PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 19/07

A Polite Policy for Revisiting Web Pages

Críston Pereira de Souza
Eduardo Sany Laber
Caio Dias Valentim
Eduardo Teixeira Cardoso

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22451-900
RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. 19/07 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena August, 2007

A Polite Policy for Revisiting Web Pages

Críston Pereira de Souza, Eduardo Sany Laber, Caio Dias Valentim e Eduardo Teixeira Cardoso

{csouza,laber}@inf.puc-rio.br, kakaio9@gmail.com, ecardoso@grad.inf.puc-rio.br

Abstract. We propose and analyze an efficient scheduling policy for revisiting web pages so as to keep a local repository as up to date as possible. The main contribution of our policy is the fact that it takes into account the politeness constraint in order to compute the revisiting times.

Our experiments suggest that the performance of our policy is very close to the optimal one and that it is significantly better than the one obtained using a reasonable variation of the policy proposed in [Wolf et. al., WWW 2002].

Keywords: Internet, Search Engines, Revisiting Policies.

Resumo. Neste trabalho, propomos e analisamos uma política de escalonamento eficiente para revisitação de páginas web de modo a manter o mais atualizado possível um repositório local. A principal contribuição de nossa política é o fato dela considerar a restrição de politeness ao computar os instantes de revisitação.

Nossos experimentos sugerem que a performance de nossa política é muito próxima da obtida por uma política ótima, e é significantemente melhor que a performance obtida por uma variação da política proposta em [Wolf et. al., WWW 2002].

Palavras-chave: Internet, Máquinas de Busca, Políticas de Revisitação.

In charge of publications:
Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22451-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3527-1516 Fax: +55 21 3527-1530
E-mail: bib-di@inf.puc-rio.br
Web site: http://bib-di.inf.puc-rio.br/techreports/

1 Introduction

A web crawler is one of the core components in a search engine architecture. Its main function is to download web pages which will be further indexed. Due to the commercial importance of web crawlers, little is known about how they are implemented by the main search engines.

As many web pages are frequently modified, the crawler must continuously revisit them to keep the search engine repository up to date. An obsolete repository may reduce the quality of the search engine service, as the query result may contain links to web pages that no longer exist, or to web pages that are no longer related to the user query. As these repositories are usually very large, only a portion of web pages can be revisited in a reasonable time interval. Hence, a crawler shall implement a scheduling policy to determine how many times and when each web page shall be downloaded so as to keep the repository as up to date as possible.

Scheduling policies must take into account computational environment limitations [3]. The number of downloads a crawler can perform in a given time interval is limited by the communication channel and the machine characteristics. In addition to that, many requests to the same web server can overload it. A way to prevent such an overload is to require a scheduling policy to be 'polite'. A typical way to be polite is waiting a fixed amount of time between consecutive requests to the same web server. This constraint is referred here as *politeness constraint*.

Two facts suggest a significant impact of the politeness constraint to the crawling policies [3]. First, the time between requests, typically ranging from 15 to 60 seconds, is relevant when compared to the crawler cycle and the average download time. Second, most of web pages are hosted in few web servers. Despite of this scenario, scheduling policies proposed in previous work [4, 8] do not take into account this constraint in order to determine when each page shall be revisited. The main contribution of this paper is to propose and analyze an efficient revisiting policy, which explicitly takes into account the politeness constraint.

1.1 Problem definition

Our problem consists of devising an efficient scheduling policy for revisiting web pages so as to keep a repository with n web pages, hosted in m web sites[1], as up to date as possible during a time horizon $[0, T]$. For this purpose we may execute C parallel crawlers. This set of crawlers can do at most R downloads in the time horizon $[0, T]$. The time between two consecutive requests to the same web server must be at least P (politeness constraint).

Following the approach of [4, 8] we assume, for each page i, a probability distribution $G_i()$ on the time interval between its consecutive changes. This assumption turns out to be important to define a way to measure how up to date is a repository.

1.1.1 Measuring repository freshness

There exist different proposals to measure how up to date is a repository [8]. Here, we employ the *staleness* metric proposed by Wolf et. al. [8]. Let x_i be the number of updates of page i in the interval $[0, T]$. Given the update times $t_{i,1}, \ldots, t_{i,x_i}$ of i, one can compute

[1] We assume that there is only one web server per web site.

the probability of this page to be out of date in a particular time at the interval $[0, T]$. The staleness metric $a_i(t_{i,1}, \ldots, t_{i,x_i})$ for a page i is the mean of this probability over the interval $[0, T]$. Wolf et. al. [8] showed that $a_i(t_{i,1}, \ldots, t_{i,x_i})$ is given by

$$\frac{1}{T} \sum_{j=0}^{x_i} \int_{t_{i,j}}^{t_{i,j+1}} \left(1 - \lambda_i \int_0^{\infty} (1 - \overline{G}_i(t - t_{i,j} + v)) \mathrm{d}v\right) \mathrm{d}t,$$

where $\lambda_i^{-1} > 0$ is the mean of $G_i(\cdot)$ and $\overline{G}_i = 1 - G_i$.

The staleness of a repository is the mean of the staleness of its pages.

1.1.2 Non-uniform importance

In typical situations some pages may be more important than others. For example, we can be more concerned about the staleness of pages with greater chance to be returned by the search engine in a user query. Some importance metrics have been proposed, such as PageRank [7] and Embarrassment [8].

When we have non-uniform importances, we shall minimize the *weighted staleness* of the repository which is defined as

$$\sum_{i=1}^{n} w_i \times a_i(t_{i,1}, \ldots, t_{i,x_i}),$$

where w_i is the importance of the page i.

1.2 Related work

Cho et. al. propose in [4] various policies for revisiting web pages and study their effectiveness under two different metrics, the *age metric* and the *freshness metric*. They employ a Poisson process to model the web page changes and, based on that, they determine optimal frequencies for revisiting them. This work does not consider the politeness constraint and it does not discuss how the updates shall be distributed to the crawlers.

In [8], Wolf et. al. propose a scheduling policy for revisiting web pages that consists of three phases. In Phase 1, the number of updates x_i of each page i is determined through the resolution of a non linear integer program. In Phase 2, the time of each update is analytically determined. Phase 3 solves a transportation problem to assign the updates programmed at the end of Phase 2 to the available crawlers. They present simulations where their policy is compared with two polices proposed in [4], the *weighted proportional* policy (the number of updates is proportional to $w_i\lambda_i$) and the *weighted uniform* policy (the number of updates is proportional to w_i). Wolf's policy is significantly better than these simple policies for most of the parameter values. In these simulations they use a mixture of 60% Poisson, 30% Pareto and 10% quasi-deterministic in the change interval distributions of the web pages. As in Cho's work, the politeness constraint is not taken into account.

Eckstein et. al. propose in [5] an optimal scheduling algorithm for monitoring information sources whose contents change at times modeled by a Poisson process. This monitoring consists of probes to the information sources, respecting a minimal time interval between consecutive probes to the same source (politeness constraint), and a maximal

number of probes available for each source (resource limitations). However, their approach can be directly applied to our problem only in the case where each server hosts only one web page.

1.3 Main results

Here, we propose an efficient scheduling policy for revisiting web pages. In oppose to the policies proposed in the literature, the one presented here takes into account the politeness constraint in the determination of both the number of updates and the update times of the pages.

Our approach is inspired in Wolf's approach and it also consists of three phases. In the first one the number of updates per page is determined through the resolution of a non linear integer program (NLIP). The main difference from the NLIP of [8] is that we add a set of inequalities, one for each server, to bound the number of updates per server. We show that our new NLIP can also be efficiently solved. Then, in Phase 2 we employ a $O(|R| \log n)$ time algorithm to determine, for each page, a list of update times that respect the politeness constraint. We shall note that the existence of the politeness constraint avoids us to employ the analytical methods of [8]. Finally, in Phase 3 we assign updates to crawlers using a much simpler and efficient scheme than the network flow method proposed in [8].

We carried out experiments where we compare our policy with a variation of the one proposed in [8]. This variation uses our Phase 3, instead of the original one, to determine which crawler shall handle each update. The results observed in these experiments indicate that our policy is able to keep the repository significantly more up to date – for most of our experiments the staleness achieved by the variation of Wolf's policy is at least 10% larger than the one achieved by ours. Although we had initially planned to compare our results with Wolf's original policy, at some point we decided to use the aforementioned variation because our implementation of the Phase 3 of Wolf's policy was spending a non-reasonable amount of time to assign updates to crawlers.

Another contribution of our work is a tight lower bound one the best possible weighted staleness that can be achieved without violating the politeness constraint. We refer to this bound as *weak politeness*. In our experiments we observe that the staleness achieved by our policy gets very close to the weak politeness lower bound, which indicates that: (i) there exists just a small room, if any, for improvement on our policy and (ii) the weak politeness is a suitable tool for evaluating the performance of scheduling policies that respect the politeness constraint.

1.4 Paper organization

This paper is organized as follows. In Section 2 we present our scheduling policy for updating web pages in a local repository. In Section 3 we discuss theoretical issues of our policy. In Section 4 we present our experimental results. Finally, Section 5 is regarded to our conclusions.

2 A policy for updating web pages in a local repository

In this section we detail each of the three phases of our policy.

2.1 Determining the number of updates per page

In order to determine the number of updates per page we solve a non linear integer program. Some notation is required to explain this program. Let n_s be the number of pages in server s, (s,i) be the ith page of server s, and w_i^s be the importance of (s,i). We use $A_i^s(x)$ to denote the minimum staleness that can be achieved by (s,i) under the constraint that it can be updated at most x times. Note that in the definition of $A_i^s(x)$ we do not consider the politeness constraint. The goal of the problem $\mathcal{S}$ presented below is to determine the number of updates $x_{s,i}$ for each page (s,i) so as to minimize

$$\sum_{s=1}^{m}\sum_{i=1}^{n_s} w_i^s A_i^s(x_{s,i})$$

constrained to

$$\sum_{s=1}^{m}\sum_{i=1}^{n_s} x_{s,i} \leq R \tag{1}$$

$$\sum_{i=1}^{n_s} x_{s,i} \leq \lfloor T/P \rfloor, \quad s = 1,\ldots,m \tag{2}$$

$$x_{s,i} \in \mathbb{N}, \quad \text{for every page } (s,i) \tag{3}$$

The objective function of the above formulation computes the weighted staleness of the repository if page (s,i), for every valid (s,i), is updated $x_{s,i}$ times in equally spaced instants along the time horizon (this is the best way to distribute the $x_{s,i}$ updates [8]).

Constraint (1) provides an upper bound R on the number of updates allowed along the time interval $[0,T]$. An estimate for R can be obtained in terms of the number of crawlers and the average download time of the pages.

Constraint (2) gives an upper bound on the number of updates per server. In fact, this constraint is a necessary condition for feasibility, that is, all schedules that satisfy the politeness constraint satisfy Constraint (2) as well. However, it shall be clear that Constraint (2) does not ensure that two updates of pages in the same server cannot occur in a time interval smaller than P. This last requirement is handled in Phase 2.

We present below an efficient algorithm to find the optimal solution for problem $\mathcal{S}$. Its optimality is proved in [6].

GREEDY Procedure

1. Set all $x_{s,i}^*$'s to 0;
2. While $R > 0$, and there exists $s \in \{1,\ldots,m\}$ such that $\sum_{i=1}^{n_s} x_{s,i}^* < \lfloor T/P \rfloor$,
   (a) Select the page (s,i) which maximizes the staleness reduction
   $$w_i^s[A_i^s(x_{s,i}^*) - A_i^s(x_{s,i}^*+1)]$$
   without violating Constraint (2);
   (b) $x_{s,i}^* \leftarrow x_{s,i}^* + 1$;
   (c) $R \leftarrow R - 1$;

This algorithm can be implemented in $O(|R| \log n)$ by using a binary heap structure which always maintains in its root the page which provides the largest gain in the objective function.

Thus, at the end of this phase we have the number of updates $x^*_{s,i}$ for every page (s, i).

2.2 Determining the update times

After calculating the number of updates per page, we define the time when each update shall occur. Were we not concerned with the politeness constraint we should distribute the updates of a single page in equally spaced instants throughout the time interval. However, for our purposes, this solution may not be reasonable since many pages stored in the same server could be assigned to close instants, which would violate the politeness constraint. Thus, we need to develop another strategy.

The second phase of our strategy is implemented through a procedure denoted by MERGE, which takes as input the number $x^*_{s,i}$ of updates for each page (s, i) and defines the time t_{sij} of the j-th update of page (s, i). For notational convenience we drop the server index s in the remaining of this section whenever the context is clear.

MERGE Procedure

Execute the following steps for every server s:

1. For each page $i = 1, \ldots, n_s$, and for each update $j = 1, \ldots, x^*_{s,i}$ of page (s, i), set $t^*_{ij} \leftarrow j\frac{T}{x^*_{s,i}+1}$;

2. Sort the updates according to t^*_{ij} (break ties using the page indexes);

3. For each page $i = 1, \ldots, n_s$, and for each update $j = 1, \ldots, x^*_{s,i}$ of page (s, i), set

$$t_{ij} \leftarrow r_{ij} \times \frac{T}{1 + \sum_{i=1}^{n_s} x^*_{s,i}},$$

   where r_{ij} is the rank of the j-th update of the (s, i) in the order produced at the end of Step 2.

Figure 1 illustrates the MERGE procedure. Consider a server s storing two pages, say p_1 and p_2, where p_1 must be updated just once and p_2 3 times, that is, $x_1 = 1$ and $x_2 = 3$. In this scenario, the algorithm set, at Step 1, $t^*_{11} = T/2$, $t^*_{21} = T/4$, $t^*_{22} = 2T/4$, e $t^*_{23} = 3T/4$. As there are two updates scheduled to the same time ($t^*_{11} = t^*_{22}$), the algorithm schedules the first update of p_1 before the second one of p_2 (page index order). Thus, we have the following ranks $r_{21} = 1, r_{11} = 2, r_{22} = 3, r_{23} = 4$, at the end of Step 2. Since there are 4 updates scheduled at server s, the algorithm splits the interval $[0, T]$ into 5 subintervals of equal lengths and it assigns $t_{21} = T/5$, $t_{11} = 2T/5$, $t_{22} = 3T/5$ and $t_{23} = 4T/5$.

MERGE procedure can be easily implemented in $O(|R| \log n)$ time.

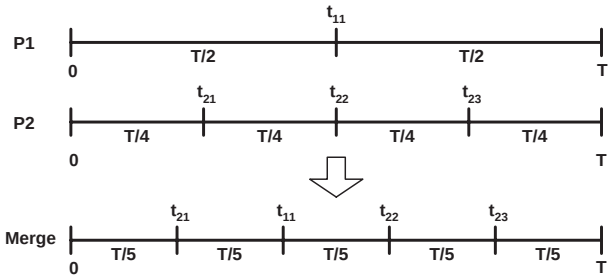

Figure 1: An example for MERGE procedure.

2.3 Assigning updates to crawlers

After determining the scheduling times, our policy defines how the crawlers perform the programmed updates. This problem of assigning crawlers to updates is called the *crawler assignment problem*. A strategy to solve this problem is presented in [8].

We employ a strategy much simpler than the one proposed in [8]. We say that an update j of a page i programmed to time t_{ij} (in Phase 2) is free at a given time t if $t_{ij} < t$ and no request was performed to the server where i is stored in the interval $[t - P, t]$. When a crawler c is available at time t our strategy assigns to c the most delayed update, among all free updates at time t. It breaks delay ties (those are only possible for updates associated with pages stored in distinct servers) selecting the server with the smallest index. This last rule is consistent with the goal of keeping the interval between consecutive updates of a page as uniform as possible. This strategy requires $O(\log m)$ time per crawler assignment.

Now we describe the strategy of [8] and present some arguments which motivated us to develop and adopt an alternative strategy. For each crawler k, the strategy of [8] assumes uniform download times. Therefore, the time interval $[0, T]$ is split into S_k uniform slots, where S_k is the number of updates that can be performed by crawler k in T time units (note that the crawlers may have different speeds). The crawler capacity is given by $R = \sum_{k=1}^{C} S_k$, where C is the number of crawlers. The strategy consists of formulating the crawler assignment problem as a transportation problem, where the source nodes have unit supply and correspond to the updates, while the demand nodes have unit demand and correspond to the crawlers slots. Each source node is connected to a demand node through a unit capacity edge. The cost of an edge (up, sl) connecting update up to slot sl is given by the absolute difference between the time in which the update up is programmed and the initial time of slot sl.

Although running in polynomial time, the known algorithms for the transportation problem may be very slow for our problem since the number of updates for usual applications is huge. In addition, in terms of minimizing the staleness, it seems more important to keep the size of the intervals between consecutive updates as uniform as possible than trying to exactly match the times programmed so far. This last observation is illustrated in Figure 2, where we consider a page that is programmed to be updated at times $T/12$, $4T/12$, $7T/12$ and $10T/12$. If the slots corresponding to these times are not available (shaded columns in Figure 2), then the approach proposed in [8] tries to assign these updates as close as possible to the programmed times. This may lead to Solution 2 illustrated at the bottom of Figure 2, where the crosses indicate the slot assigned to each update. As we can observe, this solution does not distribute the updates in a uniform fashion through-

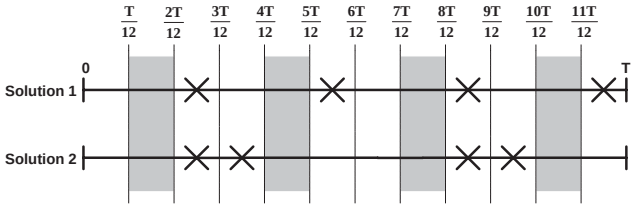

Figure 2: A potential weakness for the approach proposed in [8] for the crawler assignment problem. The shaded columns correspond to the non available slots and the crosses indicate the slot assigned to each update.

out the time horizon, which is bad for the objective function. On the other hand, the Solution 1 presented at the top of the figure provides a much more uniform distribution.

3 Theoretical Analysis

In this section we discuss a theoretical motivation behind the scheduling policy proposed in the previous section. This motivation can be translated into the following theorem.

Theorem 1 *The cost of the optimal solution of the problem $\mathcal{S}$ is a lower bound on the best staleness achieved by a revisiting schedule that respects the politeness constraint.*

The importance of this theorem relies on the fact that it shows that the cost of the optimal solution for $\mathcal{S}$ is a lower bound on the performance of any scheduling algorithm which aims to minimize the staleness without violating the politeness constraint. This lower bound may be helpful to measure the performance of other algorithms that one may propose. In fact, in Section 4 we employ this lower bound to measure the quality of the scheduling policy presented in this paper.

Theorem 1 is also important to justify the approach employed by our algorithm. Indeed, our algorithm starts with an optimal solution for the problem $\mathcal{S}$, where the politeness constraint does not need to be fully satisfied, and in the second phase it adjusts this solution so as to obtain a new solution which completely satisfies this constraint.

3.1 Proof of Theorem 1

Let B^* be the minimum staleness that can be achieved for the repository without violating the politeness constraint and let $x_{s,i}^{opt}$ be the number of updates performed by page (s,i) in a scheduling of staleness B^*. In addition, let $t_{s,i}(j)$ be the time of the jth update of page (s,i) in this scheduling. We have that

$$B^* = \sum_{s=1}^{m} \sum_{i=1}^{n_s} a_i^s \left(t_{s,i}(1), \ldots, t_{s,i}(x_{s,i}^{opt}) \right).$$

Now, let $x_{s.i}^*$ be the number of updates assigned to page (s,i) in the optimal solution of $\mathcal{S}$. Thus, it remains to show that

$$\sum_{s=1}^{m} \sum_{i=1}^{n_s} w_i^s A_i^s(x_{s,i}^*) \leq \sum_{s=1}^{m} \sum_{i=1}^{n_s} w_i^s a_i^s \left(t_{s,i}(1), \ldots, t_{s,i}(x_{s,i}^{opt}) \right). \quad (4)$$

Since the solution $x_{s,i}^{opt}$, for each (s,i), is a feasible solution for problem $\mathcal{S}$, it follows that

$$\sum_{s=1}^{m}\sum_{i=1}^{n_s} w_i^s A_i^s(x_{s,i}^*) \leq \sum_{s=1}^{m}\sum_{i=1}^{n_s} w_i^s A_i^s(x_{s,i}^{opt}). \tag{5}$$

In addition, we have

$$A_i^s(x_{s,i}^{opt}) \leq a_i^s(t_{s,i}(1), \ldots, t_{s,i}(x_{s,i}^{opt})),$$

for every page (s,i), since by definition $A_i^s(x_{s,i}^{opt})$ is the minimum staleness achievable by page (s,i) under the constraint that it is updated at most $x_{s,i}^{opt}$ times. From this last inequality and from (5) we conclude that inequality (4) holds, which establishes our result. ■

4 Experiments

In order to evaluate the quality of the scheduling policy proposed here, we performed some experiments where we simulate both the modification and download times of the web pages.

We considered a time horizon of 24 hours ($T = 24$), where $C = 10$ crawlers, with uniform speeds, are in charge of maintaining up to date a repository of 1 million pages, distributed in 2000 servers. We assume that each crawler can download 10 pages per second which implies in up to 864000 downloads in our time interval. Based on that, we set the capacity R to 864000×10. Furthermore, each crawler ensures an interval of 15 seconds between requests to the same server. For the time between consecutive modifications of a page we adopted an exponential distribution, that is, the probability of the next modification of page i occurs within the next t units of time is

$$1 - e^{-\lambda_i t},$$

where λ_i^{-1} is the mean time between consecutive modifications of page i. Although this distribution is not the most suitable one for some cases [8], it reflects reasonably well the behavior of most web pages [1, 4, 2].

For the exponential distribution, we have that [8]

$$A_i(x_i) = 1 + \frac{x_i + 1}{\lambda_i T}(e^{-\lambda_i T/(x_i+1)} - 1).$$

Unless we state the opposite the reader shall assume the following settings in our experiments: (i) the pages are distributed in the servers according to a Zipf distribution [9] with parameter $\theta = 1$ (many pages concentrated in a few servers); (ii) the pages have uniform weights and (iii) the λ_i parameters of the exponential distribution are uniformly distributed in the interval $[10^{-2}, 1]$.

In all of our experiments the weighted staleness achieved by our policy, referred as PSP (Polite Scheduling Policy) in the remaining of this paper, is compared with the weighted staleness of two other policies named MP2 and Modified Wolf (M-Wolf for short) and also with three theoretical bounds, NonPolite, InfCrawlers, and Weak Politeness (WP).

- MP2 is a policy similar to PSP except for the fact that instead of executing the MERGE procedure (Phase 2), MP2 distributes the $x^*_{s,i}$ updates assigned to (s, i), at the end of Phase 1, in equally spaced interval throughout the time horizon $[0, T]$. The motivation behind MP2 is checking the effectiveness of Procedure MERGE by comparing it with another reasonable strategy, that is, the one implemented by MP2.

- M-Wolf is a policy that first computes the number of updates per page and the times for these updates using the approach of [8] and, then, it runs Phase 3 of PSP to assign updates to crawlers. It shall be observed that the execution of Phase 3 ensures that M-Wolf respects the politeness constraint.

- InfCrawlers is the weighted staleness achievable if we would be able to exactly follow the scheduling times defined by MERGE procedure. This metric is useful to investigate the quality of the assignment strategy proposed at Phase 3 – if the weighted staleness of PSP gets close to InfCrawlers we can conclude that Phase 3 of PSP successfully assigned the programmed updates to the crawlers.

- WP is the cost of the optimal solution for problem $\mathcal{S}$. Recall that WP provides a lower bound on the optimal weighted staleness. The closer the staleness of the solution provided by PSP gets to this bound, the better it is.

- The NonPolite metric is the minimum weighted staleness that can be obtained if a large enough number of crawlers are allowed to perform R downloads along the time interval $[0, T]$ without respecting the politeness constraint. This is the bound reported in the experiments of [8]. The NonPolite metric is interesting to measure the price of being polite, that is, how much we pay in terms of staleness for respecting the politeness constraint.

The graph of Figure 3 is obtained varying the parameter θ of the Zipf's distribution that defines the distribution of pages on servers. We observed the following results.

1. The staleness achieved by PSP is smaller than the one obtained by MP2 which indicates that the way MERGE sets the update times in Phase 2 provides a non-negligible gain.

2. PSP staleness got extremely close to the InfCrawlers metric. This result is quite interesting since it suggests that the simple assignment strategy of Phase 3 cannot be improved using the more sophisticated and time consuming approach of [8].

3. PSP's staleness got close to the lower bound WP, 3% above in the average, which indicates that PSP had a performance very close to the optimal one.

4. The staleness of M-Wolf was 9.5% above the staleness of PSP in average. This significant difference occurred because some of updates scheduled by M-Wolf could not be executed in the time interval [0,T], otherwise the politeness constraint would be violated.

5. The difference between WP bound and NonPolite bound gets bigger as the distribution becomes more biased in favor of many pages concentrated in a few servers, a typical scenario in the Web. This is somehow expected since the politeness constraint exerts a strong influence in this case.

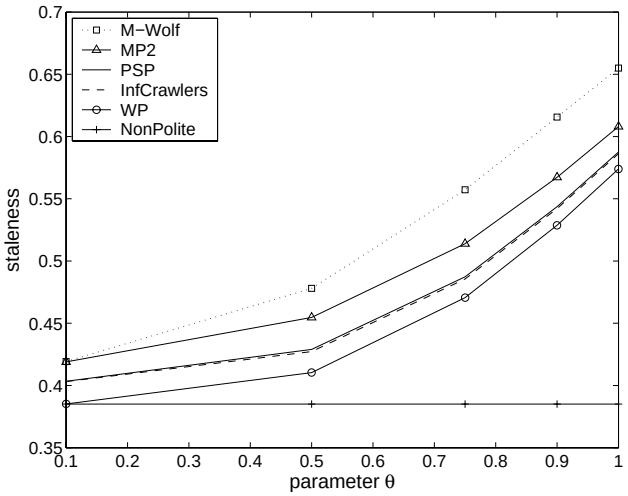

Figure 3: Weighted staleness of the repository as a function of the parameter θ of the Zipf distribution used to sample the number of pages per server.

The graph of Figure 4 is obtained varying the way we set the λ_i's of the exponential distribution. The values of the λ_i's were selected according to a Zipf's distribution, where a higher value of θ parameter indicates that a few pages are quite often modified. Again, the staleness achieved by PSP is better than those obtained by M-Wolf and MP2, close to the WP lower bound and extremely close to the InfCrawlers bound. We also ran some experiments where we varied the page weights. Since the results observed are similar to those reported in the other experiments we omit its details.

Recall that the capacity R is an input for our policy. In the experiments described above we set R as the product between the number of crawlers C and the number of updates a crawler can perform in the interval $[0, T]$ ($R = 8.64 \times 10^6$ for our default parameters). We ran some experiments to measure the sensibility of our algorithm w.r.t. the choice of R. The results are presented in Figure 5. If we set R smaller than the real capacity, then the staleness of the repository gets larger since the crawlers perform less updates than they could. The staleness also gets larger when R grows above the real capacity, however, for other reasons. The main one is that less important updates, that is, updates that would not be scheduled if we had set R to the real capacity, are selected instead of more important ones. The conclusion of this experiment is that R must be carefully selected in order to minimize the staleness.

We conclude this section mentioning that both Phases 1 and Phase 2 of our policy took just a few seconds to execute in a Pentium III Xeon 2.4 GHz with 1 GB of RAM memory. The time for assigning an update to a crawler at Phase 3 is very low due to its $O(\log n)$ time complexity – this constrasts with the network flow method of [8] that is very time consuming (at least its direct implementation).

5 Conclusions

In this work we proposed an efficient scheduling policy for revisiting web pages so as to keep a local repository as up to date as possible. In addition, we presented a simple way

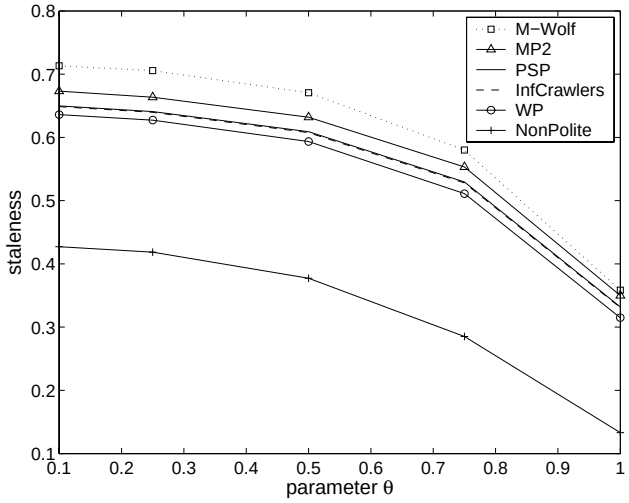

Figure 4: Weighted staleness of the repository as a function of the parameter θ of the Zipf distribution used to sample the rate of change λ_i of each page i.

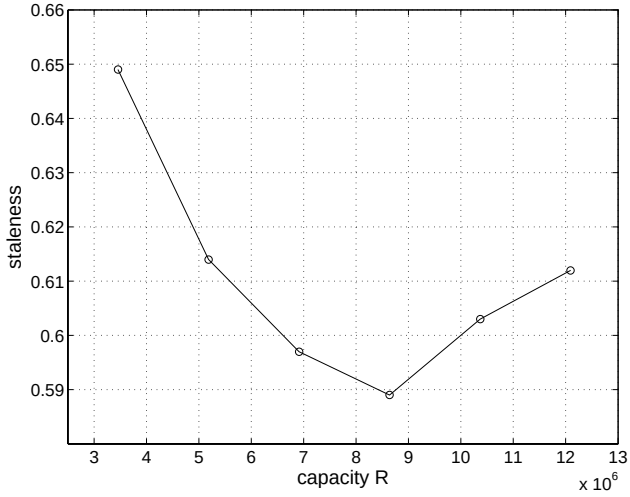

Figure 5: Weighted staleness of the repository as a function of R.

to compute a lower bound on the best staleness that can be achieved. To the best of our knowledge this is the first policy presented in the literature that takes into account the politeness constraint to compute the number of updates per page and the updates times.

The results of our experiments suggest that the performance of our policy is very close to the optimal one and it is significantly better than the one obtained by the policy proposed in [8].

References

[1] B.E. Brewington and G. Cybenko. How dynamic is the Web? *Computer Networks*, 33(1-6):257–276, 2000.

[2] BE Brewington, G. Cybenko, D. Coll, and NH Hanover. Keeping up with the changing Web. *Computer*, 33(5):52–58, 2000.

[3] Carlos Castillo, Mauricio Marin, Andrea Rodríguez, and Ricardo Baeza-Yates. Scheduling algorithms for Web crawling. In *Latin American Web Conference (WebMedia/LA-WEB)*, pages 10–17, Riberao Preto, Brazil, October 2004. IEEE CS Press.

[4] Junghoo Cho and Hector Garcia-Molina. Effective page refresh policies for web crawlers. *ACM Trans. Database Syst.*, 28(4):390–426, 2003.

[5] J. Eckstein, A. Gal, and S. Reiner. Optimal Information Monitoring under a Politeness Constraint. Technical report, Tech. Rep. RRR 16-2005, RUTCOR, Rutgers University, Piscataway, NJ, 2005.

[6] A. Federgruen and H. Groenevelt. The Greedy Procedure for Resource Allocation Problems: Necessary and Sufficient Conditions for Optimality. *Operations Research*, 34(6):909–918, 1986.

[7] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[8] J. L. Wolf, M. S. Squillante, P. S. Yu, J. Sethuraman, and L. Ozsen. Optimal crawling strategies for web search engines. In *WWW '02*, pages 136–147, New York, NY, USA, 2002. ACM Press.

[9] G.K. Zipf. *Human Behavior and the Principle of Least Effort: An Introduction to Human Ecology.* Addison-Wesley Press, 1949.