ALGORITMOS DE ESCALONAMENTO PARA ATUALIZAÇÃO DE BASES DE PÁGINAS WEB

Alunos: Eduardo Teixeira Cardoso e Caio Dias Valentim
Orientador: Eduardo Sany Laber

1. Introdução

O Crawler é um dos componentes mais importantes de uma ferramenta de busca. Cabe a este visitar os diferentes sítios para fazer a coleta das páginas que serão indexadas pela ferramenta. Devido a importância comercial dos Crawlers, pouco se sabe sobre como, de fato, as principais ferramentas de busca os implementam apesar de diversos aspectos gerais dos Crawlers estarem disponíveis na literatura [1-27].

Alguns dos trabalhos na literatura estudam os problemas de escalonamento confrontados pelos crawlers [1-12]. A maioria destes assume o crawler em um momento inicial, onde a topologia da rede não é conhecida [7-12]. O problema, neste caso, consiste em projetar uma política de visitação para as páginas que tenha bom desempenho em relação a um dado critério, como exemplo, soma das importâncias das páginas coletadas em um dado intervalo de tempo. Uma série de restrições devem ser respeitadas ao realizar as visitas. Em particular, deve haver um intervalo mínimo entre requisições ao mesmo sítio, intervalo este denominado Politeness.

Após o crawler ter coletado o conjunto inicial de páginas, faz-se necessário manter este conjunto atualizado e incrementá-lo periodicamente. Neste projeto estamos interessados em problemas de escalonamento confrontados pelos crawlers exatamente nesta fase. A diferença fundamental desta fase, em relação à inicial, é que nesta podem estar disponíveis uma série de informações sobre as páginas já visitadas que podem ser úteis para definir boas políticas de escalonamento [1, 2].

Entendemos que existe um espaço para formulação de modelos que tenham a capacidade de capturar os diferentes aspectos necessários para definir políticas otimizadas para atualização de bases de páginas Web e que possam ser resolvidos eficientemente do ponto de vista computacional.

Os principais objetivos do projeto são:

- Desenvolver modelos de otimização para o problema de escalonar atualizações de bases de páginas Web.
- Projetar algoritmos eficientes para escalonar atualizações e incremento de bases de páginas Web
- Comparar experimentalmente os algoritmos desenvolvidos com outros algoritmos propostos na literatura.

1.1 Trabalhos Relacionados

Garcia-Molina et. al. propõe em [2] diversas políticas para revisitação de páginas Web e estudam a efetividade destas sob duas métricas diferentes, o *Age* e o *Freshness*. É utilizado um processo de Poisson para modelar as mudanças das páginas ao longo do tempo e, em cima disso, são determinadas freqüências ótimas para revisitar as páginas. Este trabalho não se preocupa com a questão do Politeness e não discute como deve ser feita a distribuição das atualizações entre os crawlers.

Em [1], Wolf et. al. propõe uma política de escalonamento para revisitação de páginas que consiste de três etapas. Na primeira, a quantidade de atualizações x_i de cada página i é determinada através da resolução de um problema não-linear inteiro. Na segunda fase, o momento de cada atualização é determinado analiticamente. Na terceira fase, é resolvido um problema de transporte para determinação dos crawlers responsáveis pelas atualizações definidas no fim da segunda fase. São apresentadas simulações onde a política proposta é comparada com duas políticas descritas em [2], com resultados superiores para a maioria dos parâmetros considerados. Nesta simulação, foram utilizadas uma mistura de páginas em que 60% tinham sua modificação seguindo uma distribuição de Poisson, 30% seguindo uma distribuição de Pareto e 10% uma distribuição *quasi-deterministic*.

1.2 Principais Resultados

Baseado em idéias presentes na literatura [2], adotamos um modelo para a Web que segue um processo de Poisson para a modificação das páginas, visto que experimentos demonstram que a maioria das páginas seguem esta distribuição.

Adotamos um Politeness, definido como o intervalo de tempo mínimo entre duas requisições a um mesmo servidor, de 15 segundos. Esta restrição é importante, pois evita que, em uma situação real, o crawler gere uma sobrecarga nos servidores por conta da quantidade de requisições em um curto intervalo de tempo.

Consideramos que cada página possui uma importância intrínseca, e que nosso objetivo é, para um dado horizonte de tempo T, maximizar o nível de atualização de nossa base local através de uma média ponderada que leva em consideração a importância de cada página.

Com o modelo de Web descrito, construímos um Simulador que foi utilizado para conduzir os experimentos computacionais. Propomos também uma medida teórica que serve como limite superior para o Freshness quando respeitando o Politeness, para uso em trabalhos futuros.

Desenvolvemos uma política de escalonamento que respeita a restrição de Politeness e possui uma baixa complexidade computacional, cujos testes no Simulador mostraram que nossa política consegue obter resultados tão bons quanto aqueles que foram expostos por Wolf [1] quando sujeitos à mesma restrição de Politeness.

Os principais resultados obtidos em nossa pesquisa também são descritos em um artigo recentemente aceito na conferência LA-WEB 2007 [4].

1.3 Organização do Relatório

Este relatório está dividido da seguinte forma: Na seção 2, descrevemos nosso ambiente de simulação e testes. Na seção 3, discutimos os detalhes da política desenvolvida. Na seção 4, apresentamos nosso método de análise e os resultados obtidos. Finalmente, destinamos a seção 5 às nossas conclusões.

2. Simulador

Para que pudéssemos testar nossas políticas, optamos pelo desenvolvimento de um Simulador, através do qual teríamos uma série de vantagens em relação a experimentos realizados diretamente na Internet. Dentre estas vantagens, destacamos:

- Garantia de que estaríamos trabalhando com uma base estática, diferentemente da Internet, em que as páginas e os servidores estão sujeitos a modificações constantes e sujeito a eventuais problemas e interrupções de funcionamento ao longo do tempo.
- Menor necessidade de infraestrutura, tanto de equipamentos quanto de uma conexão com a Internet rápida o suficiente para comportar o volume de dados considerado
- Disponibilidade dos resultados em menor tempo

- Maior liberdade de testes, sem prejudicar os servidores da base de testes com diversos acessos repetidos
- Maior facilidade para testar diferentes cenários

Para seu funcionamento, o Simulador requer que sejam fornecidas informações que descrevem os crawlers a serem utilizados (com suas respectivas velocidades), os diversos servidores que farão parte do teste (também com velocidade) e os documentos (tamanho, servidor em que se encontra, importância e taxa média de atualização observada), assim como os parâmetros da política de escalonamento, discutidos posteriormente.

De posse destas informações, o Simulador calcula e mantém uma "linha do tempo" de eventos, que é a base do seu funcionamento. Todas as modificações de páginas são eventos que indicam a desatualização de um documento. O início e término do download são modelados também como eventos, que são inseridos ao longo da execução da simulação, dentre outros eventos de controle. Toda a simulação é então regida por esta linha do tempo, através do processamento do evento de maior prioridade até que aconteça o evento de "Fim de Simulação".

Devido à magnitude do volume de dados processado, uma série de preocupações com performance se tornaram essenciais para a viabilização das simulações. A linha do tempo é representada por uma lista de prioridades (heap), que permite que eventos sejam inseridos e removidos em O(log n), respeitando a ordenação temporal de forma eficiente. Uma série de estruturas auxiliares é utilizada também para tornar a computação eficiente, sobre as quais falaremos mais adiante.

Ao término da simulação, temos disponíveis uma série de medidas do desempenho da política adotada (sendo o Freshness - ou frescor da base - a mais relevante), permitindo a comparação das diferentes políticas em diferentes cenários.

2.1 Medidas de Avaliação

Para comparar o desempenho de uma política em um dado cenário, temos de avaliar quatro medidas. São elas:

- Freshness, indicativo do percentual médio de atualização da base ao longo do horizonte de tempo
- Weak Politeness, indicativo do Freshness Ótimo respeitando apenas a restrição de Politeness, que serve de cota superior para o Freshness de uma política que respeita o Politeness
- InfCrawlers, indicativo do Freshness obtido caso todas as atualizações sejam realizadas nos tempos previstos, ou seja, se tivéssemos infinitos crawlers a nossa disposição
- NonPolite, indicativo do Freshness Ótimo sem respeitar a restrição de Politeness. Serve de cota superior para o Freshness

A primeira medida é computada experimentalmente pelo Simulador, enquanto as outras três são medidas teóricas do Freshness nas dadas situações particulares. Comparando o Freshness obtido com as outras medidas, podemos ter uma noção da qualidade do resultado obtido.

De forma alternativa, podemos utilizar a métrica de Staleness ao invés do Freshness, que possui significado inverso, ou seja, é definido como (1 – Freshness) ou como o percentual da base não-atualizado.

2.2 Detalhes de Implementação

O Simulador se baseia em dois grandes grupos lógicos. O primeiro deles é a parte relativa ao ambiente Web no que diz respeito a modificação de suas páginas. O segundo reflete o funcionamento dos Crawlers em si, tratando das atualizações da base de dados.

Como já mencionado, nossa principal estrutura é uma lista de prioridades, que funciona como nossa linha do tempo de simulação, onde eventos são inseridos e processados, regendo nossa execução.

Cada evento de simulação é uma estrutura com três campos: tipo, identificador e prioridade. A prioridade é o instante de tempo em que o evento deve ocorrer, e o uso do identificador está diretamente ligado ao tipo de evento, que pode ser um dos seguintes: Crawler Livre, Documento Modificado, Escalonador Pronto, Estatística e Fim de Simulação.

No início da simulação, consideramos que a base local se encontra completamente atualizada. São inseridos eventos Documento Modificado, relativos à primeira modificação de cada um destes documentos, que uma vez processados, inserem outro evento do mesmo tipo indicando a próxima modificação, calculado através de uma distribuição de probabilidade, utilizando a taxa média de modificação de cada documento. São também inseridos eventos de Crawler Livre para cada crawler disponível, que nos permite iniciar a simulação através de seu processamento.

Um evento Crawler Livre implica em requisitar um documento para atualização, respeitando sempre a restrição de Politeness. Caso não haja documentos disponíveis para atualização em um dado momento, um novo evento Escalonador Pronto é inserido para o instante de tempo da próxima atualização, quando iremos então associá-la a este crawler. No momento da associação, um outro evento Crawler Livre é inserido para o instante em que o download termina, e o crawler pode receber outra atualização.

O evento de Estatística tem a função de calcular as métricas de avaliação até o momento, para acompanhamento do desempenho da simulação. Este evento está também implícito quando encontramos um evento Fim de Simulação, quando as atividades do simulador são cessadas e um relatório final é emitido.

3. Políticas Desenvolvidas

Das políticas propostas na literatura [1, 2], não é do nosso conhecimento nenhuma que leve em consideração o tempo de Politeness.

Nossa política é inspirada na política proposta em [1], com a diferença principal de que, desde o princípio, existe uma preocupação para se respeitar o tempo de Politeness. Assim como a política original, nossa política se divide em três fases:

1. Definição da quantidade de vezes que um documento deve ser atualizado no horizonte de tempo [0, T]
2. Definição dos momentos em que as atualizações devem ser realizadas
3. Distribuição das atualizações entre os crawlers disponíveis

A primeira fase utiliza um método guloso descrito em [1] para a distribuição de atualizações, no qual medimos o ganho em Freshness caso um documento receba uma atualização a mais do que as que já foram alocadas a ele. O documento que apresentar o maior ganho, recebe uma atualização a mais, e o processo se repete até que não seja possível alocar mais atualizações. Nesta fase, nos diferenciamos de [1] por restringir a alocação de atualizações aos documentos cujos servidores ainda tenham tempo livre suficiente para respeitar o tempo de Politeness entre as atualizações de seus documentos. Seja x_i a quantidade de atualizações da página i do servidor s, e P o intervalo de Politeness, temos:

$$\sum_{x_i \in s} \leq \left\lfloor \frac{T}{P} \right\rfloor$$

Seja R o total de atualizações restantes disponíveis para alocação, $x^*_{s,i}$ a quantidade de atualizações do i-ésimo documento do servidor s, w^s_i a importância deste mesmo documento e $A^s_i(x)$ o maior freshness que pode ser obtido com x atualizações para este documento, o algoritmo consiste nos seguintes passos para um horizonte de tempo T com intervalo de Politeness P:

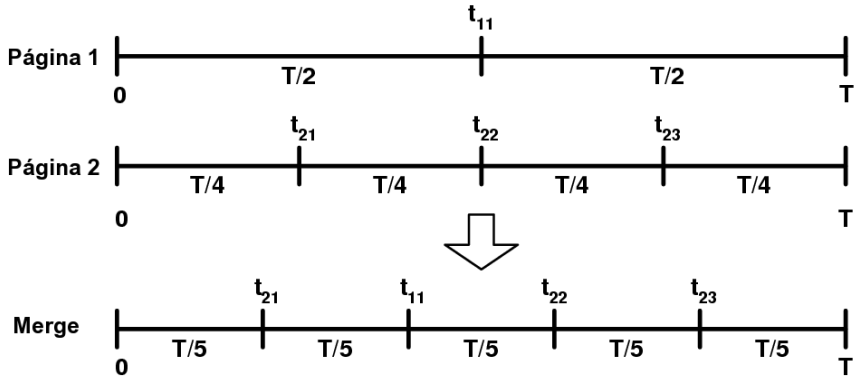
1. Inicializa todos os $x^*_{s,i}$ com 0
2. Enquanto R > 0, e existe s pertencente a {1, ..., m} tal que:
   $\sum_{i=1}^{n_s} x^*_{s,i} < \lfloor \frac{T}{P} \rfloor$

   a. Selecione a página (s, i) que maximiza o aumento de freshness:
      $w_i^s \cdot [A_i^s(x^*_{s,i}) - A_i^s(x^*_{s,i} + 1)]$
      sem violar a restrição de Politeness para o servidor s.
   b. $x^*_{s,i} \leftarrow x^*_{s,i} + 1$
   c. $R \leftarrow R - 1$

Na segunda fase, determinamos o tempo em que cada atualização irá ocorrer. Foi observado em [2] que a melhor política de visitação a uma mesma página se dá em intervalos regulares. Assim, são calculados, para cada documento, os instantes ótimos em que suas atualizações deveriam ocorrer. Uma vez calculados os instantes, as atualizações de documentos de um mesmo servidor são ordenadas crescentemente por tempo, com desempate também crescente do identificador interno de cada página. De posse desta ordenação, as atualizações de documentos neste servidor são distribuidas uniformemente ao longo do horizonte de tempo T através do procedimento que denominamos MERGE. Seja t_i a i-ésima atualização de um dado servidor, e x_s o total de atualizações daquele servidor, os tempos são determinados da seguinte forma:

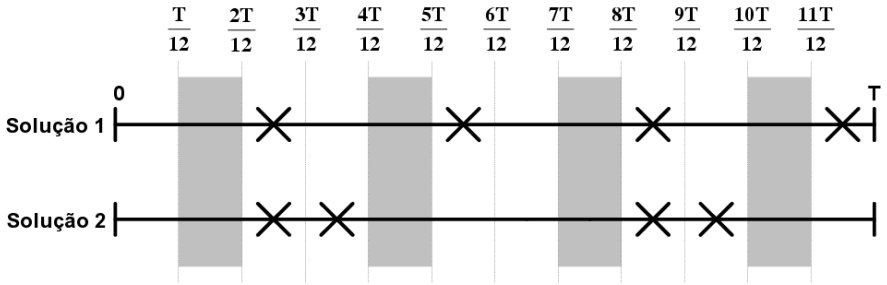
$$t_i = i \cdot \frac{T}{x_s + 1}$$

Abaixo, um exemplo do procedimento:



Na terceira fase, realizamos a distribuição de atualizações entre os crawlers de forma dinâmica, em tempo de execução, seguindo um regime de fila. Quando existir um crawler livre, é associado a ele o documento mais atrasado dentre as atualizações que podem ser feitas naquele instante. Nossa política não adianta atualizações, sob o risco de perda de eficácia, como veremos abaixo.

Embora possuam um tempo de execução polinomial, os algoritmos conhecidos para um problema de transporte podem ser ainda muito lentos para nosso problema, dado que a quantidade de atualizações em uma aplicação usual é imensa. Ainda, em termos de maximização do Freshness, se mostra ser mais importante manter o tamanho do intervalo entre atualizações consecutivas o mais uniforme possível [2] do que tentarmos seguir exatamente os tempos programados até o momento. Esta última observação é ilustrada na imagem abaixo, onde consideramos uma página programada para ser atualizada nos tempos T/12, 4T/12, 7T/12 e 10T/12. Se os slots correspondentes a estes tempos não estiverem disponíveis (colunas cinzas da figura), então a abordagem em [1] tenta associar estas atualizações o mais perto possível dos tempos programados. Isto pode levar à Situação 2, ilustrada na imagem, onde o X indica slots associados a cada atualização. Como podemos observar, esta solução não distribui as atualizações de uma maneira uniforme ao longo do horizonte de tempo, o que é ruim para a função-objetivo. Por outro lado, a Solução 1, também ilustrada, garante uma distribuição muito mais uniforme.



No Simulador, foi necessário realizar o controle dos tempos de atualização através de um conjunto de listas de prioridades, com o intuito de que, uma vez que as atualizações se atrasem, existe o risco de se infringir o Politeness caso o processamento se dê na ordem cronológica. Para respeitar esta restrição com eficiência computacional, nosso conjunto de heaps se divide da seguinte forma: um conjunto de heaps indicam, para cada servidor, a ordem de atualização de seus documentos, e um heap de servidores, que indica, para cada servidor, o menor tempo hábil de atualização de um de seus documentos, já considerando a restrição de Politeness. Assim, podemos escolher os documentos de forma eficiente, respeitando sempre o Politeness.

4. Análise Experimental

Descrevemos aqui os experimentos realizados para avaliar o desempenho da política que desenvolvemos.

O leitor deve assumir o seguinte cenário: (i) as páginas são distribuídas de acordo com uma distribuição de Zipf [5] com parâmetro $\theta = 1$ (muitas páginas concentrada em poucos servidores); (ii) páginas com pesos uniformes e (iii) os parâmetros λ_i das distribuições exponenciais, representativos da taxa média de modificação, são uniformemente distribuídos no intervalo [0.01,1]. Embora esta não seja a melhor distribuição para alguns casos de acordo com a modelagem utilizada em [1], ela reflete de forma razoável o comportamento da maioria das páginas na Web [2].

Utilizamos um horizonte de tempo T = 24 horas, com 10 crawlers de velocidades uniformes, responsáveis por manter atualizada uma base de dados de 1 milhão de documentos, distribuídas entre 2 mil servidores, seguindo os critério acima.

Assumimos que cada crawler é capaz de realizar o download de 10 páginas por segundo, o que implica um total de 8.640.000 atualizações por crawler neste intervalo de tempo. Com base nisso, definimos R = 10 x 8.640.000 como o máximo de atualizações que podemos realizar, respeitando um Politeness P = 15 segundos entre requisições a um mesmo servidor.

Em todos os nossos experimentos o Staleness (1-Freshness) ponderado obtido pela nossa política, referenciada como PSP, é comparado com o Staleness ponderado de duas outras políticas denotadas por MP2 e M-Wolf e também com três limites teóricos, NonPolite, InfCrawlers e WP.

- MP2 é uma política similar a PSP exceto pelo fato que em vez de executar o procedimento MERGE (Fase 2), MP2 distribui as atualizações atribuídas a página (s, i) em intervalos uniformes ao longo do horizonte de tempo.
- M-Wolf é uma política que primeiro computa o número de atualizações por páginas e os instantes para estas atualizações utilizando a abordagem de [1]. Depois, a política executa fase 3 da PSP para atribuir atualizações aos crawlers.
- InfCrawlers é o Staleness ponderado que seria obtido caso fosse possível seguir fielmente o escalonamento proposto pelo procedimento MERGE
- WP é um limite inferior para o Staleness ótimo ponderado. O mais próximo que a política PSP chega do valor de WP, melhor é a qualidade do resultado.
- A métrica NonPolite é o Staleness mínimo que pode ser obtido se um número suficientemente grande de crawlers pudessem executar R downloads ao longo do horizonte de tempo. Esta cota é reportada em [1].

O gráfico da Figura 1 é obtido variando o parâmetro θ da distribuição de Zipf que define a distribuição de páginas nos servidores. Os seguintes resultados foram observados:

1. O Staleness obtido pelo PSP é menor que o obtido pelo MP2, o que indica que a maneira que o MERGE define os tempos de atualização na Fase 2 provê um ganho significativo.
2. O Staleness do PSP ficou bastante próximo da métrica InfCrawlers. Isso é bastante interessante já que sugere que a estratégia proposta na Fase 3 não pode ser melhorada utilizando a técnica mais sofisticada e dispendiosa proposta em [1].
3. O Staleness do PSP ficou próximo da métrica WP. Isso indica que o resultado obtido pela PSP está próximo do resultado ótimo.
4. O staleness do M-Wolf ficou 9.5% acima do PSP na média.

Variando a forma de fixar os λ_i das distribuições exponenciais, selecionados segundo a lei de Zipf [5], aonde um maior valor do parâmetro θ indica que poucas páginas são usualmente modificadas, obtemos outro gráfico, exposto na Figura 2. Novamente, o Staleness obtido pelo PSP é melhor do que os obtidos por M-Wolf e MP2, próximo ao WP e extremamente próximo da métrica InfCrawlers. Executamos também alguns testes aonde variamos os pesos das páginas aonde obtemos resultados similares aos já descritos.

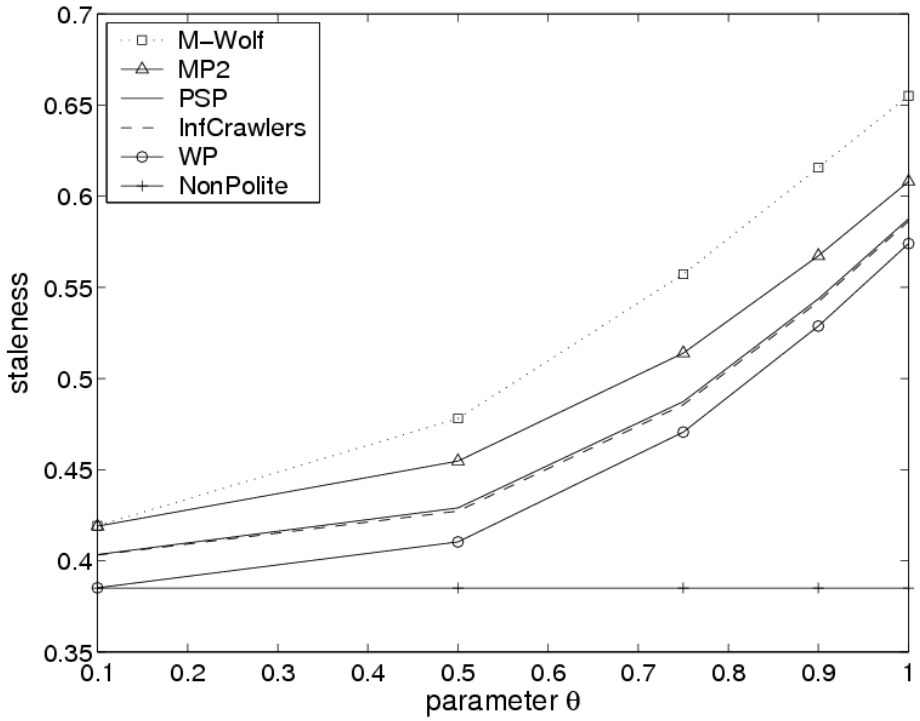


Figura 1

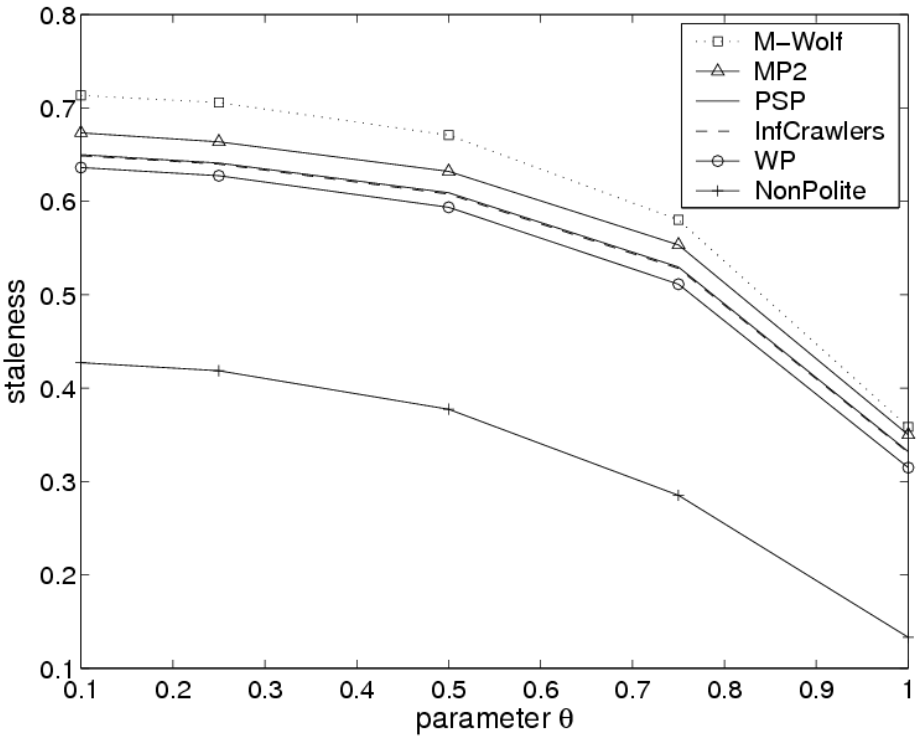


Figura 2

Concluímos esta seção mencionando que tanto a Fase 1 quanto a Fase 2 de nossa políca gastou poucos segundos para executar em uma máquina Pentium III Xeon 2.4GHz com 1 Gb de RAM

5. Conclusão

Através do desenvolvimento de um Simulador conseguimos criar um ambiente controlado no qual podemos testar o progresso e a qualidade das diversas políticas de escalonamento da literatura diante de circunstâncias variadas, assim como criar nossas próprias políticas e compará-las de forma mais direta. O simulador permite lidar com páginas com características diferentes (importância, tamanho) e distribuídas em diversos servidores heterogêneos (velocidades diferentes).

O principal resultado que alcançamos foi o desenvolvimento de uma política eficiente para revisitação de páginas que leva em conta a restrição de politeness. A restrição de politeness estabelece que duas requisições consecutivas a um mesmo servidor não devem ocorrer dentro de um intervalo de tempo pré-determinado (em geral, 15 segundos). Não conhecemos outro trabalho na literatura que tenha levado em conta esta restrição para determinar com que freqüência as páginas devem ser revisitadas. Realizamos experimentos que demonstram que:

- Considerar a restrição de politeness na construção da política tem grande impacto na qualidade da solução final;
- Nossa política tem um desempenho melhor que a política proposta por Wolf [1], quando o ambiente exige o cumprimento do tempo de politeness (como ocorre na prática).
- Nossa política se afastou pouco da solução ótima. Este afastamento foi calculado através de um limite inferior que desenvolvemos para medir o melhor frescor do repositório que pode ser obtido por uma política de revisitação, dado que a restrição de politeness deve ser respeitada. Este limite inferior pode ser utilizado como ferramenta de análise de outras políticas que venham a ser propostas.

Referências

1 – WOLF, J. L. e SQUILLANTE, M. S. e Yu, P. S. e SETHURAMAN, J. e OZSEN, L. **Optimal crawling strategies for web search engines**. In WWW ’02, pages 136-147, New York, NY, USA, 2002. ACM Press.

2 – GARCIA-MOLINA, H. e CHO, J. **Effective page refresh policies for Web crawlers**. ACM Trans. Database Syst., 28(4):390-426, 2003.

3 – MANNING, C. D. e RAGHAVAN, P. e SCHÜTZ, H. **Introduction to Information Retrieval**, Cambridge University Press. 2008.

4 – SOUZA, C. e LABER, E. e VALENTIM, C. e CARDOSO, E. **A Polite Policy for Revisiting Web Pages**. LA-WEB 2007, to appear.

5 – ZIPF, G. K. **Human Behaviour and the Principle of Least Effort: An Introduction to Human Ecology**. Addison-Wesley Press, 1949.

6 – EDWARDS, J. e MCCURLEY, K. S. e TOMLIN, J.A. **An adaptive model for optimizing performance of an incremental web crawler**. WWW, pp. 106-113, 2001.

7 – CASTILLO, C. e MARÍN, M. e RODRÍGUEZ, A. e BAEZA-YATES, R. A. **Scheduling Algorithms for Web Crawling**. *WebMedia/LA-WEB*, pp. 10-17, IEEE Computer Society, 2004.

8 – CHO, J. e GARCIA-MOLINA, H. **Parallel crawlers**. Proceedings of the 11th World Wide Web conference (WWW11), 2002.

9 – NAJORK, M. e WIENER, J. L. **Breadth-first crawling yields high-quality pages**. *WWW*, pp. 114-118, 2001.

10 – COFFMAN, E. G. e LUI, Z. e WEBER, R. R. **Optimal Robot Scheduling for Web Search Engines**. *Journal of Scheduling*, 1(1), June 1998.

11 – CHO, J. e GARCIA-MOLINA, H. e PAGE, L. **Efficient Crawling Through URL Ordering.** *Computer Networks*, Vol. 30, Number 1-7, pp. 161-172, 1998.

12 – BAEZA-YATES, R. e CASTILLO, C. **Balancing Volume, Quality and Freshness in Web Crawling**. *Soft Computing Systems - Design, Management and Applications, Frontiers in Artificial Intelligence and Applications* Vol. 87, pp. 565-572, IOS Press Amsterdam, Berlin, Oxford, Tokyo, Washington D.C., 2002.

13 – EICHMANN, D. **The RBSE spider: balancing effective search against web load**. *WWW,* 1994.

14 – PINKERTON, B. **Finding what people want: Experiencies with the WebCrawler**. *WWW*, 1994.

15 – MCBRYAN, O. A. **GENVL and WWWW: Tools for taming the web**. *WWW,* 1994.

16 – BURNER, M. **Crawling towards eternity – building an archive of the world wide web**. *Web Techniques*, 1997.

17 – MILLER, R. e BHARAT, K. **Sphinx: A framework for creating personal, site-specific web crawlers**. *WWW*, 1998.

18 – BRIN, S. e PAGE, L. **The anatomy of a large-scale hypertextual Web search engine**. *Computer Networks and ISDN Systems*, vol. 30, no. 1-7, pp. 107-117, 1998.

19 – SILVA, A. S. e VELOSO, E. A. e GOLGHER, P. B. e RIBEIRO-NETO, B. A. e LAENDER, A. H. F. e ZIVIANI, N. **Cobweb – a crawler for the brazilian web**, in *SPIRE/CRIWG*, 1999, pp.184-191.

20 – HEYDON, A. e NAJORK, M. **Mercator: A scalable, extensible web crawler**. *WWW*, vol. 2, no. 4, pp. 219-229, 1999.

21 – BURKE, R. D. **Salticus: guided crawling for personal digital libraries**, in *ACM/IEEE Joint Conference on Digital Libraries*, 2001, pp. 88-89.

22 – CHAKRABARTI, S. ***Mining the** Web.* Morgan Kaufmann Publishers, 2003.

23 – AILLERET, S. **Larbin, a multi-purpuse web crawler**. http://larbin.sourceforge.net/index-eng.html, 2002.

24 – DACHARAY, L. **Webbase web crawler**. http://www.freesoftware.fsf.org/webbase, 2004.

25 – **HT://Dig**, http://www.htdig.org, 2004.

26 – SULLIVAN, D. e SHERMAN, C. **Search Engine Watch.** http://www.searchenginewatch.com/reports, 2002.

27 – **BotSpot.** http://www.botspot.com, 2004.