Design of a Crawler with Bounded Bandwidth

Michelangelo Diligenti
Dipartimento di Ingegneria dell'Informazione
Università di Siena, Italy
michi@dii.unisi.it

Marco Maggini
Dipartimento di Ingegneria dell'Informazione
Università di Siena, Italy
maggini@ing.unisi.it

Filippo Maria Pucci
Dipartimento di Ingegneria dell'Informazione
Università di Siena, Italy
puccifili@inwind.it

Franco Scarselli
Dipartimento di Ingegneria dell'Informazione
Università di Siena, Italy
franco@dii.unisi.it

ABSTRACT

This paper presents an algorithm to bound the bandwidth of a Web crawler. The crawler collects statistics on the transfer rate of each server to predict the expected bandwidth use for future downloads. The prediction allows us to activate the optimal number of fetcher threads in order to exploit the assigned bandwidth. The experimental results show the effectiveness of the proposed technique.

Categories and Subject Descriptors: H.3 [**Information Systems**]: Information Storage and Retrieval

General Terms: Design, Experimentation, Performance

Keywords: Parallel Web Crawlers, Bandwidth Optimization

1. INTRODUCTION

The design of efficient crawlers is a key issue in the design of Web search engines [1]. Whereas the crawlers of large search engines can exploit dedicated network connections, departmental search engines usually share the network bandwidth with other services. For this reason, bandwidth control is a key point of crawler design. The design of the bandwidth control policy must achieve two contrasting goals. First, the crawler should not hinder the users who share the connection; second, it should exploit completely the assigned network resources.

Recently, efficient crawling strategies have been studied either based on the importance of the documents [2, 4] or on the topic of the pages [3, 5]. *Focused crawlers* aim at maximizing the quality of the retrieved resources but do not consider the instant bandwidth in use. Instead, bandwidth control can be achieved by optimizing the number of parallel downloads [6].

In this paper, we propose a policy to control the bandwidth of a crawler, which is particularly suited for small focused search engines. The crawler collects statistical data on the transfer rates of each server, which are then used to decide the optimal number of parallel downloads.

2. THE ALGORITHM FOR BANDWIDTH CONTROL

We assume that each URL is assigned a score, which measures the estimated importance of the related document. The crawler selects the next URL to download by popping it from a priority queue sorted according to this score. In the case of focused crawlers, the score may represent the probability that the document is on a desired topic or that it is on a path leading to relevant pages.

In our design, the crawler is composed of a set of *fetchers* and a *fetcher manager*. Each module is run as a separate thread and the pool of fetcher threads are created upon startup. A fetcher waits until the manager activates it for retrieving a given URL. Then it performs an HTTP request to the appropriate Web server and, after having transferred the document, it returns to a wait state. The manager implements the crawler control policy by feeding appropriately the fetchers. The manager selects the URLs to be retrieved using their priorities and it activates the optimal number of fetchers to exploit the assigned bandwidth.

In order to select the next URL u, the manager estimates the bandwidth $B(u)$ which a fetcher is expected to consume by retrieving the document u. When the download is completed, the manager can measure the actual transfer rate $M(u)$ and use this value to update its estimate of the server speed. The manager can compute the expected total transfer rate of the crawler CB as $\sum_{u \in S} B(u)$, where S is the set of the documents that are currently assigned to the fetchers.

Assuming that the crawler has already activated the optimal number of fetchers, when a fetcher completes its current download u_c, a slot of bandwidth becomes available to start one or more downloads. Thus, the crawler bandwidth is updated as $CB = CB - B(u_c)$ and the manager searches the URL queue for a document u_d such that the estimated bandwidth will remain below the assigned threshold L while downloading it, i.e. $CB + B(u_d) \leq L$. The search starts from the top of the queue and continues up to a given maximum depth max_d, in order to consider only the documents with higher priority. If the search is successful, the manager activates a fetcher to download u_d, otherwise it stops and waits for the completion of another download. If the expected bandwidth of the new download does not consume completely the available slot, the queue is searched for further candidates. The algorithm to activate the fetchers is the following.

1. Set $d = 0$
2. Get the document u_d at depth d in the URL queue

Copyright is held by the author/owner(s).
WWW2004, May 17–22, 2004, New York, New York, USA.
ACM 1-58113-912-8/04/0005.

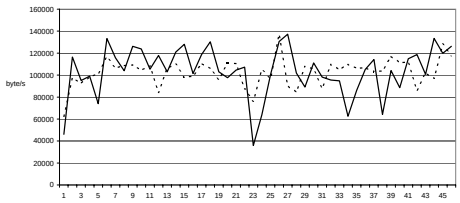

Figure 1: Measured bandwidth (continuous line) and predicted bandwidth (dotted line) in a forty seconds time window.

3. If $CB + B(u_d) \leq L$, assign u_d to an inactive fetcher and update $CB = CB + B(u_d)$

4. Set $d = d + 1$

5. Repeat step 2 until $CB \leq L$ or $d = max_d$

The manager periodically checks the queue to avoid the starvation of the fastest servers (e.g. those servers whose transfer rate exceeds the threshold L).

The statistics on the server transfer rates are stored in a lookup table indexed by the server IP address. The transfer rate is measured dividing total number of transferred bytes by the time interval from the issue of the HTTP request till the completion of the document download. This measure considers both the network delivery latency and the server response speed. Thus, the measured transfer rate depends both on the characteristics and state of the network links connecting the machine running the crawler to the server and on the speed and current load of the server machine itself. Since the network state and the server load can vary during time, we decided to model this dependency by assuming a hourly distribution: we estimate the bandwidth $B(IP, h, w)$ for each hour $h = 0, \ldots, 23$ both for working days ($w = 1$) and holidays ($w = 0$). This yields a total of 48 values for each server. We found experimentally that this assumption guarantees sufficient accuracy in the estimates. We decided to use a discrete distribution to reduce the algorithm complexity.

The values $B(IP, h, w)$ are updated after each download by using the measured bandwidth $M(IP)$. If the measure is obtained at time t (in minutes) of the day, each estimate is updated according to

$$B(IP, h, w) = (1-\alpha(t-60h))B(IP, h, w)+\alpha(t-60h)M(IP),$$

where $\alpha(y) = ke^{y^2/\sigma^2}$, being k and σ design parameters, and w is chosen accordingly with the current date. This formula was derived assuming that the bandwidth measured at time t affects the estimates for the parameters according to a Gaussian function $\alpha(t)$.

This approach to bandwidth prediction is feasible for small departmental and focused search engines, which are likely to repeatedly contact only a limited set of servers.

3. EXPERIMENTAL RESULTS

In order to evaluate the proposed algorithm, we decided to force the crawler to download the URLs contained in a predefined queue. This approach allows to compare the results since the same queue is used for all the tested settings.

In the first experiment, the bandwidth limit was set to $140KB/s$ and the actual bandwidth was measured every second. The parameters used to update the estimates were $k = 0.3$ and $\sigma = 120$, while the maximum search depth max_d was set to 6. Figure 1 shows that the predicted bandwidth (dotted line) is close to the actual bandwidth (continuous line) and it is always below the assigned threshold.

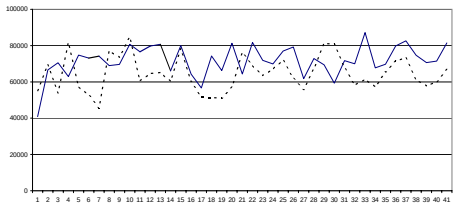

Figure 2: Measured bandwidth with $max_d = 6$ (continuous line) and with $max_d = 2$ (dotted line).

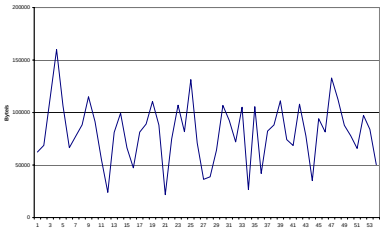

Figure 3: Measured bandwidth for a crawler with simple control on the number of fetchers.

The choice of the maximum search depth max_d affects the actual use of the available bandwidth, since by using a higher search depth it is more likely to find a download which fits the available slot. Figure 2 compares two runs of the crawler with $max_d = 2$ (dotted line) and $max_d = 6$ (continuous line). The experiment proves that a larger max_d improves the average bandwidth ($71,788$ for $max_d = 6$ versus $64,562$ for $max_d = 2$). However, when max_d increases, some URLs with a smaller score are downloaded before URLs with a lager score.

Finally, we compared our method with the one proposed in [6]. In this case, the bandwidth is controlled by stopping a fetcher when the actual bandwidth is above the limit and by starting a new one when it is below the limit. Figure 3 shows that the control is less accurate and the bandwidth oscillates considerably.

4. CONCLUSIONS

In this paper we have presented a technique to control the bandwidth of a crawler. The approach is particularly useful for small and focused search engines and the experiments, even if preliminary, have showed its effectiveness.

5. REFERENCES

[1] A. Arasu, J. Cho, H. Garchia-Molina, and S. Raghavan. Searching the web. *ACM Transactions on the Internet Technologies*, 1(1), 2001.

[2] J. Cho, H. Garcia-Molina, and L. Page. Efficient crawling through url ordering. In *Proceedings of the 7th International Conference on the World Wide Web*, 1998.

[3] M. Diligenti, F. Coetzee, L. Lawrence, C. L. Giles, and M. Gori. Focused crawling using context graph. In *Proceedings of the 27th Conf. on Very Large Data Base*, 2000.

[4] M. Najork and J. Wierner. Breath-first search crawling yields high quality pages. In *Proceedings of the 10th International Conference on the World Wide Web*, 2001.

[5] J. Rennie and A. McCallum. Using reinforcement learning to spider the web efficiently. In *Proocedings of the International Conference on Machine Learning*, 1999.

[6] V. Shkapenyuk and T. Suel. Design and implementation of a high-performance distributed web crawler. In *Proceedings of the 18th International Conference on Data Enginering*, 2002.