_ID NUMERIC,
      CONSTRAINT c3 FOREIGN KEY (RESOURCE_ID) REFERENCES RESOURCE(URL),
      CONSTRAINT c4 FOREIGN KEY (TAG_ID) REFERENCES TAG (ID)   );

   PRIMARY TABLE RESOURCE,
   SECONDARY TABLE TAGS
);
```

Figura 2. Esquema da aplicação de compartilhamento de dados ambiental do projeto SegHidro

expira, o registro é automaticamente removido. Assim, as informações sobre os recursos devem ser publicadas periodicamente no sistema pelos provedores, mantendo-as atualizadas com relação ao estado atual do recurso. Essa abordagem se mostra adequada para os casos em que as informações sobre os recursos são dinâmicas e intermitentes. A periodicidade com que as informações são publicadas deve ser determinada por cada provedor de recurso, levando em consideração fatores como o custo associado com a publicação, o tempo de atualização das informações localmente e até mesmo o *churn* na rede. Quanto maior for a intermitência dos nós, mais frequente deve ser a atualização, de forma a garantir que as informações sobre os recursos permaneçam disponíveis na rede.

3.2. Executando operações no NW-R

Uma operação que é enviada para o NW-R possui o seguinte formato: $OP ::= (type, batch[, TOUT])$, onde: *type* define o tipo de operação (atualização ou consulta); *batch* é um conjunto de instruções SQL de atualização ou consulta; e *TOUT* é o tempo que as informações deverão ser mantidas no catálogo (apenas para operações de atualização).

As operações, sejam elas de atualização ou consulta, são enviadas juntamente com um *batch* SQL. Para ambos os tipos de operação, uma expressão é gerada baseada no *batch* SQL fornecido. Essa expressão é formada por um conjunto de restrições do tipo "<atributo> <operador lógico> <valor>", e é utilizada pela camada P2P para rotear a operação para os nós que são responsáveis por manter as partes do catálogo que casam com a expressão. O nó que recebe a operação, verifica se ele é responsável por executá-la e se ele conhece outros nós que também precisam executar a operação. Caso existam, a operação é roteada para os nós pertinentes e o *batch* SQL é executado apenas nesses nós. Caso a operação seja de consulta, os resultados são retornados diretamente para o cliente.

Vale salientar que uma das principais preocupações dos processadores de consultas distribuídos está em prover suporte eficiente a operações de junção e agregação, geralmente necessárias no processamento distribuído de cláusulas SQL [Kossmann 2000]. No NW-R, não se faz necessário lidar com junções distribuídas. Isso é possível pois todas as informações que estão relacionadas de alguma forma, estão presentes no mesmo nó, ou seja, dados comuns a diferentes recursos são copiados para os nós onde as informações serão armazenadas. O custo associado ao uso dessa solução está principalmente na replicação das informações e manutenção dos índices para identificar onde elas estão dispostas na rede. Dessa forma, elimina-se o custo bastante elevado do processamento

de junções distribuídas. Quanto às operações de agregação, estas também são executadas apenas localmente. Os resultados de cada operação local são retornados para o cliente, que se responsabiliza por filtrar/agregar os resultados recebidos dos diversos nós[1].

3.3. Camada P2P

A camada P2P é implementada através de uma extensão do sistema NW [Basu et al. 2009]. Este sistema organiza as informações publicadas em um espaço de atributos multidimensional, onde cada nó é responsável por um subespaço do espaço total de atributos do sistema. Para distribuir esses dados pelos nós da rede, o NW implementa uma árvore k-dimensional que determina qual nó é responsável por determinado subespaço. Como uma árvore k-dimensional é basicamente uma árvore binária de pesquisa, faixas de valores podem ser encontradas eficientemente. Por exemplo, o número de saltos necessários para que uma mensagem alcance o nó destino e o tamanho da tabela de roteamento são uma função logarítmica do número de nós do sistema. Além disso, o balanceamento da carga de trabalho ocorre naturalmente, pois as consultas são processadas apenas nos nós responsáveis pelos valores de atributos requisitados.

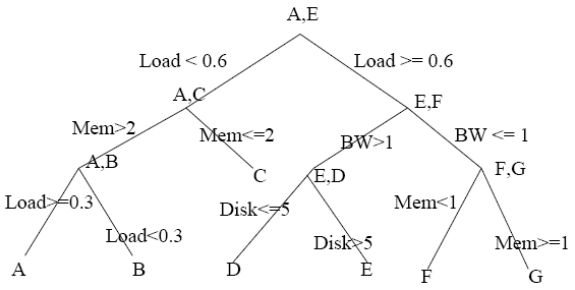


Figura 3. Divisão do espaço de atributos [Basu et al. 2009]

Level	Attr	Min	Max	IP Addr
0	Load	0.6	+inf	E
1	Mem	0	2	C
2	Load	0	0.3	B

Figura 4. Exemplo de uma tabela de rotas [Basu et al. 2009]

A Figura 3 mostra como um espaço de atributos, representando os computadores de uma grade computacional, pode ser dividido entre os nós que implementam o serviço de descoberta. As folhas da árvore representam os nós da rede NW, enquanto as bifurcações na árvore indicam como o espaço de atributos foi dividido. Cada nó NW mantém uma tabela de roteamento (Figura 4) que armazena informações parciais sobre a distribuição do espaço de atributos entre os nós do sistema. A tabela de roteamento permite que um nó roteie operações para outros nós de forma seletiva, de acordo com o subespaço de atributos mantido por cada nó da rede.

Quando um único nó participa da rede NW o sistema atua como um índice centralizado, onde todas as consultas e dados são processadas e armazenados em um só lugar. Quando outro nó junta-se à rede NW, ele busca o nó mais sobrecarregado e este divide seu espaço de atributo com o novo nó. O novo nó recebe, além da parte dos dados, a tabela de rotas do outro nó. Em seguida, ambos adicionam uma entrada no final de suas tabelas de rotas apontando um para o outro. A Figura 4 mostra a tabela de rotas do nó A da Figura 3. Por exemplo, a entrada do nível 0 indica que o nó E sabe quais outros nós guardam a informação sobre que provedores têm recursos com atributo $Load$ maior que 0.6.

No NW-R a divisão do espaço de atributos é feita de forma similar ao NW com as devidas adaptações para lidar com o modelo relacional. Os atributos usados para realizar

[1]É importante perceber que o NW-R não é uma proposta de banco de dados relacional P2P ou um substituto de qualquer sistema de banco de dados, não podendo ser portanto utilizado para fins de armazenamento persistente de dados.

o particionamento (ou fragmentação) horizontal das tabelas do esquema são os campos da tabela principal. As informações contidas na tabela principal são exclusivas, já que se referem a recursos distintos. Dessa forma, quando escolhemos um campo da tabela principal para compor o predicado que será usado como base do particionamento, garantimos que as informações sobre um determinado recurso estarão disponíveis em apenas um nó da rede. Os dados armazenados nas tabelas secundárias, por outro lado, podem estar relacionados com dados de outras tabelas, e por isso podem aparecer replicados quando os dados de um recurso migram para outro nó (depois de uma divisão do espaço de atributos). Isso ocorre pois as informações das tabelas secundárias que são relacionadas com uma tupla da tabela principal, são copiadas juntamente com a tupla que está sendo movida para o outro nó. Como discutido anteriormente, essa decisão possibilita que o NW-R processe localmente consultas que requerem junções, uma vez que todos os dados relacionados são mantidos no mesmo nó.

O índice primário e os índices secundários do NW-R possuem a mesma estrutura de uma tabela de roteamento do NW. O índice primário mantém informações sobre que faixa de valores, e de quais atributos da tabela principal do esquema, outros nós da rede são responsáveis. Isto ocorre porque cada nó armazena informações sobre recursos diferentes, cujas principais informações estão armazenadas na tabela principal. Como a maior parte das operações se refere a esta tabela, o índice primário será o mais utilizado para rotear as operações.

Os índices secundários mantêm informações para o roteamento das consultas sobre as tabelas secundárias definidas no esquema. Assim como ocorre com o índice primário, um índice secundário armazena informações parciais sobre qual espaço de atributos, da tabela secundária a qual o índice se refere, um determinado nó é responsável. Entretanto, o nó para onde a operação será roteada não armazena propriamente todas informações da tabela secundária em questão, uma vez que suas informações podem ser copiadas para diversos nós da rede. Portanto, para que essas informações possam ser localizadas, é necessário adicionar um nível de indireção ao sistema, representado pela tabela-índice.

A tabela-índice permite a indexação de informações da tabela secundária que estão presentes em vários nós da rede; o valor de um atributo pode apontar para diferentes nós da rede, caso a informação da tabela secundária requisitada esteja disposta em vários nós. Cada tabela-índice armazena apenas uma faixa de valores dos atributos da tabela secundária a qual ela se refere e vários apontadores para os nós onde aquelas informações estão armazenadas. Vale salientar que a tabela-índice é armazenada no próprio banco de dados e é particionada juntamente com as informações sobre os recursos; deste modo, esta tabela está sempre sincronizada com as novas informações publicadas no catálogo. Uma operação que contém atributos exclusivamente de tabelas secundárias é roteada da seguinte forma: um dos índices secundários correspondente a qualquer das tabelas secundárias referenciadas na cláusula SQL é usado para rotear a operação para o nó que mantém a tabela-índice correspondente. Este nó, por sua vez, se encarrega de encaminhar a operação para os nós que realmente são responsáveis por armazenar aquelas informações. Esse mecanismo é uma alternativa ao uso de *flooding*, geralmente utilizada por outras soluções P2P.

3.4. Impacto da falha de um nó

Como as informações sobre os recursos e os apontadores das tabelas-índice são armazenados através da abordagem *soft-state* e os provedores publicam as informações sobre

seus recursos periodicamente, a falha de um nó não deverá trazer grande impacto ao sistema como um todo, uma vez que apenas parte das informações serão perdidas. Além disso, as mesmas informações deverão ser reinseridas periodicamente no sistema pelos provedores.

Os índices também poderão ser reconstituídos com o mecanismo de tolerância a falhas proposto para o NW [Basu et al. 2009]. A reconstituição dos índices se dá quando a falha de um nó é detectada por outro nó monitor. Um outro nó é escolhido para assumir o espaço de atributos que era da responsabilidade do nó falho e as informações que eram armazenadas no nó falho são perdidas momentaneamente até que o provedor dos recursos as republique.

4. Avaliação

Para avaliarmos a solução, nós desenvolvemos dois simuladores. Um simulador para o NW-R e outro para o SBD P2P PeerDB, que é um dos sistemas P2P que mais se aproxima da solução proposta pelo NW-R em termos de funcionalidades. O objetivo dessa avaliação foi analisar a eficiência do mecanismo de indexação do NW-R *versus* o mecanismo adotado por PeerDB, o qual é baseado em *flooding*.

O simulador do PeerDB foi implementado para operar em três modos distintos: estático, estático estendido e dinâmico. No modo estático (**PeerDB**), cada nó se conecta diretamente a $log(n)$ nós. No modo estático estendido (**PeerDB +**), cada nó se conecta a $log(n) + c$ nós, onde c é uma constante inteira dada. Essa constante foi previamente calculada para que se obtivesse uma rede totalmente conectada de acordo com o número de nós utilizados na simulação. No modo dinâmico (**PeerDB D**), cada nó se conecta diretamente a $log(n)$ nós e pode se conectar diretamente a outros nós mais promissores gradativamente, sob o critério do número de resultados retornados pelos nós em consultas anteriores. Em todos os casos, nós variamos o nível máximo de inundação usado (TTL) a fim de se verificar qual valor seria necessário para se obter a maior taxa de cobertura. O TTL é o artifício usado pelo PeerDB para limitar a quantidade de mensagens transmitidas. Como o NW-R não utiliza TTL, ele foi considerado apenas para o PeerDB. O TTL foi configurado com os valores 1, 3, 5, 6, 8 e 10.

Foram usadas duas métricas para comparar os sistemas: sobrecarga e cobertura. A sobrecarga foi quantificada através da quantidade de informação que os processos distribuídos que implementam o serviço precisam trocar em cada operação, sendo uma indicação indireta da eficiência do serviço. Quanto menor a sobrecarga mais eficiente é a solução; a sobrecarga foi medida em $Kbytes$. A cobertura, é uma medida da acurácia do sistema e foi calculada como sendo a relação entre a quantidade de recursos que foram efetivamente descobertos e a quantidade de recursos que deveriam ter sido descobertos. Idealmente, a cobertura deve ser de 100%.

4.1. Configuração do ambiente das simulações e cenários

As simulações foram executadas para redes lógicas compostas por 500, 1.000, 3.000, 5.000 e 10.000 nós. Foram geradas 10 cargas de trabalho sintéticas diferentes, cada uma com informações sobre 100.000 recursos distintos. No caso do NW-R, a distribuição das informações foi feita automaticamente pelo mecanismo de distribuição e balanceamento de carga do sistema. No PeerDB, as informações foram distribuídas de forma que cada nó mantivesse um conjunto de informações com valores aproximados entre si, simulando a

proximidade semântica e de valores, geralmente encontrada num provedor de recursos em um cenário real[2]. As informações foram divididas de forma igualitária entre todos os nós.

O esquema de dados utilizado continha uma tabela principal e uma tabela secundária. Ambas as tabelas continham dois atributos do tipo *Double*. Os valores para os atributos foram distribuídos uniformemente sobre uma faixa de valores pré-definida. Por simplicidade, assumiu-se que os dados não expiravam, assim, todas as consultas submetidas poderiam retornar a quantidade de resultados esperados, em ambos os sistemas. Foram submetidas 100 consultas, geradas randomicamente a partir de um subconjunto das informações publicadas. Dessa forma, todas as consultas submetidas casavam com pelo menos um recurso e no caso de consultas por faixa de valores, mais de um resultado poderia ser retornado. A escolha do nó para o qual a consulta seria submetida foi feita aleatoriamente, de forma que qualquer nó possuiria a mesma chance de ser selecionado para recebê-la.

Foram gerados 4 tipos de cenários que permitiram analisar o comportamento do sistema diante de diferentes tipos de consultas. Cada cenário foi executado 10 vezes sob cargas de trabalho diferentes, que foram suficientes para obter valores médios com um nível de confiança de 95% e um erro máximo menor que 3%. Os cenários para as simulações são descritos abaixo:

- *Cenário 1*: consultas com casamento exato de valores (*exact-match queries*) sobre os 2 atributos da tabela principal;
- *Cenário 2*: consultas por faixa de valores (*range queries*) com o operador $>$ (maior que) sobre 1 atributo da tabela principal;
- *Cenário 3*: consultas com casamento exato de valores sobre 1 atributo da tabela secundária. A fim de se aproximar de um caso real de uso, foi reduzida a faixa de valores possíveis para os atributos da tabela secundária. Isso fez com que os dados da tabela secundária fossem replicados em diversos nós no momento da divisão do espaço de atributos.
- *Cenário 4*: consultas por faixa de valores com operador $>$ sobre 2 atributos da tabela secundária e com replicação dos valores da tabela secundária (semelhante ao cenário 3).

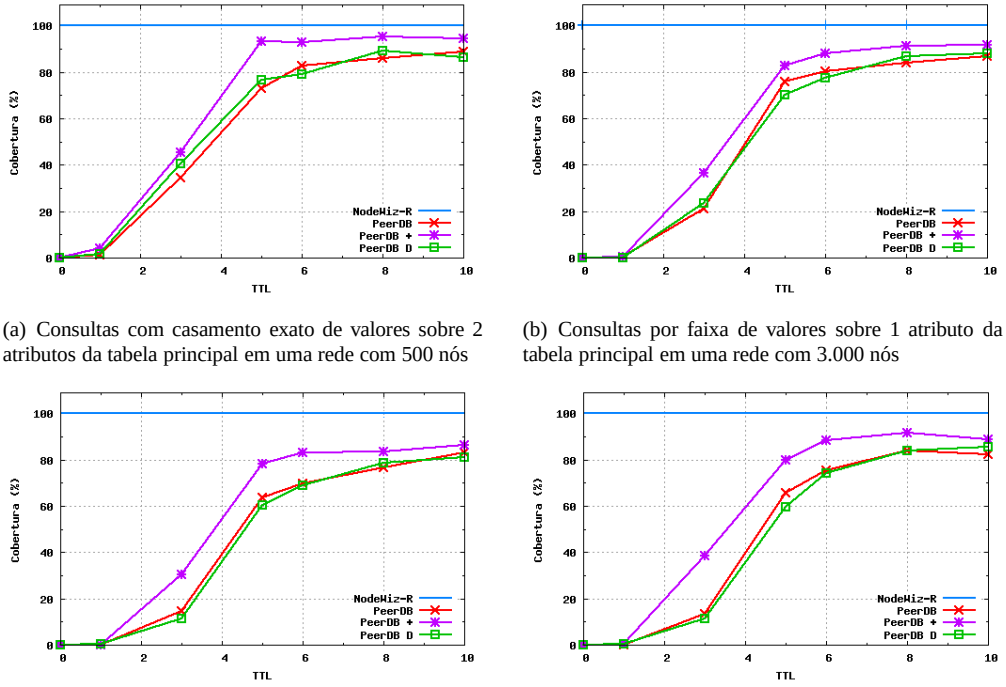
4.2. Resultados

4.2.1. Cobertura x TTL

Em todos os cenários avaliados o NW-R conseguiu obter uma taxa de cobertura de 100%, como podemos observar na Figura 5. No caso do **PeerDB** e **PeerDB D**, a taxa de cobertura média ultrapassa 70% apenas quando o TTL é superior a 6. A partir daí o aumento na cobertura é pequeno à medida que o TTL aumenta, sendo sempre abaixo de 90%. O **PeerDB +** obteve resultados um pouco melhores, conseguindo taxas de cobertura superiores a 90%. Nesse caso, a quantidade de nós conectados diretamente é maior desde o início da simulação, permitindo que mais nós sejam consultados com o mesmo TTL, aumentando a quantidade de resultados retornados. Entretanto, quanto maior o valor para o TTL maior será a sobrecarga, como veremos a seguir.

Podemos observar também que mesmo com o incremento no tamanho da rede, o NW-R ainda consegue obter uma cobertura de 100%, enquanto PeerDB apresenta uma taxa de cobertura sempre inferior. Os resultados para os demais cenários são semelhantes, mas não serão apresentados aqui por limitação de espaço.

[2]Também rodamos simulações onde a distribuição de dados no PeerDB seguia a mesma distribuição do NW-R; como esperado, nessas condições o desempenho do NW-R ainda é superior, quando comparado ao PeerDB.



(a) Consultas com casamento exato de valores sobre 2 atributos da tabela principal em uma rede com 500 nós

(b) Consultas por faixa de valores sobre 1 atributo da tabela principal em uma rede com 3.000 nós

(c) Consultas com casamento exato de valores sobre 1 atributo da tabela secundária em uma rede com 5.000 nós

(d) Consultas por faixa de valores sobre 2 atributos da tabela secundária em uma rede com 10.000 nós

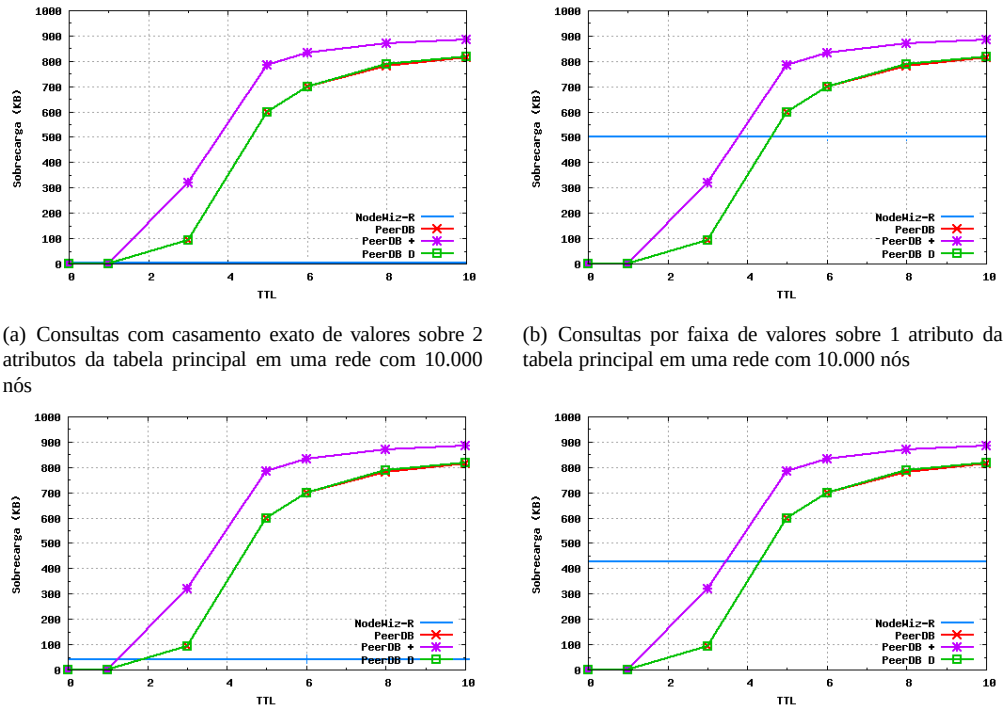
Figura 5. Taxa de Cobertura x TTL

4.2.2. Sobrecarga x TTL

A Figura 6 mostra a sobrecarga gerada versus o TTL usado para se alcançar a taxa de cobertura apresentada anteriormente. Embora **PeerDB** (em todos os modos de operação) tenha alcançado uma taxa de cobertura entre 85% e 90%, a quantidade de mensagens trocadas para se chegar a esse resultado foi substancial. Podemos observar que, para os cenários com consultas por casamento exato de valores, a sobrecarga gerada por NW-R foi mínima em relação à sobrecarga gerada por **PeerDB**. Isso se deve ao fato de uma menor quantidade de nós serem contatados e, consequentemente, uma menor quantidade de mensagens serem trocadas.

Esse comportamento já era esperado, uma vez que no NW-R somente um seleto número de nós são escolhidos para responder uma consulta. Além disso, quanto mais seletiva for a consulta mais eficiente será o roteamento da mesma. No caso de consultas com casamento exato de valores, que são altamente seletivas, poucos nós devem respondê-las. Já o **PeerDB** é insensível aos tipos de consultas enviadas, fazendo *flooding* para propagar qualquer consulta pela rede.

Por outro lado, como podemos observar na Figura 6(b) e 6(d), a sobrecarga gerada por NW-R para consultas por faixa, é bem mais elevada em relação às consultas com casamento exato de valores, dos cenários 1 e 3 (Figura 6(a) e 6(c)). Isso acontece porque uma consulta por faixa é menos seletiva do que uma consulta com casamento exato de valores. Uma ampla faixa de valores significa baixa seletividade. Assim, o número de nós contatados aumenta com a faixa de valores requerida na consulta. O aumento na sobrecarga já era esperado pelo fato das consultas do cenário 3 e 4 serem relacionadas às tabelas secundárias (Figura 6(c) e 6(d)) e considerando a replicação de dados, o que



(a) Consultas com casamento exato de valores sobre 2 atributos da tabela principal em uma rede com 10.000 nós

(b) Consultas por faixa de valores sobre 1 atributo da tabela principal em uma rede com 10.000 nós

(c) Consultas com casamento exato de valores sobre 1 atributo da tabela secundária em uma rede com 10.000 nós

(d) Consultas por faixa de valores sobre 2 atributos da tabela secundária em uma rede com 10.000 nós

Figura 6. Sobrecarga em termos de mensagens trocadas x TTL

implica em um maior número de nós a serem contatados.

Em resumo, as simulações demonstraram que o NW-R oferece uma solução melhor comparada àquela do PeerDB, e de modo geral aos sistemas que utilizam técnicas de *flooding* para disseminar consultas. Em um ambiente livre de falhas, o NW-R sempre retornou 100% dos resultados com uma baixa sobrecarga, principalmente para consultas mais seletivas, independente da tabela do esquema consultada.

5. Conclusão

Neste trabalho nós apresentamos o NW-R, um sistema P2P relacional que indexa recursos de forma expressiva, eficiente e auto-gerenciável. Nós comparamos a nossa solução com outra de propósito similar, PeerDB. Os resultados das simulações mostraram que o NW-R é bem mais eficiente em termos de escalabilidade e cobertura, o que indica que a nossa solução é uma alternativa viável aos atuais sistemas de descoberta de recursos para sistemas abertos.

Foram executados experimentos com um protótipo da solução para validar as simulações. Os experimentos foram conduzidos em uma rede local, para uma rede lógica composta por 64 e 128 nós. Os resultados apresentaram somente pequenas variações nas médias de nós contatados para alguns cenários, mas comprovou-se que os sistemas são estatisticamente indistinguíveis (utilizando o método t-test).

Nosso objetivo atual é usar o NW-R para melhorar o sistema de *match-making* na comunidade do OurGrid, um grid P2P que está em produção desde 2004, além de utilizá-lo como serviço de localização de dados ambientais no projeto SegHidro.

Referências

Araújo, E., Cirne, W., Wagner, G., Oliveira, N., Souza, E., Galvão, C., and Martins, E. (2005). The SegHidro Experience: Using the Grid to Empower a Hydro-Meteorological Scientific Network. *Proceedings of 1st IEEE Conference on e-Science and Grid Computing*, pages 64–71.

Basu, S., Costa, L., Brasileiro, F., Banerjee, S., Sharma, P., and Lee, S.-J. (2009). Nodewiz: Fault-tolerant grid information service. *Peer-to-Peer Networking and Applications*, 2(4):348–366.

Boncz, P. and Treijtel, C. (2004). AmbientDB: Relational Query Processing in a P2P Network. *Databases, Information Systems, and Peer-To-Peer Computing: First International Workshop, DBISP2P 2003: Berlin, Germany, September 7-8, 2003: Revised Papers*.

Cirne, W., Brasileiro, F., Andrade, N., Costa, L., Andrade, A., Novaes, R., and Mowbray, M. (2006). Labs of the World, Unite!!! *Journal of Grid Computing*, 4(3):225–246.

Harchol-Balter, M., Leighton, T., and Lewin, D. (1999). Resource discovery in distributed networks. In *PODC '99: Proceedings of the eighteenth annual ACM symposium on Principles of distributed computing*, pages 229–237, New York, NY, USA. ACM.

Heine, F., Hovestadt, M., and Kao, O. (2005). Processing Complex RDF Queries Over P2P Networks. In *Proceedings of the 2005 ACM workshop on Information retrieval in peer-to-peer networks*, pages 41–48. ACM New York, NY, USA.

Huebsch, R., Chun, B., Hellerstein, J., Loo, B., Maniatis, P., Roscoe, T., Shenker, S., Stoica, I., and Yumerefendi, A. (2005). The Architecture of PIER: an Internet-Scale Query Processor. *Proc. Conference on Innovative Data Systems Research (CIDR)(Jan. 2005)*.

Iamnitchi, A., Foster, I., and Nurmi, D. C. (2002). A Peer-to-Peer Approach to Resource Location in Grid Environments. In *HPDC '02: Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing*, page 419, Washington, DC, USA.

Kossmann, D. (2000). The State of the Art in Distributed Query Processing. *ACM Computing Surveys (CSUR)*, 32(4):422–469.

Liarou, E., Idreos, S., and Koubarakis, M. (2007). Continuous RDF Query Processing Over DHTs. In *In Proceedings of* 6^{th} *International Semantic Web Conference (ISWC), 2nd Asian Semantic Web Conference (ASWC)*, pages 324–339.

Ng, W., Ooi, B., Tan, K., and Zhou, A. (2003). Peerdb: A p2p-based system for distributed data sharing. *Proceedings of the 19th International Conference on Data Engineering*.

Oppenheimer, D., Albrecht, J., Patterson, D., and Vahdat, A. (2005). Design and implementation tradeoffs for wide-area resource discovery. In *HPDC '05: Proceedings of the High Performance Distributed Computing, 2005. HPDC-14. Proceedings. 14th IEEE International Symposium*, pages 113–124, Washington, DC, USA. IEEE Computer Society.

Rhea, S., Geels, D., Roscoe, T., and Kubiatowicz, J. (2004). Handling churn in a DHT. In *Proceedings of the USENIX Technical Conference*.

Sidirourgos, L., Kokkinidis, G., and Dalamagas, T. (2005). Efficient Query Routing in RDFS schema-based P2P Systems. In *Proceedings of the* 4^{th} *Hellenic Data Management Symposium (HDMS'05)*, Athens, Greece.