Proxy Ativo: Diminuindo a Latência através de Pushing

Bruno Gusmão Rocha, Adriano Alonso Veloso
Wagner Meira Jr., Virgilio A. F. Almeida

Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
Av. Antônio Carlos, 6627 - Pampulha - Belo Horizonte - MG - Brasil

Abstract

The majority of the Internet users still access the Web via slow modem connections. Studies have shown that the limited modem bandwidth is the main contributor to the latency perceived by users. In this paper we introduce one approach to reduce the latency by *pushing* between proxies and its clients, in special low-bandwidth clients. The approach takes advantage of idle periods between requests to pull proxy-cached objects which will be probably accessed next by the client. We use data mining concepts to develop a efficient prediction model and evaluate the potential of this technique at reducing client latency throught simulations based on real workloads from a major proxy in Minas Gerais.

Resumo

A maioria dos usuários da Internet ainda acessa a rede através de lentas conexões via modem. Estudos mostraram que a limitada banda dessas conexões é uma das principais causas da latência percebida pelos usuários. Apresentamos neste artigo uma forma de se reduzir a latência através de *pushing* entre proxies e seus clientes, com destaque para aqueles que possuem uma banda estreita. A abordagem se aproveita do período ocioso entre requisições de um cliente para enviar objetos cacheados no proxy que provavelmente serão acessados em seguida. Para isso, nós usamos conceitos de mineração de dados com a finalidade de desenvolver um modelo de previsão eficiente, e avaliamos o potencial dessa técnica através de simulações sobre uma carga real proveniente de um grande proxy do estado de Minas Gerais.

Palavras-chave— ***Pushing*, *Proxy*, Latência, Mineração de Dados, Banda Estreita**

1 Introdução

Um tempo de resposta rápido é fundamental para garantir a satisfação e a qualidade de serviços fornecidos aos usuários da Internet. Porém, o problema da latência se manifestou de tal maneira que a World Wide Web ganhou o irônico apelido de "World Wide Wait".

A latência experimentada pelos usuários pode ter causas diversas, como a demora nos servidores para processar as requisições (especialmente se estes estiverem sobrecarregados), congestionamentos na rede, o tempo de propagação devido à distância física entre clientes e servidores, ou ainda ineficiências no protocolo HTTP, como observado em [1]. Para a maioria dos usuários que ainda acessa a Web através de conexões via modem entretanto, a pequena banda disponível é a principal responsável no aumento da latência. Estudos mostraram que mesmo se o cache em *proxy* fosse empregado com o melhor desempenho possível, a latência nos clientes se reduziria apenas de 3% a 4% devido às conexões via modem [2]. Entretanto, a utilização de cache tanto no *proxy* quanto no navegador do cliente ainda são algumas das mais efetivas técnicas para se reduzir a latência.

A idéia por trás dessas técnicas é simples: sempre que um usuário acessa um recurso Web, ele verifica se este já está disponível em seu cache local. Se a resposta for afirmativa, o usuário recebe uma cópia local do recurso, evitando uma conexão lenta com o *proxy*, e reduzindo o tráfego na rede. Caso contrário, a requisição é feita ao proxy, que por sua vez verifica se ele possui o documento em seu cache. Se o documento estiver cacheado, ele é enviado ao usuário, e se não estiver, a requisição é repassada ao servidor remoto ao qual ela se destinava originalmente.

A eficiência desse mecanismo pode ser aumentada através da utilização de técnicas de *prefetching* e *pushing* [13], as quais buscam prever qual recurso será requisitado no futuro, de forma que este possa ser transferido antes mesmo de ser requisitado. O *proxy* pode ser ativo ou passivo na decisão de quando e qual recurso será transferido. Caso ele seja ativo, ou seja, é o *proxy* e não o cliente quem decide sobre a transferência, a técnica é chamada de *pushing*. Em caso contrário, dá-se o nome de *prefetching*. O *proxy* está em excelente posição de fazer especulações sobre futuras referências, uma vez que atende a numerosas requisições de uma população específica de clientes, que apresentam um padrão de navegação mais constante. O *pushing* se coloca então como uma técnica muito interessante para a redução da latência, e é sobre esse cenário que trabalharemos no decorrer deste artigo.

A qualidade do *pushing* está intrinsicamente ligada à quantidade de banda disponível, ao tamanho do cache, e ao poder de previsão do método que deverá predizer as futuras requisições do cliente. Apesar da banda curta oferecida por conexões via modem, a existência de banda disponível para a realização do *pushing* é garantida pelo fato de sempre haver períodos ociosos entre requisições do cliente , no qual ele está utilizando os recursos acessados. Neste artigo, apresentamos e analisamos uma técnica capaz de reduzir a latência em clientes que utilizam conexões lentas via modem através de *pushing* a partir do *proxy*. Para isso, nós estudamos uma técnica de Mineração de Dados com alto poder de previsão, chamada regra de associação temporal. Tal técnica leva em consideração a correlação temporal entre objetos para gerar implicações, ou regras, com alto poder de previsão. De posse dessas regras, o *proxy* pode ainda levar em consideração características relevantes sobre as

requisições que ele recebe dos clientes, como a popularidade dos objetos, a quantidade de acessos a cada recurso, as características da população, e a relevância do objeto sugerido para o pushing. O restante do artigo está organizado da seguinte forma. Na Seção 2 descrevemos a arquitetura de um *proxy* ativo de um sistema baseado em *pushing*. Na Seção 3 apresentamos a técnica de Mineração de Dados que foi adotada para determinar os objetos que serão utilizados pelo *pushing*. Na Seção 4 apresentamos o algoritmo de predição que foi incorporado ao *proxy*. Na Seção 5 caracterizamos a carga de trabalho utilizada, e apresentamos os principais resultados obtidos nos experimentos. Finalmente, na Seção 6 concluímos nosso trabalho, e indicamos direções para trabalhos futuros.

2 Proxy Ativo

A função básica de um *proxy* é interceptar todas as requisições entre um conjunto de clientes e os servidores reais aos quais elas são destinadas, verificando se ele mesmo pode responder essas requisições e repassando-as aos servidores em caso negativo. O *proxy* ativo é uma extensão do *proxy*, cujo papel é reunir informações sobre o padrão de navegação dos usuários, prever quais objetos serão mais provavelmente acessados em seguida e inicializar o *pushing* desses objetos para os clientes. *Pushing* é o envio de objetos aos clientes sem que estes tenham sido requisitados, com intuito de diminuir a latência e o desperdício de banda dos usuários. Algumas questões se tornam importantes quando a utilização dessa técnica é considerada.

A primeira delas é como enviar de forma eficiente os objetos do *pushing*. Várias soluções foram propostas, com destaque para a apresentada em [4, 5], que envia os recursos previstos para os clientes anexados em mensagens comuns de resposta, se aproveitando das conexões persistente providas pelo protocolo HTTP 1.1 e evitando o estabelecimento de novas conexões TCP. Tal mecanismo foi de fato implementado em [4, 5] e parece o mais apropriado por requerer relativamente poucos melhoramentos no protocolo atual de requisição-resposta e nenhuma mudança no protocolo HTTP 1.1. As outras questões relevantes à eficácia do *pushing* referem-se a que objetos enviar aos clientes e quando fazê-lo. Descreveremos agora a arquitetura do *proxy* ativo, e como esses questões foram respondidas.

A Figura 1 mostra a arquitetura proposta para o *proxy*. O cliente faz uma requisição ao *proxy*, que além de tratá-la normalmente, a repassa ao *proxy* ativo (identificado pela linha tracejada na Figura 1). O *proxy* ativo mantém uma lista contendo os objetos requisitados para cada sessão ativa [1], que representa o caminhamento do cliente durante a sessão. O objeto requisitado é então acrescentado a lista daquele cliente no *proxy* ativo, e essa lista é passada ao algoritmo de previsão. A função do algoritmo de previsão é retornar os objetos com mais chance de serem acessados pelo cliente em seguida, baseando-se na comparação entre o caminhamento do cliente nessa sessão e padrões previamente observados de todos os outros clientes em um determinado período. Assim, o algoritmo de previsão retorna uma lista de objetos candidatos, ordenados pela probabilidade de acesso. O *proxy* ativo

[1] As sessões são identificadas pelo IP das requisições e pelo intervalo entre elas. Uma sessão é considerada terminada se não houver nenhuma requisição no intervalo de 30 minutos

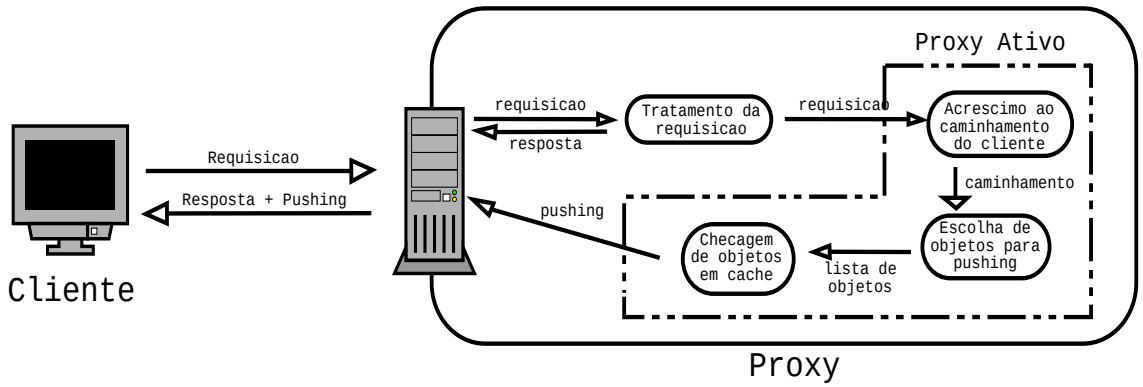

Figura 1: Arquitetura do *proxy*

verifica quais desses objetos estão no cache do *proxy*, ou seja, quais desses objetos ele pode efetivamente servir, e quais deles não estão no cache do navegador do cliente, para evitar transferências desnecessárias. Após os objetos que não obedeçam esses requisitos serem eliminados da lista de candidatos, tem-se o conjunto de objetos utilizados para o *pushing*. Finalmente, o *proxy* envia a resposta ao cliente e escalona a transferência dos objetos do *pushing* um a um para imediatamente após o momento em que a resposta da requisição corrente seja totalmente recebida pelo cliente. Se o usuário envia uma nova requisição, o *pushing* é interrompido e qualquer objeto parcialmente transferido é descartado do cache do cliente, a menos que seja justamente esse o objeto requisitado. A lista de objetos candidatos é então esvaziada, uma nova lista é calculada pelo *proxy* ativo, e o processo se reinicia. Dessa maneira, quando um cliente faz uma requisição ao *proxy*, temos 3 possibilidades: o objeto requisitado está no cache do navegador, está sendo enviado pelo *pushing* ou deve ser buscado normalmente no *proxy*.

Como pode ser observado, o cache do navegador está sendo utilizado tanto para o cache em si quanto como um repositório de objetos enviados por previsões. Existem vários artigos que analisam o compromisso entre o espaço destinado ao pushing e ao cache [6, 7]. Para uma melhor eficácia da predição, o navegador deve também mandar um comunicado ao *proxy* de que houve um hit no cache. Esse comunicado pode ser enviado anexado à requisição 1, de forma a não se gerar tráfego extra de rede.

O principal objetivo do *pushing* através do *proxy* ativo é reduzir a latência observada pelos usuários. Essa redução é conseguida basicamente através do aumento dos hits no cache do navegador, e portanto outro importante fator é a acuidade das previsões realizadas pelo *proxy*. A porcentagem dos hits no cache conseguidos com o *pushing*, pode nos informar de forma fidedigna o quão eficiente é o método de previsão. Finalmente, a banda desperdiçada com a transferência de objetos não utilizados pelo cliente tem pouco impacto, visto que essa banda não seria utilizada de qualquer forma devido aos períodos ociosos existentes na conexão do modem entre requisições de recursos. Esse período se deve ao fato do cliente interagir com um recurso (como ler uma página, por exemplo) antes de requisitar o próximo.

3 Determinação dos Objetos para *Pushing*

Um método eficiente para descobrir qual o próximo objeto a ser requisitado pelo usuário é de fundamental importância para o bom funcionamento de uma técnica de *pushing*. A técnica empregada em nossa abordagem é baseada em um conceito bem fundamentado de Mineração de Dados, chamado Regras de Associação Temporal.

A tarefa de gerar regras de associação temporal consiste em encontrar padrões de acesso que freqüentemente são realizados pelos usuários, e extrair regras capazes de indicar como a ocorrência de um determinado padrão de acesso influencia na ocorrência de outro padrão. Um padrão de acesso α é denotado como $(\alpha_1 \mapsto \alpha_2 \mapsto ... \mapsto \alpha_q)$, onde o elemento α_j é um recurso que foi requisitado pelo usuário. Sendo assim, um padrão de acesso é uma lista ordenada de requisições feitas por um usuário. Definimos o *suporte* de um padrão de acesso X, como $\sigma(X)$, e seu valor é dado pela porcentagem dos usuários que possuem tal padrão de acesso. Caso o suporte de um padrão de acesso seja maior ou igual ao suporte mínimo especificado pelo usuário, esse padrão é chamado de freqüente. Nesse contexto, podemos definir uma *sessão* como sendo um conjunto de padrões de acesso de um determinado usuário.

Uma regra de associação temporal é uma implicação da forma $X \Rightarrow Y$, onde X e Y são padrões de acesso freqüentes. X é chamado de *antecedente* e Y é chamado de *conseqüente*. O suporte da regra é dado por $\sigma(X \cup Y)$, ou seja, a probabilidade conjunta de que um usuário possua X e Y como padrões de acesso, e sua *confiança* é dada por $\sigma(X \cup Y)/\sigma(X)$, ou seja, a probabilidade condicional que o usuário possua Y dado que ele possui X. Uma regra é confiável se sua confiança é maior ou igual a confiança mínima especificada pelo usuário.

A geração de regras de associação temporal geralmente é dividida em duas etapas distintas. O objetivo da primeira etapa é encontrar todos os padrões de acesso freqüentes, enquanto o objetivo da segunda etapa é gerar regras que possuam alta confiança. Regras de associação temporal são capazes de oferecer implicações com alto poder de previsão porque levam em consideração a correlação temporal existente entre os objetos requisitados por um usuário. Além disso, sua definição representa de forma precisa o cenário composto pelos padrões de acesso de um usuário da Web, uma vez que tais padrões estão intrinsicamente associados à idéia de sequência temporal.

Para a geração de regras de associação temporais foi utilizado o algoritmo SPADE, apresentado em [8], que divide o espaço de busca por padrões de acesso freqüentes em sub-espaços de busca totalmente independentes e apresenta um tempo de execução muito menor que o de outros algoritmos desenvolvidos para a mesma função [9, 10]. O algoritmo foi modificado porém, para gerar apenas as regras que tenham como conseqüente padrões de tamanho unitário, pois para efeito de *pushing* estamos interessados apenas no objeto seguinte a sequência que o antecedeu. Desta maneira, para a sequência $A \rightarrow B \rightarrow C$ seriam geradas as regras $A \Rightarrow B$, $A \Rightarrow C$ e $A \rightarrow B \Rightarrow C$, mas não a regra $A \Rightarrow B \rightarrow C$ (supondo obviamente que todas essas regras são freqüentes e confiáveis).

Considere os padrões de acesso mostrados na Figura 2 (usada como exemplo por todo este artigo). A figura registra dezesseis acessos para 4 objetos (A até D), provenientes de

Identificador	Time-Stamp	Objeto
1	100	A
1	102	D
1	105	A
2	103	C
2	117	D
2	122	B
3	101	D
3	105	C
3	108	B
4	109	A
4	114	C
4	120	D
4	127	B
5	107	D
5	111	B
5	112	C

Suporte	Regras
0.8	D -> B
0.6	C -> B
0.4	A -> D C -> D CD -> B D -> C
0.2	A -> A A -> B A -> C AC -> D AC -> B AD -> B AD -> A B -> C D -> A DC -> B DB -> C

Figura 2: Padrões de Acesso

cinco usuários distintos. A Figura 2 também mostra todas os padrões de acesso freqüentes com seu respectivo suporte.

3.1 Seleção das Regras de Associação Temporal Relevantes

Um dos maiores problemas da estratégia baseada em regras é o enorme número de implicações que são geradas. Os dois maiores inconvenientes advindos desse grande número de regras são:

- *Regras redundantes.* Sejam duas regras $R_1 : X_1 \rightarrow Y_1$ e $R_2 : X_2 \rightarrow Y_2$. R_1 e R_2 são redundantes se e somente se $X_1 \subset X_2$ ou $X_1 \supset X_2$ e $Y_1 = Y_2$. Por exemplo:

  $R_1 : B \Rightarrow C$, que significa que se o último objeto requisitado foi B, o próximo será C.

  $R_2 : A \rightarrow B \Rightarrow C$, que significa que se os últimos objetos requisitados foram A e B, o próximo será C.

  Nesse caso R_1 é mais geral que R_2, pois aquela pode ser aplicada em todos os casos onde esta pode. Assim podemos remover R_2 sem prejuízo ao poder de predição do modelo.

- *Regras contraditórias.* Sejam duas regras $R_1 : X_1 \Rightarrow Y_1$ e $R_2 : X_2 \Rightarrow Y_2$. A função $\mu(X)$ denota o último item do padrão de acesso X. R_1 e R_2 são contraditórias se e somente se $\mu(X_1) = \mu(X_2)$ e $Y_1 \neq Y_2$. Por exemplo:

$R_1 : B \Rightarrow C$, que significa que se o último objeto requisitado foi B, o próximo será C.

$R_2 : B \Rightarrow D$, que significa que se o último objeto requisitado foi B, o próximo será D.

$R_3 : A \rightarrow B \Rightarrow E$, que significa que se os últimos objetos requisitados foram A e B, o próximo será E.

Aqui temos três regras que sugerem escolhas diferentes de objetos para o *pushing* para o mesmo último objeto requisitado. É fundamental para o modelo escolher dentre as regras aplicáveis aquela que indique o objeto com maior probabilidade de ser o próximo acessado pelo usuário.

Definir quais regras usar e quais descartar são problemas de fundamental importância para a eficiência do *pushing*. Definimos assim o conceito de *relevância*, que é um número que representa o quão aplicável é aquela regra. Quanto maior o valor da relevância, melhor a regra. A seguir discutiremos os fatores que definem a relevância de uma regra.

3.1.1 Suporte

Um dos critérios para se diferenciar as regras é o seu suporte. Quanto maior o suporte de uma regra, maior a quantidade de usuários representados por ela. É esperado que regras mais freqüentes sejam mais usadas no futuro que regras menos freqüentes. Na figura 2 por exemplo, temos que a regra $D \Rightarrow B$ foi contemplada por quatro dos cinco usuários, enquanto a regra $D \Rightarrow A$ foi contemplada por apenas um usuário. O que o suporte nos diz é que, probabilisticamente, o objeto B é o que teria mais chances de se seguir ao objeto D, e por isso ele seria um bom candidato para o *pushing*.

3.1.2 Confiança

Outro fator a ser considerado é a confiança. Regras que possuem confiança alta apresentam uma chance maior de que seu conseqüente seja verdadeiro caso seu antecedente o seja, isto é, apontam para casos em que o objeto indicado para o *pushing* ocorreu mais vezes em conjunto com o padrão de acesso descrito pelo antecedente da regra. Se tomarmos como exemplo as regras $C \Rightarrow B$ e $C \Rightarrow D$ na figura 2, temos que a confiança da primeira é $c_1 = \sigma(C \cup B)/\sigma(C) = 0.6/0.8 = 0.75$ enquanto a confiança da segunda é $c_2 = \sigma(C \cup D)/\sigma(C) = 0.4/0.8 = 0.50$. Com isso, se tomarmos apenas os usuários que requisitaram C, observaremos que B é mais visto que D. Mais formalmente, a probabilidade condicionada de termos B dado que temos C é maior quanto maior for a confiança, e daí é maior a probabilidade que B seja o próximo documento a ser requisitado pelo usuário.

3.1.3 Tamanho da Regra

Como mencionado anteriormente, a sessão de um cliente é formada por uma série de requisições a um conjunto de objetos. O antecedente de uma regra captura justamente a

parte final dessa seqüência, a partir do documento mais recentemente acessado. A regra $D \Rightarrow C$ na figura 2 mostra que quando último objeto acessado é D, C será provavelmente o próximo a ser requisitado. Já a regra $CD \Rightarrow B$ diz que quando o cliente pediu D e antes disso já havia solicitado C, B terá mais chances de ser o objeto desejado em seguida. Em outras palavras, quanto mais dados temos sobre o caminhamento dos usuários, mais subsídios temos para fazer previsões acuradas. O tamanho da regra nesse sentido, reflete a quantidade de informação acumulada sobre os padrões de acesso dos usuários.

3.1.4 Popularidade dos Objetos

A semântica das regras de associação está intrinsicamente ligada à aplicação a qual se destinam. No presente caso, as regras são geradas sobre objetos em proxy que se comportam de uma maneira específica e cuja população possui características peculiares. Podemos conseguir resultados consideravelmente melhores se tirarmos partido das propriedades específicas dos proxies.

Vários estudos observaram que a freqüência relativa com a qual documentos são requisitados no proxy seguem a mesma distribuição que a observada em certos fenômenos que obedecem a lei de Zipf [11, 14]. A lei de Zipf é estendida de forma a afirmar que a probabilidade relativa de uma requisição para o i-ésimo objeto mais popular é proporcional a $1/i^{\alpha}$ com α assumindo valores tipicamente menores que a unidade.

Dessa forma, a precisão da escolha de um objeto dentre aqueles sugeridos por regras contraditórias pode ser aumentada se levarmos em conta o *ranking* de cada um deles. Assim conseguimos aumentar a qualidade de nossas previsões incorporando ao modelo um comportamento típico de objetos em proxy.

4 Algoritmo de Predição

O algoritmo de predição é uma parte fundamental do processo, visto que é ele quem vai determinar a eficiência e a própria viabilidade do *pushing*. O algoritmo aqui utilizado baseia-se nos conceitos discutidos na seção anterior.

Na Seção 2, foi explicada a arquitetura do *proxy* ativo. Aqui detalharemos o modo como o algoritmo de predição seleciona dentre as regras disponíveis aquela que possui a maior probabilidade de sucesso, e como regras redundantes e contraditórias são eliminadas do processo de escolha.

4.1 Cálculo de Relevância

O primeiro passo do algoritmo de predição é ordenar as regras por sua relevância. Como visto na Seção 3, há diversos fatores que podem aumentar a eficiência das regras de associação em *proxy*. Seja a regra $R : X \rightarrow Y$ com suporte s, confiança c, tamanho $|R|$ e que a probabilidade de acesso de Y segundo a lei de Zipf seja p. A relevância r dessa regra pode

ser dada pela combinação de todos esses fatores:

$$r = sc|R|p = \frac{\sigma(X \cup Y)^2|R|}{\sigma(X)i^\alpha}$$

Cumpre observar contudo que dois fatores estão exercendo um peso desproporcional aos demais no cálculo. O primeiro deles é o tamanho. O suporte e a confiança são dados por números entre zero e um, mas o tamanho de uma regra é sempre um número maior que dois, já que toda regra deve ter, no mínimo, um antecedente e um consequente. Podemos contornar esse problema normalizando o tamanho. Assim, esse fator passa a corresponder a $|R|/max(|R|)$, onde $max(|R|)$ é o tamanho da maior regra gerada.

A probabilidade dada por Zipf também mostra um peso superior ao desejado, por ter um caráter de decaimento exponencial. Isso pode levar sempre à eleição de regras que apontam para objetos mais populares, fazendo com que o conhecimento advindo dos outros fatores se esvaia no papel coadjuvante para o qual estes são empurrados. Dessa maneira, utilizamos o valor dado por $log(1/i^\alpha)$ no cálculo da relevância, que passa a ser:

$$r = \frac{\sigma(X \cup Y)^2|R|}{\sigma(X)max(|R|)} log\left(\frac{1}{1^\alpha}\right)$$

4.2 Corte de Regras

O número de regras geradas cresce exponencialmente com a diminuição do suporte mínimo definido pelo usuário, que além dos já citados problemas de redundância e inconsistência pode levar ao comprometimento do desempenho do algoritmo e a uma alta utilização de memória. Usamos aqui uma versão modificada da LSI-Tree apresentada em [12] para efetuar o corte de regras e aumentar a eficiência da seleção.

A LSI-Tree é uma árvore onde cada nó contém um item de uma regra. Os filhos da raiz representam o último objeto acessado, seus filhos representam os objetos acessados antes dele, e assim sucessivamente. Uma regra é identificada através do encaminhamento da raiz até um dos nós. Se essa regra realmente existe, então existe um objeto associado a este nó que indica o documento que será o alvo do *pushing*. Caso contrário, essa sequência é apenas uma subsequência dentro de uma regra maior. Regras redundantes não são inseridas na LSI-Tree e insere-se dentre as regras contraditórias aquela com a maior relevância.

Na nossa implementação, se um nó identifica uma regra, pode ter associado a ele mais de um objeto. Em outras palavras, regras contraditórias com relevâncias próximas e que representam exatamente o mesmo padrão de navegação coexistem na árvore, com os documentos sugeridos ordenados pela sua relevância.

Considere as regras mostradas na tabela 1. As regras 1 e 2 são inseridas normalmente na árvore. A regra 3 possui uma relevância maior que a regra 2, no entanto ela é redundante, já que pode ser sempre substituída pela regra 2 em todos os casos em que se aplica. Daí ela é cortada da árvore. A regra 4 não é redundante, entretanto sua relevância é bem menor que a da regra 2, e ela também é desconsiderada. A regra 5 possui relevância maior que a regra 2 e sugere um documento diferente para o *pushing*, portanto ela é inserida com a

	Regras	Relevância
1	$A \Rightarrow D$	0.32
2	$B \Rightarrow C$	0.67
3	$A \rightarrow B \Rightarrow C$	0.71
4	$A \rightarrow B \Rightarrow D$	0.17
5	$C \rightarrow A \rightarrow B \Rightarrow D$	0.78
6	$C \rightarrow A \rightarrow B \Rightarrow E$	0.75

Tabela 1: Entrada da LSI-Tree

criação dos nós C e A, sendo o nó A filho do já existente nó B. A regra 6 é contraditória com a regra 5, porém suas relevâncias são muito semelhantes, o que leva a inserção do documento E na lista de objetos do nó C. A árvore resultante pode ser observada na figura 3.

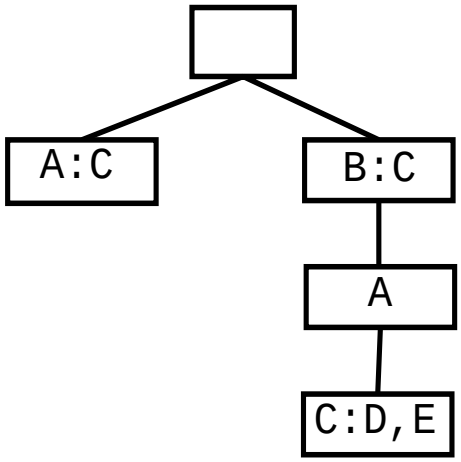

Figura 3: LSI-Tree Modificada

Para decidir quais documentos deverão ser enviados para o cliente via *pushing*, o *proxy* mantém uma lista com as requisições feitas até então por este cliente e passa essa lista ao algoritmo de predição. O algoritmo então percorre a árvore buscando a maior regra que se iguale ao caminhamento descrito pela lista, e retorna os objetos sugeridos pela regra encontrada. Tomando como exemplo a árvore da figura 3, suponhamos que um cliente tenha acessado os objetos A, C, A e B. Após a requisição para B, o *proxy* passaria essa sequência de acessos ao algoritmo de predição, que por sua vez verificaria que o maior caminhamento na árvore possível para essa sequência é $B \rightarrow A \rightarrow C$. O algoritmo então retornaria uma lista com os documentos D e E, e o *proxy* iniciaria a transferência desses objetos para o cliente assim que este estivesse inativo.

5 Avaliação Experimental

Nesta seção apresentamos os experimentos realizados e os resultados obtidos pela abordagem proposta. Todos os experimentos foram realizados com base em uma carga de trabalho real formada a partir de uma sequência de requisições extraídas de um grande *proxy* de Minas Gerais. Tal carga de trabalho é composta de 4028724 requisições a 835263 objetos estáticos, realizadas durante um período de oito dias consecutivos. A carga de trabalho proveniente dos primeiros quatro dias foi utilizada para a geração das regras de associação temporal relevantes, enquanto a carga proveniente dos últimos quatro dias foi utilizada para execução do experimento. Antes de avaliarmos os resultados obtidos com a técnica proposta, vamos analisar as características mais importantes da carga de trabalho utilizada.

5.1 Caracterização da Carga de Trabalho

Primeiramente verificamos a distribuição de popularidade dos objetos no período analisado. Como se pode observar na figura 4(a), os objetos seguem uma distribuição aproximada de Zipf com $\alpha = 0.87$. Tal fato torna extremamente desafiadora a tarefa de diminuir a latência observada pelos clientes com a utilização de caches, já que torna-se necessário um grande aumento no tamanho do cache para alcançarmos uma pequena diminuição na latência observada.

Também foi analisado o tempo decorrido entre as requisições. A figura 4(b) mostra como este tempo está distribuído . Podemos perceber um padrão auto-similar, que se repete a cada dia. Tal padrão é uma característica inerente a cargas de trabalho provenientes da *web*. O tempo médio observado entre as requisições analisadas é 2.442 segundos.

Por fim, verificamos a relação entre o número de objetos e seu tamanho. Como pode ser visto na figura 4(c), 75% dos objetos requisitados tem o tamanho menor que 10000 bytes. Admitindo que os clientes se conectem por modens de 56K a uma velocidade média de 44K, podemos comprovar a viabilidade do *pushing*, já que 88% dos objetos podem ser transmitidos em menos de 2.442 segundos.

5.2 Simulador

O simulador é baseado em eventos, que são escalonados e direcionados ao *proxy* ou ao cliente de acordo com seu tipo. A simulação funciona da seguinte maneira. As requisições são lidas pelos clientes de um arquivo de requisições retiradas do *log* real do *proxy*. Essas requisições contém o objeto a ser acessado e o tempo em que devem ser feitas. Cada cliente então verifica o conteúdo do seu cache, e faz uma requisição ao *proxy* se o objeto não está no cache ou se ele puder estar expirado. Essa requisição é colocada na fila do escalonador, que é ordenada pelo tempo das requisições e respostas. A requisição é enviada ao *proxy* no momento apropriado e retirada da fila do escalonador. O *proxy*, que contém informações sobre todos os objetos da simulação em um *hash*, trata essa requisição e monta uma resposta contendo o objeto, o cliente ao qual se destina e a latência. A latência é

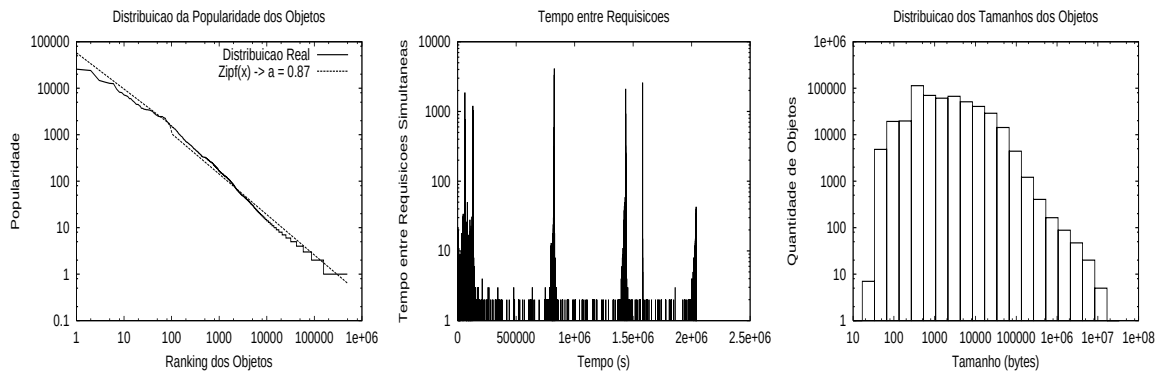

Figura 4: a) Distribuição da popularidade dos objetos, b) Tempo decorrido entre requisições consecutivas, e c) Número de objetos em relação ao tamanho

calculada dividindo-se o tamanho do objeto enviado pela largura da banda do cliente, e somando a esse resultado o tempo levado pelo *proxy* para processar essa requisição, que é uma informação extraída do *log* real. A resposta então é colocada na fila do escalonador e entregue ao cliente no tempo correspondente ao tempo em que a requisição foi feita ao *proxy* somado a latência calculada pelo mesmo. Quando a opção de *pushing* é habilitada, o *proxy* mantém uma lista com as requisições feitas por cada cliente ativo, e passa essa lista ao algoritmo de previsão exatamente como detalhado na seção 2. Nós consideramos objetos que foram transferidos por *pushing* como objetos normais do cache, que utiliza uma política LRU de substituição de objetos. A data de referência desses objetos é a data em que foram transferidos ao cache. O mecanismo funciona bem quando o método de predição é eficaz e o cache é grande.

5.3 Resultados Obtidos

Em nossos experimentos empregamos basicamente duas métricas: a taxa de acerto no cache e a latência observada pelo cliente. Para o estudo do comportamento dessas métricas, variamos dois parâmetros:

1. *O tempo destinado para o treinamento*, ou seja, o período em que as requisições foram feitas e utilizadas para a geração das regras de associação relevantes.

2. *O tamanho do cache do navegador*, dado proporcionalmente em relação a soma do tamanho de todos os objetos, ou *working-set*. A razão para tal é a necessidade de se analisar o comportamento do cache e o impacto do *pushing* de uma forma consistente com nossa carga de trabalho, evitando que o tamanho fosse definido por um número absoluto pouco significativo (como 32Mb por exemplo, um tamanho comum de cache de navegadores, mas que representaria aqui um cache infinito, já que não há requisições suficientes para ocupar esse espaço na maioria dos casos).

Um parâmetro importante para a eficiência do algoritmo de geração de regras é o suporte mínimo definido pelo usuário. Quanto maior o suporte mínimo, mais rapidamente o algoritmo executará mas menos regras serão geradas. Caso o valor do suporte mínimo seja alto demais, muitos objetos não irão gerar regras relevantes, e consequentemente, o poder de previsão do *proxy* diminuirá. Com base na observação da figura 4 (a), escolhemos o valor de 0.1%, que garante que um grande número de objetos (os mais frequentes) possuam regras significativas.

A figura 5 apresenta os resultados obtidos em nosso primeiro experimento: a latência observada pelos clientes. Definimos um valor ótimo e uma aproximação para o maior valor da latência devido a utilização do *pushing*. O valor ótimo é obtido utilizando-se *pushing ótimo*, ou seja, o *proxy* sempre sabe, com absoluta certeza e antecipadamente, qual será o próximo objeto requisitado pelo cliente. Note que esse limite não necessariamente leva a uma diminuição da latência de 100%, porque o *proxy* só inicia um *pushing* quando o cliente está ocioso. Caso o cliente faça uma nova requisição antes da anterior ter sido completamente transmitida, o *pushing* não será efetivado, embora tenha sido corretamente previsto.

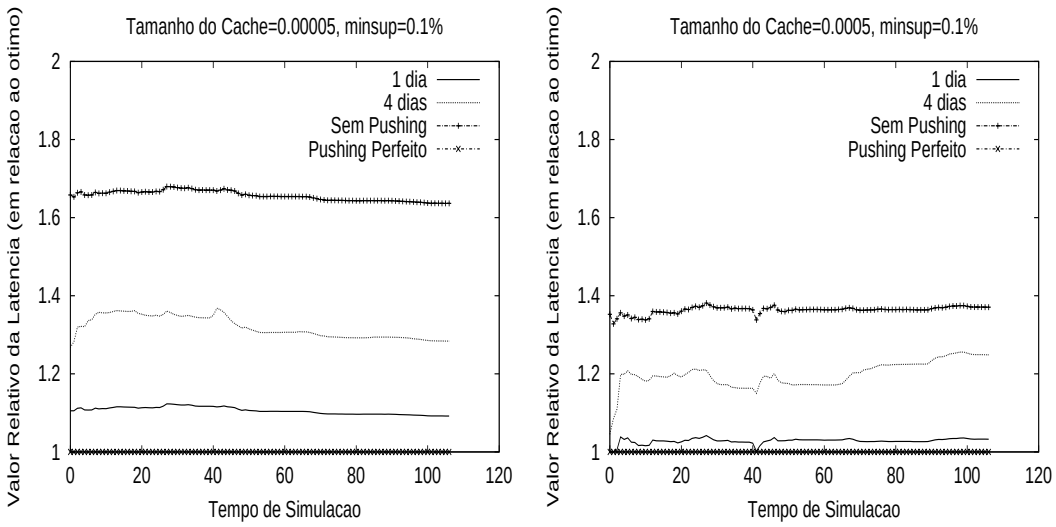

Figura 5: Diferentes valores de latência, observadas com a variação do *working-set* e do tempo de treinamento

Conseguimos claramente distinguir a relação entre os parâmetros variados no experimento e os resultados obtidos. Quanto maior o tamanho do cache empregado, menor a latência. Esse resultado já era esperado, uma vez que o aumento no tamanho do cache leva a um maior número de objetos potencialmente armazenáveis, que leva ao aumento da taxa de acerto e a diminuição da latência. Outro resultado importante é observado com a variação do tamanho da base de treinamento das regras. O melhor resultado prático obtido foi sempre aquele no qual o treinamento foi baseado no último dia, enquanto o pior foi sempre aquele no qual o treinamento foi baseado em todos os quatro dias. Esse resultado

se deve ao fato de que o poder de previsão do algoritmo ir diminuindo ao mesmo passo em que o conhecimento gerado pelas regras de associação se torna obsoleto. Quando o experimento foi configurado com o menor tamanho de cache e o maior tempo de treinamento, os resultados conseguidos foram muito próximos a nossa aproximação para o caso ótimo.

Podemos observar também que a latência aumenta com o passar das tempo, se distanciando cada vez mais do melhor caso. A causa deste fato está ligada à evolução nos padrões de acesso, que envelhecem as regras utilizadas pelo algoritmo de previsão.

O melhor resultado prático conseguido indica a eficiência de nossa abordagem, que consegue atingir taxas de acerto de aproximadamente 52%. Esse resultado quando comparado à taxa de acerto obtida com o *pushing ótimo*, indica uma eficiência de 90%.

No segundo experimento, estamos interessados em descobrir como a taxa de acerto no cache do navegador é afetada pela utilização do *pushing*. Assim como foi feito no primeiro experimento, utilizamos limites teóricos para melhor avaliar os resultados. Como a diminuição da latência é naturalmente dependente do aumento da taxa de acerto no cache, o segundo experimento apresenta uma correlação muito grande com o primeiro. Podemos observar na Figura 6 que quanto maior o tamanho do cache e menor o tempo de treinamento, maior a taxa de acerto. Novamente, o pior resultado foi verificado ao utilizarmos regras com a base de treinamento contendo requisições mais antigas e menor tamanho de cache. A taxa de acerto chegou a ser menor que a taxa observada no "pior caso". Isso se deve ao tratamento indistinto dado aos objetos no cache. Com um maior número de previsões imprecisas, ocorre um aumento no número de objetos não utilizados no cache, que por ser pequeno acaba descartando objetos úteis devido a política LRU adotada. Entretanto, o aumento na taxa de acerto do cache foi significativa, alcançando uma eficiência de até 90% em relação ao *pushing ótimo*, quando aplicada a melhor configuração.

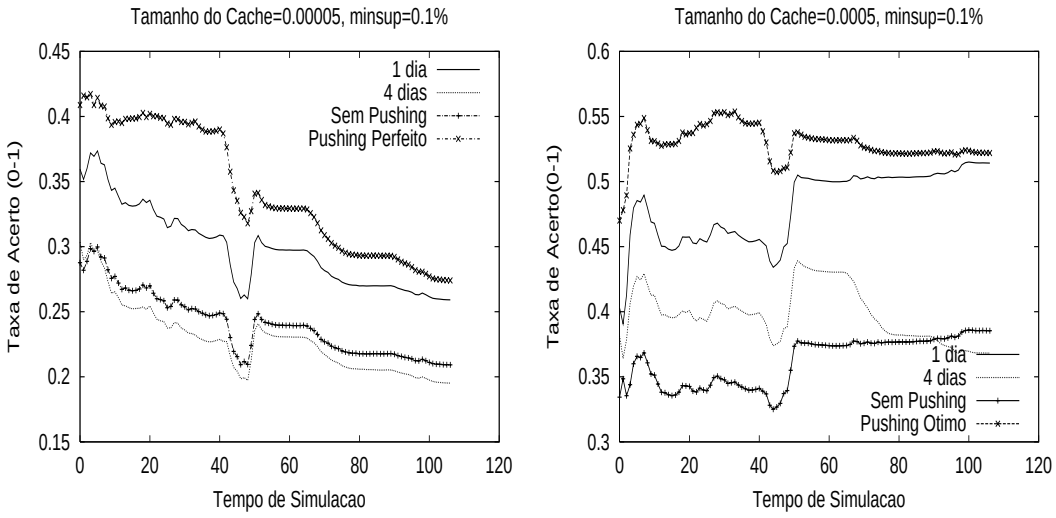

Figura 6: Diferentes taxas de hit no cache, observadas com a variação do *working-set* e do tempo de treinamento

6 Conclusão e Trabalhos Futuros

A latência ainda é um dos principais problemas enfrentados pelos usuários de conexão via modem ao acessar a Web, comprometendo não só a qualidade de serviço como também a viabilidade de várias aplicações de comércio eletrônico. Nós investigamos aqui uma técnica para diminuir a latência percebida por esses usuários através de *pushing* entre usuários de banda estreita e *proxies*, conseguindo resultados expressivos. Usamos conceitos e algoritmos da área de mineração de dados e definimos uma relevância de regra apropriada para a aumentar a eficiência do nosso algoritmo de previsão.

Há várias limitações no nosso estudo. O cálculo da latência foi feito baseando-se na estimativa do tamanho da banda dos acessos a modem, informação infelizmente ausente no log disponível. O log também foi tratado para retirar páginas dinâmicas não cacheáveis pelo *proxy*, e portanto não candidatas a *pushing* o que de certa forma poderia degradar o desempenho do algoritmo. Entretanto, acreditamos que as estimativas fornecem uma boa base para a compreensão de como a estratégia funcionaria e seus ganhos potenciais.

Como trabalhos futuros, temos a intenção de verificar os resultados de nossa estratégia para clientes de banda larga, pois não há empecilhos para o emprego da abordagem nesse contexto. Também pretendemos investigar métodos de cálculo de regras dinamicamente, de modo a evitar o problema do envelhecimento das regras. Além disso, pretendemos buscar mais informações sobre as características dos documentos no *proxy*, e estudar como elas podem ser transformadas em novos critérios para definição de relevância de regras ainda mais precisas.

Referências

[1] V. Padmanabhan, J. Mogul: "Improving HTTP Latency", *Proceedings of Second World Wide Web Conference*, p. 995-1005, Oct. 1994.

[2] R. Carceres, F. Doublis, A. Feldmann, M. Rabinovich, G. Glass: "Web Proxy Caching", *Proceedings of SIGMETRICS Workshop on Internet Server Performance*, 1998.

[3] M. Crovella, P. Barford: "The Network Effects of Prefetching", *Proceedings of IEEE Conf. on Computer and Communications*,p. 1232-1240, Mar. 1998.

[4] D. Duchamp: "Prefetching Hyperlinks", *Proceedings USENIX Symposium on Internet Technologies and Systems*, Oct. 1999.

[5] E. Cohen, B. Krishnamurthy, J. Rexford: "Improving End-to-End Performance of the Web Using Server Volumes and Proxy Filters", *Proceedings of ACM SIGCOMM Conference*,p. 241-253, Aug. 1998.

[6] P. Cao, E. Felten, A. Karlin, K. Li: "A study of integrated prefetching and caching strategies", *Proceedings of ACM SIGMETRICS*,p. 188-197, May 1995.

[7] R. Patterson, G. Gibson, E, Ginting, D. Stodolsky, J. Zelenka: "Informed Prefetching and Caching", *Proceedings of 15th ACM Symposium on Operating Systems Principles*, Dec. 1995.

[8] M. J. Zaki: "Efficient Enumeration of Frequent Sequences", *Proceedings of the 7th Intl. Conf. on Information and Knowledge Management*, Nov, 1998.

[9] R. Srikant, R. Agrawal: "Mining Sequential Patterns: Generalizations and Performance Improvements", *In 5th Intl. Conf. in Extending Database Technology*, 1996.

[10] C. Faloutsous, Y. Manolopoulos: "Fast Subsequente Matching in Time-Series Databases", *Proceedings of ACM SIGMOD Conference on Management of Data*, 1999.

[11] L. Breslau, P. Cao, L. Fan, G. Phillips and S. Shenker: "Web Caching and Zipf-like Distributions : Evidence and Implications", *Proceedings of the IEEE Infocom Conference*, 1999.

[12] T. Li: "Web-Document Prediction and Presending Using Association Rule Sequential Classifiers", *M.S. Thesis*, Jul. 2001.

[13] J. Gwertzman, M. Seltzer: "The Case for Geographical Push-Caching", *Proceedings of HotOS'95:The Fifth IEEE Workshop on Hot Topics in Operating Systems*, Washington, May, 1995.

[14] V. Almeida, M. Crovella, A. Bestavros and A. Oliveira "Characterizing reference locality in the WWW", *Proc. IEEE/ACM International Conference on Parallel and Distributed System (PDIS)*, Dec, 1996.