Um sistema distribuído para busca de caminhos em grafos dinâmicos

Marcelo Vinagreiro
Alfredo Goldman
Instituto de Matemática e Estatística
Universidade de São Paulo
Rua do Matão, 1010 - CEP 05508-090 - São Paulo - SP
{mlv, gold}@ime.usp.br

Resumo

Este trabalho descreve um novo modelo para cálculo concorrente de caminhos em grafos dinâmicos os quais podem estar particionados em um conjunto de servidores interconectados. Aspectos dinâmicos em cálculos de caminhos têm sido bem explorados em trabalhos anteriores, neste trabalho, consideramos também aspectos de distribuição. O arcabouço proposto pode ser usado com algoritmos de cálculo de caminhos dinâmicos ou estáticos. O modelo pode ser usado em sistemas simulando o estado de uma rede ou o monitormento das condições de trânsito de uma cidade. O artigo também apresenta alguns detalhes de implementação bem como resultados de testes.

1 Introdução

Um dos principais assuntos em Teoria de Grafos [1] é o problema da busca de caminhos entre dois ou mais pontos de um grafo. Este problema já foi bem estudado, principalmente para grafos estáticos. Neste trabalho, apresentamos um modelo que permite o cálculo de estimativas de caminhos em um grafo dinâmico, ou seja, um grafo no qual os pesos de suas arestas mudam com o decorrer do tempo. Diferentemente de trabalhos similares, como o proposto por Fawcett e Robinson [2], nos quais, a computação de rotas é realizada em um único servidor, que mantém o grafo completo em memória, de forma centralizada, o trabalho aqui apresentado permite que o grafo esteja particionado em um conjunto de servidores, que se comunicam através de uma rede de interconexão, a qual supostamente não falha.

O principal objetivo é apresentar soluções eficientes e escaláveis que permitam manter um modelo de monitoramento das condições de um sistema que possa ser representado na estrutura de um grafo dinâmico, como por exemplo, o monitoramento das condições do trânsito de uma dada região, como a cidade de São Paulo. Um sistema dessa natureza pode ser encontrado no sítio do projeto SIDAM[1] (Sistemas de Informações Distribuídas para Agentes Móveis). Neste trabalho nós não estudamos formas de particionar o grafos nos diversos servidores, este é um tópico de pesquisa futura.

Esse texto está organizado da seguinte forma: na próxima seção serão apresentados trabalhos relacionados ao problema exposto. Na seção 2 é mostrado o algoritmo de busca de caminhos em grafos dinâmicos proposto por Narváez et al.[5]. Na seção 3 é definido o algoritmo para roteamento distribuído baseado em árvores dinâmicas. Na seção 4 são mostrados alguns resultados obtidos e finalmente a seção 5 apresenta algumas conclusões. Deve-se ressaltar que não foi encontrado na literatura nenhum trabalho que tratasse ao mesmo tempo a distribuição e a possibilidade de mudanças em um grafo.

1.1 Trabalhos relacionados

1.1.1 Algoritmos para caminho mínimo

Um dos principais assuntos em teoria de grafos é o problema do caminho mínimo (*Shortest Path Problem*) [1]. Nesse problema, deseja-se encontrar uma rota mais curta (se alguma existir) entre dois vértices de um grafo. Ou seja, dado um grafo $G = (V, A)$, sendo V o conjunto de vértices e A o conjunto de arcos, associa-se uma função $w : A \rightarrow \Re$ mapeando os pesos de cada arco, onde, a função pode representar possíveis critérios de minimização. Assim, o peso de um caminho $p = (v_0, v_1, ..., v_k)$ é a soma dos pesos de seus arcos: $w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$. Um caminho de peso mínimo de um vértice u a um vértice v é o de peso $min(w(p))$ se existe ao menos um caminho, e ∞, caso contrário.

Para resolver esse problema, existem vários algoritmos cada qual com peculiaridades próprias, por exemplo, o al-

[1] `http://www.ime.usp.br/~sidam/`

goritmo de Bellman-Ford descrito na seção 24.1 de [1]) resolve o problema para grafos com pesos quaisquer, com valores negativos ou não. Por outro lado, o algoritmo proposto por Dijkstra (descrito na seção 24.3 de [1]) admite grafos com pesos não-negativos.

1.1.2 Algoritmos para grafos dinâmicos

Em geral, muitos problemas que envolvem grafos apresentam uma natureza dinâmica, tal como simular o estado de uma rede em um grafo, onde constantes atualizações dos valores dos pesos bem como a inserção/remoção de arcos ocorrem freqüentemente. Dessa forma, deseja-se encontrar algoritmos que realizem buscas de rotas em grafos dinâmicos de uma forma eficiente. Para tanto os algoritmos propostos por Ramalingan e Reps (RR) [6], Frigioni, Marchetti e Nanni (FMN) [3] ou Narváez et al. [5] têm gerado bons resultados em termos práticos, comparando-se com as melhores implementações de algoritmos de caminhos mínimos em grafos estáticos, como o algoritmo de Dijkstra implementado com filas de prioridades.

A vantagem dos algoritmos dinâmicos em relação aos algoritmos estáticos é que eles conseguem realizar uma atualização na árvore de caminhos mínimos considerando apenas os vértices afetados por mudanças de peso de arcos do grafo, ao passo que os algoritmos estáticos recalculam a árvore considerando-se todos os vértices do grafo.

2 Algoritmo dinâmico de Narváez et al.

Neste item são apresentados brevemente os conceitos gerais envolvidos no trabalho proposto por P. Narváez, K. Siu e H. Tzeng [5] para realizar a atualização de árvores de caminhos em grafos que sofrem alterações de seus pesos. Em geral, os algoritmos de cálculos de caminhos em grafos dinâmicos assumem a existência de uma árvore de caminhos mínimos previamente calculada. A partir dessa árvore, eles são executados considerando-se as mudanças de pesos dos arcos do grafo.

O arcabouço proposto é constituído por um algoritmo básico (*basic algorithm*). Esse algoritmo mantém uma fila Q que é inicializada com os nós que serão afetados pela alteração de peso de um arco, sendo que para cada nó, são mantidos também a nova distância potencial em relação à raiz e o possível novo nó pai na árvore. Dependendo de como a fila Q é implementada, o arcabouço pode se transformar em versões dinâmicas de algoritmos estáticos bem conhecidos, como o de Dijkstra.

O algoritmo básico pode se especializar em duas modalidades: o primeiro método incremental ou o segundo método incremental.Simulações realizadas pelos autores mostram que o segundo método incremental possui um melhor desempenho na média embora apresente soluções com qualidade inferior ao primeiro método.

No trabalho apresentado neste artigo, os testes utilizando algoritmos de busca dinâmicos são feitos com os algoritmos propostos por Narvaez et al., pois, dentre outros motivos, os resultados obtidos são tão bons quanto os melhores algoritmos de busca em grafos dinâmicos conhecidos. Maiores detalhes podem ser encontrados na referência [5].

3 Roteamento adaptativo através de árvores dinâmicas

A partir deste momento, serão apresentadas as idéias do sistema proposto neste trabalho para permitir o cálculo de estimativas de caminhos em grafos distribuídos e que sofrem atualizações. A denominação aqui utilizada para o sistema é "Roteamento adaptativo através de árvores dinâmicas", justamente por se basear em árvores de caminhos que sofrem mudanças.

O principal objetivo do trabalho mostrado a partir desta seção, é apresentar um arcabouço que permite prover estimativas para caminhos em grafos dinâmicos particionados em mais de um servidor, utilizando-se algoritmos conhecidos para computação de caminhos.

Conforme será visto na seção 4, o modelo foi testado tanto com algoritmos estáticos quanto dinâmicos para cálculo de caminhos.

3.1 Considerações iniciais

Assuma inicialmente um domínio qualquer de aplicação que possa ser modelado na estrutura de um grafo G. Considera-se que o grafo G representando o domínio da aplicação está particionado em N subgrafos $G_i = (V_i, A_i)$, onde $1 \leq i \leq N$.

Cada subgrafo G_i está armazenado na memória de um servidor chamado de estação-base (EB_i). As estações-base se comunicam através de uma rede de interconexão fixa. Todas as atualizações de pesos dos arcos contidos no subgrafo G_i devem ser encaminhadas à estação EB_i. As estações-base também são responsáveis por receber requisições de clientes e fornecer rotas entre dois pontos de G.

A região de fronteira da área de cobertura de EB_i é representada por um conjunto de vértices especiais, os quais formam a borda ($B(G_i)$) de EB_i. As bordas são nós de junção inter-estações, ou seja, o grafo G, representante do domínio completo, é formado pelo conjunto dos subgrafos G_i colapsados pelas bordas $B(G_i)$. Considere também que $G_i \bigcap G_j = B(G_i) \bigcap B(G_j)$, ou seja, a intersecção dos vértices entre estações é dada pela intersecção dos conjuntos de nós das respectivas bordas, sem a existência de nós

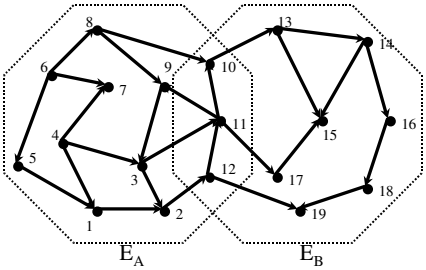


Figura 1. Topologia de um domínio de aplicação sob a cobertura de duas EBs adjacentes.

internos. Na figura 3.2, duas estações E_A e E_B são responsáveis pela cobertura de todo um domínio de aplicação (G). Nesse caso:

$$B(G_A) = \{1, 2, 5, 6, 8, 10, 11, 12\}$$

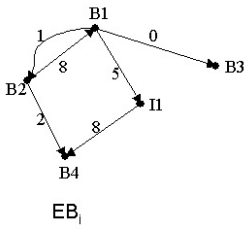
$$B(G_B) = \{10, 11, 12, 13, 14, 16, 18, 19\}$$

$$B(G_A) \bigcap B(G_B) = \{10, 11, 12\}$$

Uma estação-base somente conhece o estado do subgrafo que está contido em seu interior. Dessa forma, para calcular um caminho entre dois pontos quaisquer do domínio de aplicação, as estações se comunicam com um conjunto especial de servidores que possuem informações do estado global do sistema. Esses servidores são denominados de servidores de busca (SB). Uma estação-base também pode ser um servidor de busca. Assume-se que a rede interligando servidores de busca e estações-base é confiável e que as mensagens chegam aos seus destinos na ordem em que foram enviadas.

Seja o **Conjunto *Raízes*** de um grafo G_i, ou $Raizes(G_i)$, como um subconjunto qualquer de $B(G_i)$. Ou seja, $Raizes(G_i) \subseteq B(G_i)$. Nesse caso, define-se a **Tabela *Custos de Caminhos entre Raízes*** de um grafo G_i, ou $TabCustRaizes(G_i)$, como sendo o conjunto dos custos dos melhores caminhos conhecidos ligando todas as combinações possíveis de trajetos entre nós presentes no conjunto $Raizes(G_i)$.

A figura 2 apresenta um grafo G_i com 4 nós em $Raizes(G_i) = \{B_1, B_2, B_3, B_4\}$ e um nó interno I_1. O conjunto $TabCustRaizes(G_i)$ é mostrado na tabela da figura 2, contendo os melhores caminhos entre os pontos no Conjunto Raízes, por exemplo, de R_1 a R_4, o caminho mínimo tem custo 3.



TabCustRaizes(Gi):

Pares	X->Y	Y->X
B1-B2	1	8
B1-B3	0	-
B1-B4	3	-
B2-B4	2	-

Figura 2. Conjunto Raízes e Tabela Custos de Caminhos entre Raízes de G_i

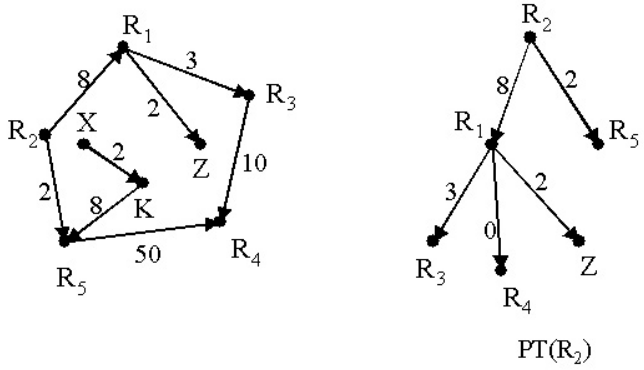


Figura 3. Árvore pórtico com raiz R_2

3.2 Árvore pórtico

Considere a árvore de caminhos mínimos (T) com raiz r calculada a partir de algum algoritmo de busca de caminhos, como o de Dijkstra [1]. Seja $r \in Raizes(G_i)$, definimos a árvore pórtico (ou apenas pórtico) de raiz r, $PT(r)$, como a árvore de caminhos mínimos com raiz em r até todos pontos de G_i alcançáveis a partir de r. A figura 3 apresenta uma árvore pórtico com raiz em B_2, $PT(R_2)$.

3.3 Árvore-modelo

Para otimizar o tempo de cálculo de caminhos, os servidores de busca podem utilizar estruturas pré-calculadas, denominadas de árvores-modelo. Considere um servidor de busca SB_j e um domínio de aplicação particionado em N estações-base EB_i, com $1 \leq i \leq N$. Nessa situação, SB_j mantém um grafo SG_j com os valores $Raizes(G_i)$ das estações EB_i. Para cada conjunto de nós em $Raizes(G_i)$ são criados vértices especiais $Origem(G_i)$ e $Sorvedouro(G_i)$. São adicionados arcos especiais partindo de $Origem(G_i)$ e chegando em P tal que $P \in Raizes(G_i)$. Analogamente, são inseridos arcos especiais de Q a $Sorvedouro(G_i)$ tal que $Q \in Raizes(G_i)$.

A figura 4 apresenta a ilustração de um grafo SG

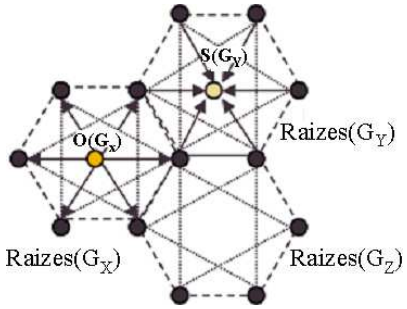


Figura 4. Vértices especiais Origem e Sorvedouro.

contendo vértices de 3 Conjuntos Raízes (X, Y, Z), bem como os arcos representando as ligações entre os nós dos Conjuntos Raízes. Também são apresentados os nós $Origem(G_X)$ ($O(G_X)$) ligados aos nós do Conjunto $Raizes(G_X)$ e $Sorvedouro(G_Y)$ ($S(G_Y)$) ligando os nós do Conjunto $Raizes(G_Y)$ a $S(G_Y)$. Os nós $Origem(G_Y)$, $Origem(G_Z)$, $Sorvedouro(G_X)$ e $Sorvedouro(G_Z)$ bem como os pesos dos arcos foram omitidos para não prejudicar a visualização da figura.

Os nós e arcos especiais são adicionados no momento da inicialização do grafo em SB_j. O peso atribuído aos arcos especiais é zero. Os arcos especiais permitem que vários valores de pesos sejam a eles associados, desde que apresentem um conjunto de identificadores único. Isso permite que vários processos clientes que estão realizando cálculos de caminhos possam modificar os pesos dos arcos especiais de acordo com suas necessidades.

Basicamente, uma árvore-modelo MT_i é a árvore de caminhos com raiz em $Origem(G_i) \in SG$. Cada servidor de busca SB_j mantém uma árvore modelo para cada conjunto $Raizes(G_i)$, representando informações de EB_i. Conforme será apresentado na próxima seção, em intervalos regulares, há um processo coordenador que solicita a atualização do estado global do sistema para um dado estado k. Assim, considere MT_i^k a árvore-modelo com raiz em $Origem(G_i)$, calculada com os dados do estado k.

Assim que uma solicitação de cálculo de caminhos de X a Y no estado k chega em SB_j, onde $X \in Raizes(G_i)$ e $Y \in Raizes(G_j)$, SB_j pode utilizar MT_i^k previamente calculada para acelerar o cálculo. Para isso, SB_j necessita receber informações atualizadas dos valores dos pesos dos arcos de $Origem(G_i)$ a R tal que $R \in Raizes(G_i)$ (conjunto *CustosX_ate_Raizes*) e de R a $Sorvedouro(G_j)$ tal que $R \in Raizes(G_j)$ (conjunto *CustosRaizes_ate_Y*) e, com esses valores, executa o algoritmo descrito na seção 2 sobre uma cópia temporária dos dados[2] que definem MT_i^k, assumindo as atualizações de pesos recebidas dos conjuntos *CustosRaizes_ate_Y* e *CustosX_ate_Raizes*. Maiores detalhes podem ser encontrados na seção 3.4.

[2]Apenas os dados são copiados. Os vértices e arcos são únicos em SG, pois permitem armazenar múltiplas informações indexadas por identificadores únicos.

3.4 Proposta deste trabalho

A partir deste momento é apresentado o sistema proposto neste trabalho. Em princípio, podem ser definidas duas funcionalidades: busca de caminhos e atualização de estados. Por busca de caminhos entenda-se a capacidade de prover estimativas de rotas a clientes da aplicação. Além disso, dadas as características dinâmicas da aplicação, é necessário também proporcionar mecanismos de atualização da base de informação periodicamente. Existem dois conjuntos de servidores: estações-base e servidores de busca. De acordo com o tipo de funcionalidade a qual o servidor se destina, ele terá um conjunto definido de ações a serem desempenhadas. A próxima seção apresenta as idéias gerais do sistema. Maiores detalhes podem ser encontrados em `http://www.ime.usp.br/~mlv/distgraph/index.html`.

3.5 Processamento

O processamento do servidor depende do seu papel no sistema. Para tanto, assuma o parâmetro k como sendo o identificador único de um dado estado global do sistema.

3.5.1 Processamento em EB_i

Cada estação-base EB_i realiza os seguintes passos:

1. EB_i mantém o subgrafo G_i em memória;
2. Em intervalos regulares, EB_i obtém/atualiza as árvores pórticos $PT(r)$, $r \in Raizes(G_i)$. O procedimento de atualização do estado global (`k`) do sistema será discutido em maiores detalhes na seção 3.8;
3. Com os dados de $PT(r)$ para o estado `k`, EB_i gera $TabCustRaizes(G_i)^k$ e envia para o conjunto de servidores de busca (SB).

Deve-se ressaltar que na etapa 2, a atualização de $PT(r)$ pode ser realizada através de um algoritmo de busca, como o proposto por Narváez et al. [5]. O sistema permite processamento concorrente no grafo G_i durante a execução de algoritmos para construção/atualização de árvores de caminhos.

3.6 Processamento em SB_j

Cada servidor de busca SB_j recebe das estações-base o conjunto $TabCustRaizes$ descrevendo os custos dos melhores caminhos ligando todas as combinações possíveis de pontos presentes nos respectivos conjuntos $Raizes$ para um dado estado k.

Assim que todos conjuntos $TabCustRaizes(G_i)^k$, $1 \leq i \leq N$ são recebidos SB_j cria/atualiza as árvores MT_r^k. O procedimento de atualização do estado global (k) do sistema será discutido em maiores detalhes na seção 3.8. Após o instante em que as árvores MT_r^k tenham sido geradas/atualizadas, para cada conjunto $TabCustRaizes(G_i)^k$ recebido, SB_j envia uma mensagem para EB_i indicando que o conjunto $TabCustRaizes(G_i)^k$ foi recebido com êxito.

3.7 Busca de caminhos

Seja uma requisição de caminho de X a Y enviada à estação-base EB_X. O processamento de busca de caminhos é feito em dois níveis:

Intra-estação: busca no interior de EB_X e/ou,

Inter-estação: nesse caso EB_X consulta algum servidor de busca SB_i para incluir no cálculo a informação do estado global do sistema.

Os clientes podem solicitar dois tipos de requisições de busca:

Global: um caminho completo de X a Y. Neste caso, deseja-se encontrar todos os arcos que compõe o caminho.

Local: um subcaminho de X a Y. Esse subcaminho é um caminho de X a R tal que R está em $Raizes(G_X)$. Nesse tipo de requisição o sistema escolhe com base no estado global do sistema, o caminho de menor custo de X a Y, mas retorna ao cliente apenas o subcaminho de X a R, conforme descrito anteriormente.

As requisições locais podem ser particularmente interessantes para dispositivos móveis, uma vez que a unidade móvel poderia solicitar trechos de caminho antes de cada *handoff* [7] efetuado entre estações, mantendo informações mais atualizadas. Além disso, a resposta também poderia ser usada em algoritmos que fornecem suporte para agendamento de *handoffs* futuros.

A busca de um caminho de X a Y é realizada por 4 procedimentos principais:

- **BuscaEB**: executado em EB_X;
- **BuscaInterEB**: executado em EB_X;
- **ConsultaPorticEB**: executado em EB_Y;
- **BuscaSB**: executado em SB_j.

Uma requisição (req) de caminho de um ponto X a um destino Y é enviada à estação-base EB_X, a qual executa o método **BuscaEB**. Este método calcula uma árvore com raiz X ($T(X)$), com as informações de um dado estado, (k), em geral, o estado mais recente. Caso o ponto Y não esteja em $T(X)$ ou o caminho encontrado em $T(X)$ não seja satisfatório, o método **BuscaInterEB** é executado.

O procedimento **BuscaInterEB** escolhe um servidor de busca SB_j para solicitar que seja calculado um caminho entre os pontos X e Y. Dessa forma, a partir de $T(X)$, esse procedimento constrói um conjunto (*CustosX_ate_Raizes*), contendo os pesos do melhores caminhos de X até $Raizes(G_X)$, a ser enviado para SB_j.

SB_j recebe como parâmetros o conjunto *CustosX_ate_Raizes*, X, Y, a requisição req e o estado (k) passado por EB_X e executa o procedimento **BuscaSB**, que calcula um caminho de X a Y, se existir. Para tanto, SB_j pode invocar remotamente na estação-base EB_Y o método **ConsultaPorticEB** para obter informações sobre os pesos dos melhores caminhos conhecidos de $Raizes(G_Y)$ a Y, para o estado *k*.

SB_j utiliza a árvore-modelo MT_X^k. Conforme foi apresentado no item 3.3, as árvores-modelo mantêm informações dos custos dos melhores caminhos conhecidos entre todos os pontos $R \in Raizes(G_i)$ tais que $1 \leq i \leq N$. Estes valores são calculados em intervalos de tempo conforme explicado no item 3.6.

Os conjuntos *CustosX_ate_Raizes* e *CustosRaizes_ate_Y* são calculados a cada requisição de caminho, para obter os pesos dos melhores caminhos conhecidos de X a $Raizes(G_X)$ e de $Raizes(G_Y)$ a Y, respectivamente. Resumidamente, a função **BuscaSB** obtém um caminho P' entre X e Y, para o estado *k*. Para isso, um algoritmo como o dinâmico proposto por [5] é executado sobre MT_X^k, após ter sido aplicado à MT_X^k os valores de pesos descritos nos conjuntos *CustosX_ate_Raizes*, *CustosRaizes_ate_Y*, conforme descrito na seção 3.3.

O resultado retornado pela função **BuscaSB** depende do tipo de requisição: se a requisição for local, é retornado o nó R_t em $Raizes(G_X)$ que origina o melhor caminho conhecido para chegar em Y bem como o custo C calculado para o caminho de X a Y. No caso de uma requisição do tipo Global, é retornado o caminho completo P calculado de X até Y. Nesse caso, após o servidor de busca obter a rota de X a Y, presente no caminho P', obtido com os dados de SG, o servidor de busca SB_j necessita contactar cada uma das estações EB_i que possuem um subcaminho P_i de P', visando obter a descrição completa dos nós que compõe P_i.

3.8 Atualização do sistema

O processo de atualização do sistema segue um modelo parecido com a técnica de barreiras de sincronização [4], utilizada em sistemas paralelos e distribuídos. Para tanto, é escolhido um coordenador dentre as estações-base do sistema (EB_C), o qual será o responsável por iniciar o processo de atualização global. O coordenador envia uma mensagem *Update_Rset(k+1)* (supondo-se que o estado atual é k, $k+1$ identifica o novo estado) para cada uma das estações-base. Uma mensagem com a ação *Update_Rwset(k+1)* e contendo o novo conjunto $TabCustRaizes^{k+1}$ é enviada para os servidores de busca do sistema.

Assim que um servidor de busca SB_j recebe a mensagem *Update_Rwset(k+1)* com o novo estado $TabCustRaizes(G_i)^{k+1}$ de EB_i, com $1 \leq i \leq N$, é iniciado o procedimento para iniciar a atualização do seu grafo interno SGj, com os dados de $TabCustRaizes(G_i)^{k+1}$. Assim que todos os conjuntos $TabCustRaizes(G_i)^{k+1}$ para o estado $k+1$ são recebidos por SB_j, é realizada a atualização do conjunto de árvores-modelo de $SB(j)$ (apresentadas na seção 3.3). Esta etapa consiste em executar o algoritmo dinâmico ou o estático, para que cada uma das árvores-modelo seja atualizada com os novos valores (estado $k+1$) recebidos das estações EB_i. Assim que as árvores tenham sido atualizadas, é enviada uma mensagem *Received_Rwset(k+1)* para cada estação EB_i, confirmando o recebimento dos conjuntos $TabCustRaizes(G_i)^{k+1}$, do estado $k+1$.

Após todos os servidores de busca terem enviado a mensagem *Received_Rwset(k+1)*, para a estação EB_i, é enviada uma mensagem *Received_Rset(k+1)* para o coordenador, EB_C. Depois que cada estação EB_i, com $1 \leq i \leq N$ tenha enviado a mensagem *Received_Rset(k+1)* para EB_C, EB_C envia uma mensagem (*Commit_RSet(k+1)*) para que cada estação EB_i passe a utilizar efetivamente o seu novo estado. A mensagem *Commit_RSet(k+1)* também é enviada para os servidores de busca, para que os mesmos utilizem os dados do novo estado nas próximas buscas.

A figura 5 apresenta o procedimento de atualização de 3 estações. EB_1 foi escolhida para ser a estação coordenadora EB_C. Supõe-se que o estado global do sistema é k e EB_C inicia a transação para atualizar o sistema para o estado $k+1$. Para isso, inicialmente EB_C envia uma mensagem *Update_RSet(k+1)* para EB_1 e EB_2. Após as duas estações receberem a mensagem de atualização, EB_1, EB_2 e EB_C executam a etapa (I), a qual consiste em atualizar as suas árvores pórticos e enviar os novos conjuntos $TabCustRaizes(G_i)^{k+1}$ para os servidores de busca. Esta etapa não foi representada para não sobrecarregar a figura. Assim que todos os servidores de busca responderem que receberam os novos conjuntos $TabCustRaizes_i^{k+1}$ para

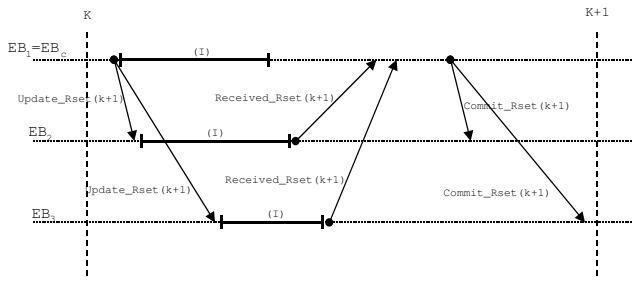


Figura 5. Processo de atualização do sistema para o estado $k+1$.

a estação EB_i, a estação EB_i envia a mensagem *Received_RSet(k+1)* para EB_C[3]. Quando todas estações enviarem a mensagem *Received_RSet(k+1)* para EB_C (incluindo a própria estação EB_C), EB_C envia a mensagem *Commit_RSet(k+1)* para todas estações e servidores de busca, os quais fazem o estado $k+1$ tornar-se o estado atual.

4 Alguns Resultados

Nesta seção são mostrados alguns resultados sobre três aspectos abordados neste trabalho. Os testes foram realizados com os equipamentos disponíveis no Laboratório de Computação Paralela e Distribuída[4], em máquinas Pentium II com 400 MHz e 128 MB de memória RAM, em sistema operacional Linux. A versão da linguagem Java utilizada foi o pacote JDK 1.3.1[5] e o pacote CORBA usado é o JacORB 1.4.1[6]. A implementação do sistema em linguagem Java utilizando CORBA pode ser encontrada em `http://www.ime.usp.br/~mlv/distgraph/index.html`.

Foram realizados diversos testes, entretanto para este texto são mostrados resultados médios variando-se os principais parâmetros, dentre eles, o Conjunto Raízes considerado, o número de nós e também o comportamento do sistema variando-se o número de clientes ativos. Deve-se ressaltar que os testes comparativos foram realizados com os mesmos dados.

Para os testes apresentados, foi considerada a utilização do algoritmo estático de Dijkstra utilizando filas de prioridades e a versão dinâmica do primeiro método incremental (primeiro Dijkstra incremental), conforme apresentado na seção 2.

[3] Deve-se ressaltar que os servidores de busca somente enviam as mensagens de recebimento dos conjuntos $TabCustRaizes(G_i)^{k+1}$ após terem atualizado as suas árvores-modelo.

[4] http://www.lcpd.ime.usp.br/

[5] http://www.javasoft.com/

[6] http://www.jacorb.org

Um dos parâmetros mais importantes do sistema apresentado neste trabalho é o Conjunto Raízes. Dependendo do conjunto sendo considerado, o sistema pode ter um desempenho melhor ou pior. Existem um compromisso claro entre o desempenho e a qualidade da solução. Por isso, foram realizados vários testes em grafos particionados de diversas maneiras. Para os testes de verificação da influência do Conjunto Raízes no sistema, foram considerados grafos contendo aproximadamente 10000 nós, com grau 4. Os gráficos foram particionados de diversas formas e armazenados em uma quantidade variando de 1 a 5 estações-base, utilizando-se 1 servidor de busca. Em geral, o comprimento atribuído às bordas das partições geradas nos testes variaram entre 250 a 500 nós. Para as bordas fixadas, realizaram-se testes variando-se o Conjunto Raízes entre 25% até 100% da borda considerada. Além disso, os testes efetuados utilizaram taxas médias de 50% de modificações de pesos dos arcos a cada rodada de atualização, com pesos gerados aleatoriamente dentro de uma faixa definida.

Os gráficos 6.a, 6.b, 6.c e 6.d apresentam os tempos para a atualização de estados para servidores de busca utilizando o algoritmo estático (SB-E), servidores de busca utilizando o algoritmo dinâmico (SB-D), estações-base utilizando o algoritmo estático (EB-E), estações-base utilizando o algoritmo dinâmico (EB-D), respectivamente, para grafos com aproximadamente 10000 particionados em 2, 3, 4 e 5 estações-base.

O gráfico 7.a apresenta resultados para buscas de caminhos locais para pares de vértices aleatórios (em geral, exigindo consulta ao servidor de busca), para grafos particionados em 1, 2, 3, 4 e 5 estações-base, variando-se o Conjunto Raizes.

Conforme mostrado pelos gráficos, o uso de algoritmos dinâmicos na atualização de servidores de busca apresentou um desempenho comparativo melhor do que em estações-base. Isso ocorre pois os grafos armazenados em servidores de busca são, em geral, menores em número de nós e com grau elevado. Esse comportamento foi descrito pelos autores do arcabouço proposto por Narváez et al. [5], onde são apresentados testes que mostram que à medida que o grau aumenta, a complexidade dos algoritmos diminui. Maiores detalhes estão na seção 9.5 de [5]. Também pode ser visto que o comprimento dos Conjuntos Raízes influencia bastante os tempos de buscas e atualizações. Entretanto, deve-se lembrar que o uso de partições com Conjuntos Raízes menores pode modificar a qualidade da solução obtida. Testes realizados nesse sentido demonstraram que o erro médio da qualidade da solução variou entre 10 a 25%. O ideal é formar partições que gerem **bordas** de comprimentos menores e considerar Conjuntos Raízes de comprimento próximos ao comprimento das bordas. Entretanto, políticas de interpolação vem sendo avaliadas para melhorar a solução de Conjuntos Raízes com comprimentos infe-

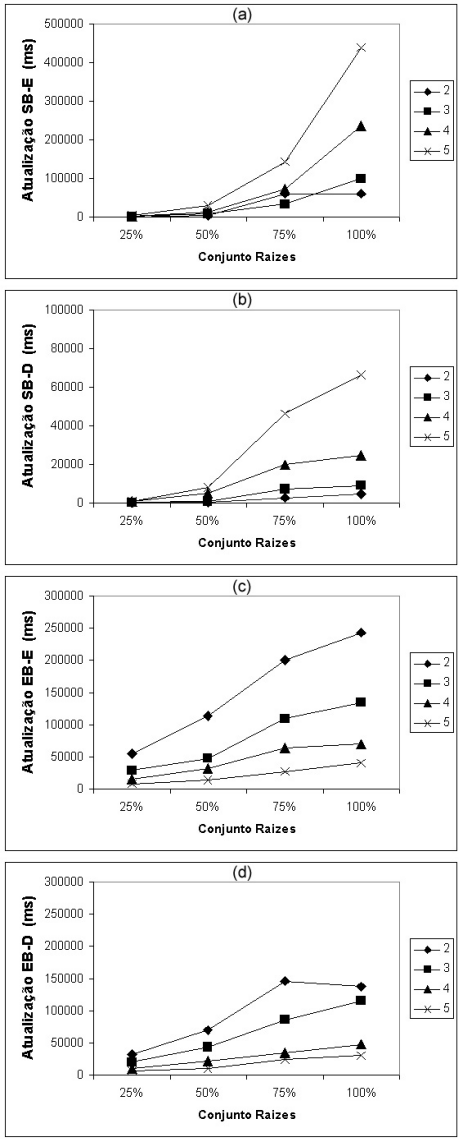


Figura 6. Tempos de atualização em função do Conjunto Raízes

riores às bordas.

Também foram efetuados testes para avaliar o desempenho dos procedimentos de busca em função do número de clientes ativos. Para tanto, foram observados os comportamentos de sistemas variando-se o número de estações-base e também o número de servidores de busca. Os testes efetuados utilizaram grafos contendo aproximadamente 10000 nós com grau 4, em média. A taxa de atualização de arcos foi de 50% em média e foram geradas partições de forma que os subgrafos apresentassem bordas de 100 a 300 nós, considerando-se Conjuntos Raízes de 70%. Com esse ambiente foram realizadas buscas locais aleatórias, como as buscas anteriores. O gráfico 8.a apresenta os resultados de tempos médios de buscas em sistemas utilizando-se um servidor de busca e 1, 2 ou 3 estações-base. Para os testes, definiu-se que 5000 ms seria o máximo tempo de resposta médio aceitável. O gráfico 8.a mostra que sistemas centralizados apresentam um limite não muito alto para o número de clientes ativos: 573 clientes, em média (mostrado na linha pontilhada do gráfico 8.a).

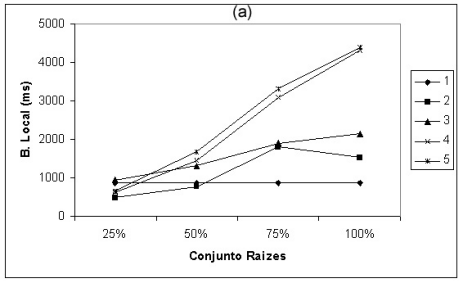


Figura 7. Tempos de buscas em função do Conjunto Raízes

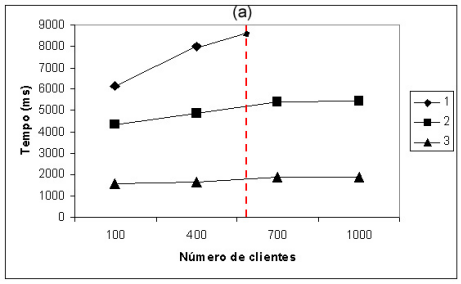


Figura 8. Número de clientes ativos em função do número de servidores

5 Conclusão

Neste artigo apresentamos um sistema completo e escalável para o armazenamento de grafos distribuídos em diversos servidores. Esses grafos armazenados podem ou não sofrer atualizações de estado. O trabalho também permite que buscas sejam realizadas, considerando-se algoritmos conhecidos.

Mostramos através de uma implementação que o sistema permite tanto a atualização dinâmica dos grafos, como a realização de consultas eficientes sobre estimativas de caminho. Foram realizados testes com versões de algoritmos estáticos e dinâmicos para cálculo de caminhos.

Como trabalhos futuros podemos estudar formas de particionamento dos grafos nos diversos servidores e a implementação de outros algoritmos para grafos dinâmicos.

Referências

[1] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2nd. edition, 2001.

[2] J. Fawcett and P. Robinson. Adaptive routing for road traffic. *IEEE Computer Graphics and Applications*, 20(3):46–53, May/June 2000.

[3] D. Frigioni, A. Marchetti-Spaccamela, and U. Nanni. Incremental algorithms for single-source shortest path trees. *Proceedings of Foundations of Software Technology and Theoretical Computer Science*, pages 113–124, December 1994.

[4] V. Garg. *Elements of Distributed Computing*. John Wiley and Sons, 1st. edition, 2002.

[5] P. Narváez, K. Siu, and H. Tzeng. New dynamic algorithms for shortest path tree computation. *IEEE/ACM Transactions on Networking*, 8(6):734–746, December 2000.

[6] G. Ramalingam and T. Reps. An incremental algorithm for a generalization of the shortest path problem. *Journal of Algorithms*, 21:267–305, 1996.

[7] M. Schwartz. *Mobile Wireless Communications*. Cambridge University Press, 1st. edition, 2005.