Towards a Content-Provider-Friendly Web Page Crawler *

Jie Xu Qinglan Li Huiming Qu Alexandros Labrinidis

Advanced Data Management Technologies Laboratory
Department of Computer Science, University of Pittsburgh
Pittsburgh, PA 15260, USA
{xujie, qinglan, huiming, labrinid}@cs.pitt.edu

ABSTRACT

Search engine quality is impacted by two factors: the quality of the ranking/matching algorithm used and the freshness of the search engine's index, which maintains a "snapshot" of the Web. Web crawlers capture web pages and refresh the index, but this is always a never-ending quest, as web pages get updated frequently (and thus have to be re-crawled). Knowing when to re-crawl a web page is fundamentally linked to the freshness of the index, given the size of the Web today and the inherent resource constraints: re-crawling too frequently leads to wasted bandwidth, re-crawling too infrequently brings down the quality of the search engine.

In this work, we address the scheduling problem for web crawlers, with the objective of optimizing the quality of the index (i.e., maximize the freshness probability of the local repository as well as of the index). Towards this, we utilize feedback from the users (content providers) on when their web pages are updated and consider the entire spectrum of collaboration, from no feedback to explicit update schedules. We propose a unified online scheduling algorithm which utilizes different levels of collaboration from content providers. Extensive experiments with real web traces demonstrate that cooperation from users plays a major role in improving search engine index quality.

1. INTRODUCTION

Major web search engines are the most important places that content providers would like to be listed because people always turn to search engines to find the specific web page out of the mass information. Search engines can bring the content providers with large amount of traffic to increase their popularity.

Web crawlers are employed to capture and refresh the web pages in a search engine's local repository, where a keyword-based index is built for search. There are two factors that are of paramount importance to the quality of search engines: (1) the freshness of the search engine's index, and (2) the quality of the ranking/matching algorithm that sorts the search results based on the relevance to the keyword(s). The two factors are both challenging, yet orthogonal to each other. In this paper, we concentrate on (1), that is, how to optimize the freshness of the index so that the search results reflect the most up-to-date information from the original web sites.

Since web pages are frequently updated by the content providers, freshness of the search engine's index is always "endangered" and crawling is a never-ending procedure. Given the size of today's web, as well as the inherent resource constraint, the exact synchronization between the index (as well as local repository) and the live web is hard to achieve. However, if the search engines have more information about the update schedule of the content providers, the crawling decision can be made easier. For example, a stock information web site periodically refresh their web pages; or a NFL team's web site at least will update its web pages after each game. Most commercial web sites have such routine update schedules, which could be used to collaborate with search engines to enhance the efficiency and effectiveness of the crawling schemes. Since the schedule of re-crawling is important to content providers as well, (re-crawl too frequently lead to the waste of content providers' bandwidth, and re-crawl too infrequently may damage content providers' popularity), collaboration definitely gives search engines and content providers a win-win situation.

In this work, we concentrate on the scheduling problem for web crawlers, with the objective of maximizing the repository freshness. Our main contributions are as follows:

- We propose to utilize feedback from the users (content providers) on when their web pages are updated and consider the entire spectrum of user cooperation (e.g., from no feedback to full update schedules).
- We propose a unified online scheduling algorithm which considers the negative impact (i.e., degradation if the page is not crawled) while incorporating content providers' multi-level collaboration.
- We implement the web crawler framework and evaluate our algorithm in various settings. Extensive experiments with real web traces demonstrate that cooperation from the users can play a major role in improving search engine index quality.

The remainder of this paper is organized as follows. Section 2 summarizes the related work. Section 3 describes our

*Funded in part by NSF ITR Medium Award ANI-0325353 and NIH-NIAID grant NO1-AI50018.

Proceedings of the 10th International Workshop on Web and Databases (WebDB 2007), June 15, 2007, Beijing, China.

system, quality metric, and user profiles. Section 4 elaborates our unified online scheduling algorithm and priority score calculation. In Section 5, we present our empirical experimental results with real web traces. Finally, we conclude and discuss our future plans in Section 6.

2. RELATED WORK

Regarding how much information the crawler has, different researchers made different assumptions. However, as far as we know, no work has considered a unified algorithm to incorporate different levels of collaboration between the search engines and the web servers being crawled.

Pandy and Olston [7] assume no feedback from content providers. Their scheduling algorithm aims at maximizing the user-centric search repository quality, which includes addressing page ranking, index maintenance and repository freshness. In our case, we focus on the repository freshness problem, and leave index maintenance and page ranking for our future work.

Wolf et al. [9] consider two cases, one is without user feedback, the other is with fixed potential time points for updates (along with a probability for the update to happen), which they called *quasi-deterministic*. Their objective is to minimize the "embarrassment" of search engines (i.e., when the web page that a user gets is not relevant to his/her query). Towards this, they first apply probability theory and theory of resource allocation problems to find out the theoretically optimal crawling times, then map the scheduling problem to a *transportation problem* based on *network flow theory*. Although the algorithm complexity is polynomial time in deterministic cases, given the huge input dimension, it can not be used as an online algorithm (as ours).

Both [2] and [6] address the cache synchronization problem, which is very close (and can be mapped) to the crawler scheduling problem. Cho and Garcia-Molina [2] assume that the synchronization is uniform over time and the order over elements is fixed, then concentrate on the refreshing resource allocation problem. Their algorithm uses *method Lagrange multipliers* to get the optimal results. As an opposite, we assume the crawling capability is known, and we integrate resource allocation with the scheduling algorithm, since they are highly interconnected. Olston and Widom [6] assume that every data source cooperates with caches actively, which is extremely hard to achieve in the crawler scheduling problem considering the size of today's web.

In calculating priority scores, we essentially borrow ideas from the previous work of one of the authors [5] and combine with the divergence graphs from the work of [6], to illustrate the negative impact of a web page, if it is not scheduled to be re-crawled. Finally, we also incorporate the popularity of a web page as an important weight factor for the scheduling of re-crawls.

3. SYSTEM

In this section, we present the basic search engine architecture, describe our quality metric, and then classify the user profiles based on their different levels of collaboration.

3.1 System Overview

Web search engines are the gateways where most users look for specific web pages from the huge internet repository. In general, they employ multiple crawlers to grab the

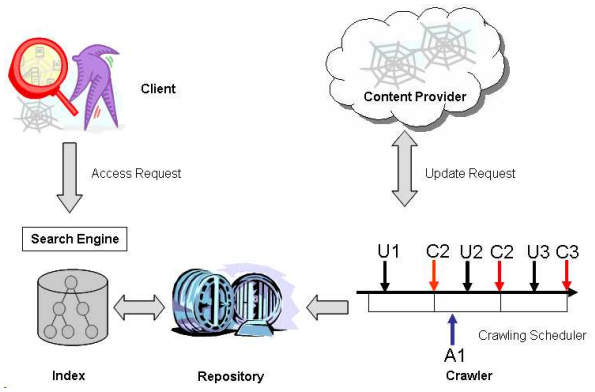


Figure 1: General Web Search Engine Architecture

"snapshots" of live web pages all over the world. The snapshots are stored in search engines' local repositories. Then an index is built up over the local repository (as in Figure 1). A query is normally in the form of a keyword set (probably with priority). When clients submit queries, the search engine will first match the keyword set with its inverted index, and then apply a page ranking algorithm [1, 3, 4] to rank each web page that has passed the filtering from the first step (i.e., is considered relevant to the query). Finally, the search engine returns to the user a prioritized list of links, which point the user to the live web pages. The higher the link's position in the list, the higher the relevance to the user's query.

In this architecture, the users' experience depends highly on the quality of the index, which is in turn decided by the freshness/precision of the local repository. The ideal case is that the local repository owns exact snapshots of live web pages. However, this is hard to achieve since (1) web pages are frequently updated, and (2) the crawling capability is constrained.

Even with significant crawling capacity, the crawling scheduler is still a necessary module. For example, as in Figure 1, the crawling scheduler tries to schedule three available crawling slots. Although the crawling capability is enough (equal number of updates and crawling slots), access of web page 1 ($A1$) still gets a stale snapshot, as the scheduler chooses to re-crawl web page 2 ($C2$) at the first crawling slot. Thus, an accurate and efficient web crawl scheduling algorithm is of vital importance to minimize the divergence between local repositories and the live web.

3.2 Quality Metric

As we mentioned before, the quality of the index depends heavily on the freshness of the local repository. We define the quality as an aggregated freshness degree over the entire local repository. In other words, we measure the probability that an index access comes from a fresh (up-to-date) web page in the local repository $\mathbf{R}$, which we denote as $P_{fresh}(\mathbf{R})$.

$P_{fresh}(\mathbf{R})$ is decided not only by the freshness, P_{fresh}, of the web pages but also by the access frequency, f_{access}, of the web pages in $\mathbf{R}$. We measure the overall freshness as the weighted sum of the freshness of each web page in the local repository [5], where the weight is the access frequency of the corresponding web page, as shown in the following.

$$P_{fresh}(\mathbf{R}) = \sum_{p_i \in \mathbf{R}} (f_{access}(p_i) \times P_{fresh}(p_i)) \qquad (1)$$

$$f_{access}(p_i) = \frac{\text{Number of accesses on } p_i}{\text{Total number of accesses on } \mathbf{R}} \quad (2)$$

$$P_{fresh}(p_i) = \begin{cases} 1 & \text{if there is no missing update} \\ \frac{1}{\sum_m m \times (t_{now} - t_m)} & \text{if there is missing update} \end{cases} \quad (3)$$

where $f_{access}(p_i)$ is the access frequency of page p_i and $P_{fresh}(p_i)$ is the freshness of page p_i. Freshness degradation is computed with the difference of current time and each missed update m times the number of missed update, assuming time is an ascending real value and current time t_{now} is the largest number on the time axis. $P_{fresh}(p_i)$ is inverted proportionally to the weighted sum of freshness degradation caused by each missed update.

The freshness of the repository, $P_{fresh}(\mathbf{R})$, is a real number between 0 and 1 since both $f_{access}(p_i)$ and $P_{fresh}(p_i)$ scales from 0 to 1. The higher the value of $P_{fresh}(\mathbf{R})$, the better overall quality of repository $\mathbf{R}$.

3.3 User Profiles

An effective web page crawl scheduler depends on four different types of information: Access History, Crawling History, Past Update Time Points, and Future Update Time Points. Access history and crawling history are all collected from the search engine site, whereas the update information depends on how much content providers can provide. In this paper, we classify three types of users (i.e., content providers) based on the collaboration levels with the search engines:

- *Normal User*: the web site, such as a personal home page, that doesn't provide any information about updates.
- *Smart User*: the web site that sends a signal every time it has been updated, so that the crawler can record all its past update time points (i.e., its update history).
- *Super User*: the web site that provides not only its update history, but also its future update schedules (with the probability that an update will happen). Examples are commercial service providers such as stock information providers, which have routine update schedules.

Other than Normal Users, both Smart Users and Super Users cooperate with search engines to some extent by providing partial or full update information. We summarize the different user types with various information in Table 1.

4. PROPOSED ALGORITHM

In this section, we elaborate on our online scheduling algorithm for web crawlers with different levels of collaboration. The objective is to maximize the freshness of the pages stored/indexed in the search engines' local repositories, while also considering the pages' access probability.

4.1 Scheduling Re-crawls

The search engine assigns priority scores to web page updates from all web sites. For the normal user or the smart user, only one priority score is assigned to the most recent update that has happened, whereas for the super users, in addition to the most recent update, each possible future update will be assigned a priority score as well.

These prioritized past updates (e.g., the most recent update happened) and future updates are put into a priority queue where the available crawler is always assigned to the update with highest priority score. Assuming we have M crawlers, each has the crawling capacity to crawl P pages per crawling cycle. Then we have totally M × P time slots to schedule at the beginning of each crawling cycle. For the past update, the current available crawling slot will be assigned. For the future update, the first available crawling slot after the future update time will be assigned.

4.2 Priority Scores

We calculate priority scores for each web page update according to its negative impact which is the freshness degradation if the page is not re-crawled. The bigger the negative impact, the higher the priority score, and the sooner the web page will be re-crawled. We distinguish three cases:

- **Known Update History**
  For web pages with known update histories, we estimate the negative impacts by timing up the number of accesses that hit the stale web page and the time duration that the web page stayed stale. Assume N_t is the number of accesses that hit the stale web page from the last crawl to time t, and t_m is the time when the m_{th} missed update happens for the web page.

  $$S_{past}^{known} = \sum_{m=0}^{U} ((t_m - t_{m-1}) \times N_{t_m}) \quad (4)$$

  where U is the total number of missed updates, and N_{t_m} is the number of stale accesses from the last crawl to the time of the m_{th} missed update, which is incrementally increased as more missed updates are counted.

- **Unknown Update History**
  For web pages with unknown update history, since we have no idea about past update time points, we estimate the negative impact with an area of a triangle:

  $$S_{past}^{unknown} = \frac{1}{2} \times (t_{now} - t_{LC}) \times (N_{t_{now}} + c) \quad (5)$$

  where t_{LC} is the time of last crawl and c is a small constant to adjust the difference between Equation 4 and the triangle area.

- **Future Update**
  Web pages with future update information provide the future update schedule with the probability, P_{update}, that each update will happen. We calculate one priority score for each future update. Similar to the priority score of S_{past}^{known} and $S_{past}^{unknown}$, we use the probability of future access to reflect the importance of each future update. As we don't have future access information, we estimate the future access, C_p for the pth cycle, by using the access history of the past s crawling cycles. In our experiments, s equals to 5. We apply the aging scheme to emphasize the recent histories as shown in the following.

  $$S_{future} = P_{update} \times \frac{\alpha \times C_{p-1} + (1-\alpha) \times \sum_{d=p-s}^{p-2} C_d}{\sum_{d=p-s}^{p-1} C_d} \quad (6)$$

User Type	Normal	Smart	Super
Access History	Yes	Yes	Yes
Crawling History	Yes	Yes	Yes
Past Update Time Points	No	Yes	Yes
Future Update Time Points	No	No	Yes
Priority Score	$S_{past}^{unknown}$ (Eq. 4)	S_{past}^{known} (Eq. 5)	$S_{unified}$ (Eq. 7)

Table 1: User Types

Having computed priority scores based on different levels of available information, we summarize how to use the priority scores for different users in Table 1. For normal users and smart users, the priority score of each web page will be the $S_{past}^{unknown}$ and S_{past}^{known} we calculated in Equation 5 and Equation 4 respectively, normalized by the total number of accesses. For super users, the priority score comes from two parts, past and future. For each future update point, we calculate the weighted sum of normalized S_{past}^{known} in Equation 4 and the S_{future} in Equation 6 which has been normalized, as

$$S_{unified} = \beta \times S_{past}^{known} + (1 - \beta) \times S_{future} \quad (7)$$

where β balances the relative impact of past information versus future information.

5. EXPERIMENTAL STUDY

The objective of our experiments is to investigate the impact of different environmental settings to our algorithm using real web traces. The goal of our algorithm is to maximize the aggregated freshness of the local repository, which decides the quality of search engine index consequentially. In addition, we evaluate the stability and scalability of our algorithm under various workloads.

5.1 Experimental Setup

The experimental analysis leverages on a prototype framework built for this project. The system integrates three types of user profiles with the online scheduling algorithm, according to priority scores computed by the equations introduced in Section 4.2.

Update Trace We have been crawling the RSS feeds for some popular news source (CNN, NY Times, Yahoo, etc) every 30 minutes since Aug 2005. After an initial data cleaning process to remove noise, these data sets are ready to be used as the update traces. We observed that there are no significant update patterns during the entire time period, so we extracted update requests from a two-month period as the update trace in our experiments.

Access Trace Using the same set of web pages from our update trace, we generated synthetic access requests following Zipf and Poisson distributions, on the accessed web pages and the time intervals between accesses.

Architecture Update requests from multiple users (content providers) and access requests are parsed into the framework, after going through a filter which removes unrelated requests; then the ranking engine computers priority scores for each web page and each update request (for Super Users). The scheduler then dispatchs crawling jobs according to the priority scores. All experiments were run multiple times and figures reflect the average statistics from these runs. The machines we used in our simulation is a Dell PowerEdge

Parameter	Value
Date Period	Nov 1, 2006 - Dec 31, 2006
Update Frequency	Max 2928 times per page
Access Size	20k bytes - 50k bytes
Access Zipf θ over page	0.85
Access Zipf θ over time	0 - 1
α (refer to Eq. (6))	0.8
β (refer to Eq. (7))	0.8
s (refer to Eq. (6))	5
P_{update} (refer to Eq. (6))	0 - 1

Table 2: Simulation Parameters

2950 server with two Dual-Core Intel Xeon 3GHz CPU processors and 4MB L2 cache. It is equipped with 16GB RAM and 1.5TB hard disk, running Redhat Linux Version 3.4.6-3.

The values of the major parameters are shown in Table 2. Although we explored the algorithm sensitivity to parameter values within the stated range, we present here experiments only on the values specified in Table 2, unless otherwise noted. We present results with the α and β fixed to 0.8, and we have additional evidence suggesting that results are fundamentally the same with variable access trace sizes.

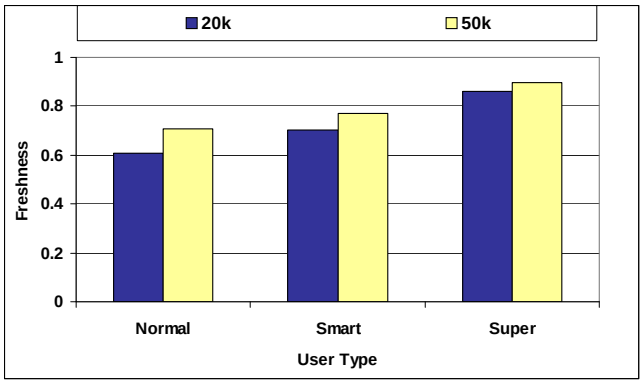


Figure 2: Freshness degree in a homogeneous environment; all users are of a single type (normal, smart, super) in each experiment. Smart user outperforms normal user by 10%, super user outperforms smart user by 15%.

5.2 Varying Environment

5.2.1 Homogeneous Environment

First, we create a homogeneous environment, in which only one type of content providers exists. We ran our experiments on two different sizes of access traces to test the scalability of our algorithm, the larger size access (50k) and the smaller size access (20k). The aggregated freshness over the repository **R** are reported in Figure 2.

Not surprisingly, the more information content providers give the crawler, the better quality the crawler achieves. Specifically, in the 20k trace, the freshness of smart users outperforms normal users around 10%. With future update time point information available for super users (even with a probability to happen), the quality is improved another 15% comparing to smart users, where only past history is known. The trend is the same for the 50k trace, although

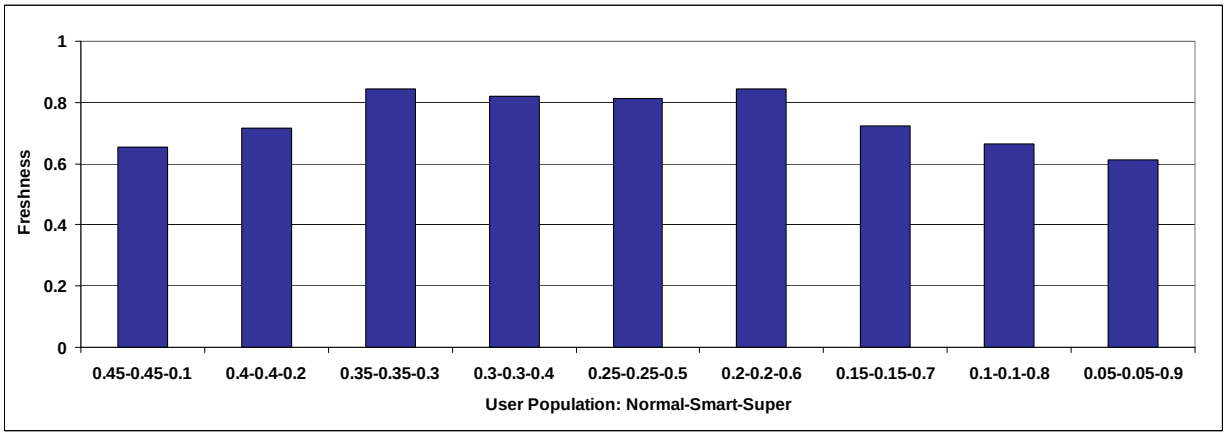


Figure 3: Freshness Degree under Various User Populations (Normal vs Smart vs Super); by having near 30% of the population be super users, the crawler achieves the best overall freshness.

the scale varies a bit. Since the trend for the 50k trace is always close to the 20k trace, we only report results from the 20k trace for the remaining experiments.

5.2.2 Heterogeneous Environment

In the second set of experiments, we mix three types of content providers together, and watch the overall freshness degree under various user population compositions. We modify the population of the super users, ranging it from 0.1 (10%) to 0.9 (90%) to make the spectrum complete, and allocate the remaining population evenly into normal and smart users. The results are shown in Figure 3.

From a first look at Figure 3, we might think that there is no clear trend to predict the aggregated quality over the user population. However, when we closely observe the result, it does present a trend: the best performance is achieved when super users take near 30% and 60% of the population respectively. Before the super users' population reaches 30% (phase 1), the performance increases monotonically. And after it gets to 60% (phase 3), the performance decreases monotonically. Between these two points (phase 2), the performance varies a bit, but not as large as the others.

The reason that freshness increases in phase 1 is that as super users increase, the more information is provided, and the overall scheduling quality is in turn improved. However, as the super users' population reaches 30%, the crawler has enough information to achieve the best performance, that is why the freshness remains the same in phase 2. Moreover, when super users take more than 60% of the population, they provide more information than the crawler can utilize, and as such they start competing with each other, taking up the local maximum time slots instead of the global maximum and bringing down the overall freshness no matter how other users perform.

An interesting point is when we compare the super user in the homogeneous environment, the freshness is actually higher than all the freshness in phase 3, and almost the same as freshness in phase 2. This is an unexpected result. Our guess is since all users behave the same in a homogeneous environment, they will eventually reach a balanced status so that the overall performance will not be affected much by the competition. However, in a heterogeneous environment, there are other types of users also. As a result, they will probably become the victims of super users' competition, so that their crawling slots were taken by super users, however, which does not bring a positive impact on the overall freshness.

In summary, results from the heterogeneous environment demonstrate that we do not need everyone to be a super user to achieve the best overall performance. If we get 30% of users to fully collaborate, we will be able to get the best possible freshness, however, even small percentages will lead to an improvement compared to no collaboration. With many of the content providers having a regular update schedule (e.g., news feeds), we expect such collaboration to be relatively easy.

5.3 Varying Workload

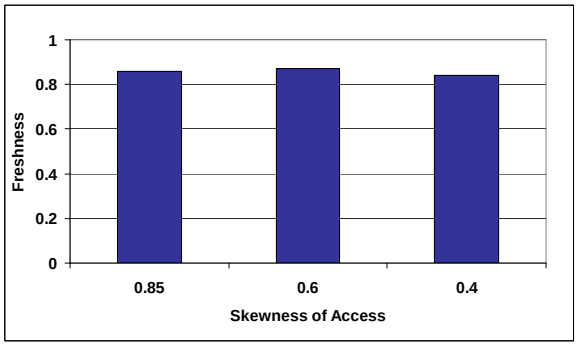
Having observed the performance of our algorithm in different environments, we now observe the impact of different workloads.

First, we test different access patterns over time by changing the θ value in the Zipf distribution to adjust the access skewness. We choose the most commonly used θ value 0.85 to represent the highly skewed access pattern, 0.6 to represent a moderately skewed pattern, and 0.4 to represent an almost unskewed access pattern. As shown in Figure 4(a), the freshness is almost unchanged, which means our algorithm is stable to different access patterns.

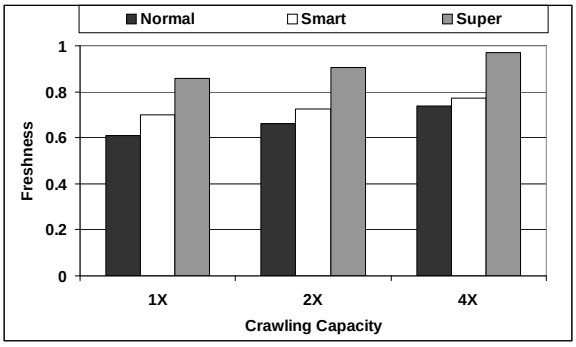
In the second set of experiments, we enlarge the crawling capacity, and observe the change of aggregated freshness. The results are shown in Figure 4(b). As expected, increasing the crawling capacity improves the freshness. The larger the crawling capability, the bigger performance improvement super users achieve compared to normal/smart users. Another observation is that as the original crawling capacity is large enough (smart users achieve around 86% freshness), doubling/four-fold increase of the crawling capacity does not lead to a linear increase on the overall freshness.

5.4 Varying Future Update Info. Accuracy

In the last experiment, we tested the impact of super users' prediction precision. We vary the probability of super users' precise prediction from 1 (100% precise) to 0.25 (25% precise), and report the overall freshness in Figure 5.



(a) Access Skewness over Time (i.e., different θ value)



(b) Crawling Capacity: original - double - four times

Figure 4: Freshness Degree under Various Workload: our scheduling algorithm is stable enough to handle different access skewness and different crawling capacity.

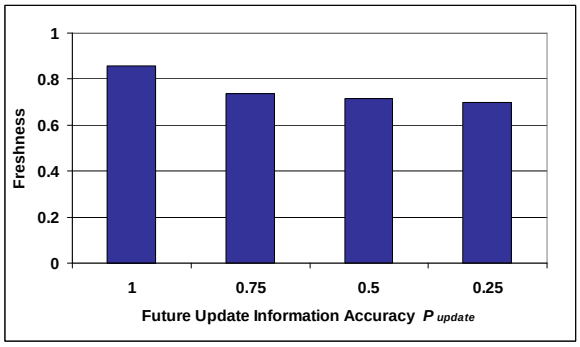


Figure 5: Freshness Degree under different Super Users' Prediction Precision: the more accurate of super users' future update information, the better the aggregated freshness.

As we expected, the more precise information content providers give to the crawler, the better crawling quality can be achieved. The decrease on performance is large when precision dropped from 100% to 75%. After that, freshness does not drop as much as the first drop. The reason is that when a super user provides an inaccurate future update time point, and the update time point is successfully scheduled (which happens with high probability), one crawling slot is "wasted" if this was inaccurate, since the update does not happen (which leads to a stale web page being crawled). Moreover, this takes one slot from one of other users, that one of his/her web pages potentially can be refreshed, which makes performance even worse.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we addressed the scheduling problem for web crawlers, with the objective of optimizing the aggregated freshness degree of the crawler's local repository. We proposed an online scheduling algorithm which crawls web pages according to their negative impact, if they are not refreshed. Moreover, we utilize feedback from the content providers on when their web pages are updated and consider the entire spectrum of collaboration, from no feedback to explicit update schedules. Extensive experiments with real web traces demonstrate that cooperation from the users can greatly improve search engine index quality. As long as having near 30% content providers' collaboration, the crawler can achieve the best overall freshness. The stability and scalability of our algorithm are confirmed as well by our experimental results.

All our experiments are based upon the real update trace we collected and the synthetic access traces. In the near future, we plan to incorporate more real traces. We will also extend this work by building the inverted index and measure the direct user experience by taking into account the page ranking algorithms [1, 3, 4]. Finally, we would like to develop a user-centric scheduling algorithm based on the micro-economic paradigm (previous work in [8]).

7. REFERENCES

[1] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[2] J. Cho and H. Garcia-Molina. Synchronizing a database to improve freshness. In *Proceedings of SIGMOD '00*, pages 117–128, New York, NY, USA, 2000. ACM Press.

[3] J. Cho, H. García-Molina, and L. Page. Efficient crawling through URL ordering. *Computer Networks and ISDN Systems*, 30(1–7):161–172, 1998.

[4] J. Cho and S. Roy. Impact of search engines on page popularity. In *Proc. of the 13th international World Wide Web conference*, 2004.

[5] A. Labrinidis and N. Roussopoulos. Update propagation strategies for improving the quality of data on the web. In *Proceedings of VLDB '01*, pages 391–400, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[6] C. Olston and J. Widom. Best-effort cache synchronization with source cooperation. In *Proceedings of SIGMOD '02*, pages 73–84, New York, NY, USA, 2002. ACM Press.

[7] S. Pandey and C. Olston. User-centric web crawling. In *Proceedings of WWW '05*, pages 401–411, New York, NY, USA, 2005. ACM Press.

[8] H. Qu, J. Xu, and A. Labrinidis. Quality is in the eye of the beholder: Towards user-centric web-databases (demo). In *Proceedings of SIGMOD'07*, Bejing, China, 2007.

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