VISE 視覺化介面的查詢處理核心設計
Design of the Query-Processing Kernel in VISE Visualization Interface

楊正仁　　楊鎮州　　陳英祥
元智大學資訊工程學系
Cheng-Zen Yang, Chen-Cho Yang, and Ing-Xiang Chen
Department of Computer Science and Engineering
Yuan Ze University
Email: {czyang,steven,sean}@syslab.cse.yzu.edu.tw

Abstract

To help users find information of interest from large amount of Web pages, we have proposed a visualizing interface called VISE for amending the shortcomings existed in the list-based interface of most common Web search engine. From previous study, VISE is good at presenting the search results in a graphical interface and showing the hyperlink relationships between the result entries. However, VISE suffers from its lengthy processing time. In this paper, we propose a TTL-based caching scheme to improve the performance of query processing. In addition to the TTL caching scheme, the VISE query-processing kernel is multi-threaded for further performance improvement. To evaluate the performance improvement of the caching scheme, we have conducted several experiments. The experimental results show that the query processing time is indeed improved after the caching scheme is incorporated. Although there may be stale data, the stale data rate is kept under 5%. Although the experiments are preliminary, we believe that the caching design improve the practicability of VISE to help users efficiently find the important information.

Keywords: visual interface, TTL-based caching, query processing, multi-threading, Web search engines.

1. Introduction

Because of the explosively growing amount of information on the World Wide Web, finding information of the most interest becomes a difficult problem. To solve the problem, Web search engines provide services in helping people find and retrieve information of interest from the huge amount of Web pages. Without the assistance of search engines, the results found by manually tracing the hyperlinks may be limited, and the finding process is time-consuming.

However, most popular Web search engines such as Google, MSN, and Yahoo provide their search results in a list-based interface where the title, summarized description, and URL of each entry are listed in lines. Two main drawbacks exist in the list-based representation. First, a list-based representation can provide limited information for each result entry due to the limitation of text representation. Users need to spend much time to navigate all possibly related items. However, since the provided information is limited, users may give up their searching after they browse top ten to twenty pages [1]. Second, it is hard to show relationships between the search results in a list-based interface. For example, displaying the linkage relationships are not considered in the list-based representation. Whether two entries are on the same Web site can be also hardly shown in the list-based representation.

On account of the shortcomings of the list-based interface, research efforts have been initiated to improve the visual representation of the search results. The research systems include WebQuery [2], HyperSpace [3], CardVis [4], and INSYDER [5]. In our previous study, we have also proposed a visualizing interface called VISE (Visual Interface for Search Engines) [6] specifically to visualize the search results of Web search engines. VISE is designed to improve the drawback of the list-based interface by visualizing more information to help users sift through the ranked pages with link relevance.

Figure 1 shows the system architecture of VISE. The query formulator processes the query terms in a legal form and sends them to the back-end search engine. After the search engine returns query results, the crawling analyzer analyzes the results, retrieves the content of each result entry, and yields the hyperlink structure of the results. Then, the visualization engine draws the graph of the search results according to the hyperlink structure. The overall relevance relationships are thus visualized in the graphical interface. The search entries are represented with different nodes, and linkage relationships are demonstrated with connected lines.

However, the VISE prototype has a severe performance problem because the crawling analyzer spends much time in retrieving remote Web pages on the fly. Although this guarantees that the visualization is up-to-date, the lengthy retrieval

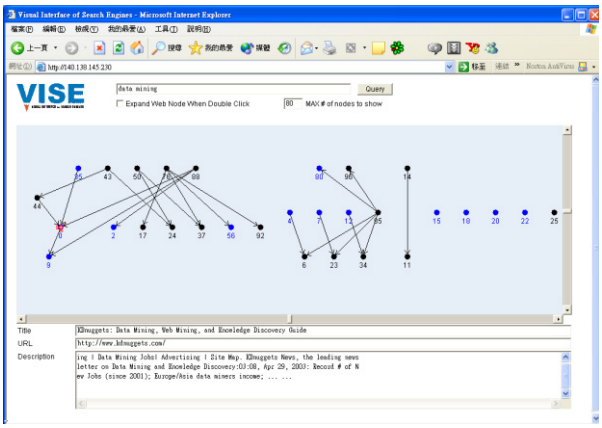

Figure 1: The visualized interface of VISE.

process makes VISE impractical to provide good service. For example, in some worst cases VISE may spend over one hour in retrieving all Web pages according to the results of a general query.

To improve the practicability of VISE in query processing, two approaches are incorporated in the VISE query-processing kernel. First, a TTL (Time-To-Live)-based caching scheme is designed to cache previously retrieved query pages. Second, the query-processing kernel is multi-threaded to parallelize the query processing. Caching mechanism has been proved to be a very effective mechanism in system performance improvement. Specifically, Web caching plays an important role in reducing server load, reducing network traffic, and decreasing service response time. On the other hand, the parallelized query processing highly reduces the response time. Although multi-threading incurs overheads in thread generation, the overheads can be neglected if compared with retrieval network latency.

In this paper, we present the TTL-based cache design of the VISE query-processing kernel and other improvements in retrieval process. In the caching scheme, consistency maintenance is an important issue to avoid accessing stale information. The consistency maintenance mechanism is therefore discussed in the cache design. A replacement policy is also described to show how to maintain an enough cache space. Besides, the improvements on the query-processing procedure of retrieving remote objects are also elaborated.

We have conducted several experiments to measure the performance improvement in VISE and the effectiveness of the cache design. Although the experimental results are still preliminary, they show the great improvement on the practicability of VISE.

The rest of the paper is organized as follows. Section 2 gives an overview of previous consistency algorithms and the studies about lifespan of Web pages. Section 3 elaborates the cache design of VISE and other improvements. Section 4 describes our experimental results and summarizes the discussion. Section 5 concludes the paper.

2. Consistency and Lifespan Considerations

Caching mechanism has been proved to be a very effective mechanism in system performance improvement to reduce server load, network traffic, and thereafter the service response time. Two important issues need to be discussed in a cache design. The first issue is to maintain the consistency between the cache and the remote source. The second is to perform cache replacement for keeping enough cache space.

According to the categorization of consistency algorithms classified by Cao and Ozsu [8], the ways to maintain cache consistency fall into three categories: the client validation approach, the server invalidation approach, and client-server (C/S) interaction approach. In the client validation approach, the cache manager at the client side is responsible for maintaining the consistency. With server invalidation, cached objects are always assumed to be up-to-date. Whenever an object is changed on the server, the server notifies all the client caches to maintain the consistency. In the C/S interaction approach, the client and the server work interactively to maintain the cache consistency.

Considering the strictness of the consistency, each approach can be further classified into two sub-categories: strong cache consistency and weak cache consistency. They give a more detailed classification table of the cache consistency algorithms here shown in Table 1 [8]. In the strong consistency model, the consistency between cached copies and original ones is always maintained. In the weak consistency model, the cached copies may be stale for a while, and then the consistency is maintained. The TTL (Time-To-Live)-based approach is a client-validation, weak consistency approach, which is suitable for VISE. This is because the back-end Web servers are not aware of the cache in VISE. The Web servers cannot actively maintain the consistency. In addition, maintaining strong consistency from VISE will incur many validation messages and thus suffer from heavy network traffic.

However, the lifespan of the cached objects should be considered carefully in the TTL-based

Table 1: The classification of cache consistency algorithms by Cao and Ozsu [8]. Our TTL-based cache design is in the client validation approach with weak consistency maintenance.

	Client Validation	Serer Invalidation	C/S Interaction
Strong	Polling-every-time	Invalidation	Lease
Weak	TTL and PCV	PSI	N/A

approach to avoid keeping stale copies too long. In previous studies [9, 10], the lifetime of different domain of Web pages is analyzed according to their network domains. The analysis from [10] shows that about 50% of the *.com* Web pages and about 90% of the *.edu* Web pages remains unchanged on the 10th day. The lifetime of about 50% of the *.edu* Web pages can be even lasted for almost 4 months. The study in [9] has similar results. We will accordingly decide the TTL parameters in our scheme.

For keeping enough cache space, many replacement polices have been discussed and investigated in Web caching to improve the caching performance [12,13,14,15,16]. According to categorization by Aggarwal et al. [16], replacement policies can be categorized as: direct extensions of traditional policies, key-based policies, and function-based replacement policies. From these studies, LRU shows its averagely prominent performance in replacement control.

3. Query-Processing Kernel Improvements

To improve the retrieval performance of VISE, the query-processing kernel is enhanced by incorporating a Web page cache and a multi-threaded retrieval crawler. The cache stores the previously retrieved and analyzed Web pages. It also keeps other information such as the hyperlink relationship. The design details are described in the following.

3.1 Cache Design

Figure 2 depicts the architecture of VISE augmented with the Web page cache. After the crawling analyzer receives the query results from the search engine, it analyzes the cached Web pages and synthesizes the hyperlink structures from the cache instead of immediately retrieving them from remote Web servers. If the analyzed Web pages are invalid or the pages are not in the cache, the cache manager

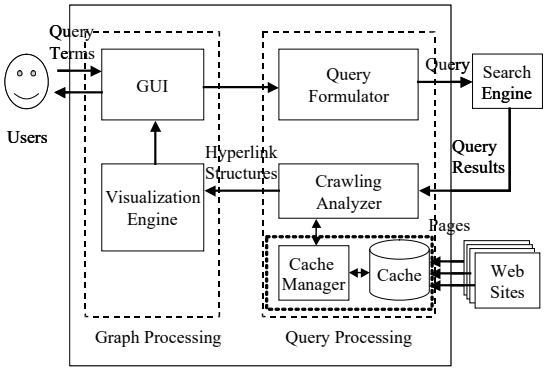

Figure 2: The operation process of VISE augmented with the Web page cache.

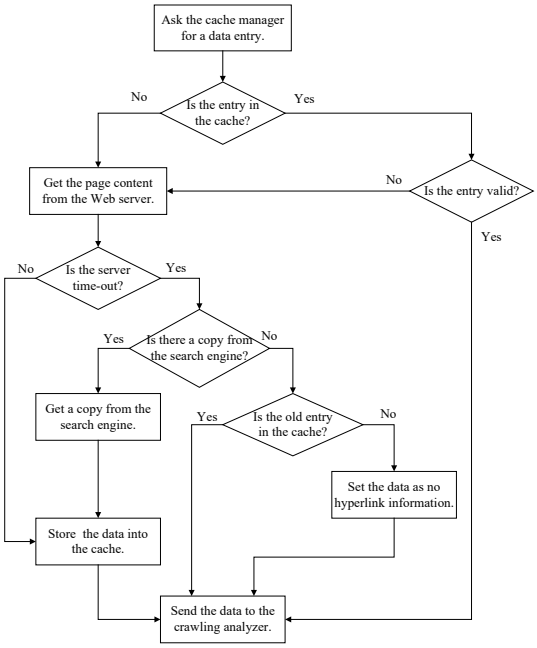

Figure 3: The flowchart of crawling procedure.

starts the crawler to fetch these pages. Due to the complicated network conditions, the retrieving time may be lengthy and block the future visualization process. Therefore, to decrease the probably lengthy processing, several approaches are adopted. First, the retrieval process is multithreaded so the retrieval is parallelized. Second, the waiting time is bounded. If the retrieval time exceeds the time-bound, VISE will try to use the pages cached at the back-end search engine. However, there are only some search engines providing such functionality. In this case, the locally cached pages will be still used even if they are stale. In the worst case, VISE may not find any Web page for the URL. Then this entry will be discarded.

The cache also maintains the correspondent hyperlink reference relationships. If the cached Web pages are valid, the visualiation engine will directly process these hyperlink struction. The visualization performance is therefore imporved.

3.2 The Crawling Procedure

In the original VISE, the crawling analyzer is designed to retrieve and analyze the Web pages on the fly. However, this real-time retrieval incurs a severe performance problem. With the incorporation of the Web page cache, the query-processing bottleneck is highly relieved. Figure 3 depicts the complete flowchart of the improved crawling procedure. The crawling analyzer will first interact with the cache manager to check the valid cached Web pages and reference relationships.

In the crawling procedure, only if the requested entries are in the cache and valid, will they be used in further visualization. Otherwise, the cache manager will try to request the pages from the origin servers. If the requests are not time-out, the requested pages will be stored in the cache and then sent to the crawling analyzer. If the requests are time-out as file is not found (error message 404) or server does not respond (error message 603), the cache manager will try to request the cached data from the back-end search engine and store the copies in the cache for later hyperlink analysis. If no Web page is available from the Web servers and search engines, the cache manager will try to use the stale but available entries from the cache. In the worst case, VISE cannot find any page. In such a case, the result URLs are discarded and will not be visualized in VISE.

3.2 Cache Consistency

Because Web content providers may arbitrarily create/modify/remove the pages, cached entries are probably thus become stale. To reduce the number of the stale entries, the cache manager takes the responsibility to maintain the consistency. As previously mentioned, a client-validation and weak consistency approach is used in our cache design.

To avoid a large number of network messages for consistency maintenance, a Time-To-Live (TTL) prediction approach is adopted in the consistency maintenance protocol. From the studies [9, 10], Web pages of different domains have different average lifetimes. In Table 2, about 80% of the Web pages would generally change in 3 to 27 days. Because the VISE cache stores data entries from different domains, which have different cache lifetimes, the TTL mechanism will effectively reduce the number of consistency maintenance messages if the TTL parameters are carefully adjusted.

In HTTP protocol, the "expires" header field ideally provides the expiration date of a Web page, however, most Web pages do not provide such expiry information [10]. To predict the expiration date of Web pages without expiry information, an adaptive TTL approach with the lower bound of lifetime in different domains is adopted. The adaptive TTL is calculated as (*Now – Last-Modified*) * *M*, where *M* is an experience value determined heuristically. To prevent frequent data consistency in a short period of time, a threshold of TTL is set. If the TTL value of a page is smaller than the average time of 80% of the Web pages in that domain, it will be assigned to the value of the average domain lifetime. For example, when an estimate of adaptive TTL for a *.com* domain is smaller than 3 days, the value of TTL will be assigned to 3. The estimate of expiration date of a Web page will be *Now + Estimated TTL*.

Table 2: The lifetime for different domains when 50% and 80% of pages do not change according to [10].

Domain	Lifetime for 50% page unchanged	Lifetime for 80% page unchanged
.com	11	3
.net, .org	50	8
.edu	100	18
.gov	120	27

3.3 Replacement Policy

In a cache system, when the space is too small to store any new entries, the cache manager needs to remove valueless entries until the cache space is large enough to store new entries. Many replacement polices have been discussed and investigated to improve the caching performance [12,13,14,15,16]. Among these approaches, the Least-Recently-Used (LRU) algorithm is simple but averagely very effective. In LRU, the cache manager removes the least recently accessed pages until there is sufficient space for the new document [11]. Therefore, in the VISE cache design, LRU is used for the page replacement.

The replacement also considers the staleness of the cache entries. The stale entries will be first selected to be replaced with LRU. If the cache space is not large enough, the cache manager will select valid entries to evict them according to the LRU policy.

4. Performance Evaluation

To evaluate the performance improvements, we have conducted several evaluation experiments. In the experiments, the performance of cache-improved VISE is compared with the performance of the original VISE. The experimental systems are developed with JDK1.4.0 01 and executed on Red Hat Linux 7.3. In the experiments, Google is used as the back-end search engine in the VISE prototype. The experimental cache size is 40GB.

The performance is measured by querying different strings. In the first experiment, "data mining" was queried to evaluate the effectiveness of the caching scheme. In the second experiment, "SARS" was queried to see the difference of querying a hot term. In Table 3 and Table 4, both the processing times of the first-time query and the average of the following 10 times are listed. In these experiments, the improvement of cache is very obvious. In the third experiment, the performance of cache consistency was evaluated for the same queries performed in the first experiment. The experimental results show the influence of the TTL scheme.

Table 3: The processing time and cache usage for querying “data mining”.

	Processing Time (ms)	Cache Access	Miss	Hit	
				Valid	Invalid
The first-time query	274279	200	200	0	0
Avg. of the following 10 times of queries	8063	200	17	183	0

Table 4: The processing time and cache usage for querying “SARS”.

	Processing Time (ms)	Cache Access	Miss	Hit	
				Valid	Invalid
The first-time query	342441	200	200	0	0
Avg. of the following 10 times of queries	10047	200	9	189	2

Finally, the hit rates of similar queries about the NBA player “Shaquille O’ Neal” were measured to show the performance when there were many cache hits.

4.1 Experimental Results for Running Cache

In the experiments, we first measured the performance of the original VISE by querying “data mining” and retrieving 200 result entries. The processing time was total 3950729 milliseconds. The lengthy processing time is mainly because the crawling process was sequential and the crawler needed to wait for retrieval completion.

After VISE used the improved multi-threaded query-processing kernel, the processing time was reduced to total 274279 milliseconds for the first-time query. In this case, the cache was empty. We continued to query “data mining” for 10 times to see the imporvement when some Web pages had been cached. The processing time was further reduced to total 8063 milliseconds, acceptable for most users.

Table 3 and Table 4 further show the number of cache misses and valid cache hits. In the case of querying “data mining”, the average processing time was 8063 milliseconds, and the total data size, which contained 6335 hyperlinks, was 3765 KBs. In the case of querying “SARS”, the average processing time was 10047 milliseconds, and the total data size, which contained 9979 hyperlinks, was 6419 KBs. After the following 10 times of querying “data mining” and “SARS”, there were respectively 183 and 189 data entries stored in the cache. In both cases, there were separately 17 and 9 data misses, and the average hit ratios were 91.5% and 94.5%.

The average misses of querying “data mining” are 17. The misses were caused by the Web pages that could not be retrieved and cached in the previous querying. Besides, the search engine happened to return different search results. This randomized response also resulted in cache misses.

In querying “SARS”, 3 of 9 cache misses of are because of removed pages, and 2 of 9 cache misses are for invalid pages whose lifetime provided by HTTP “expired” field was always too short to be accessed. Others were because of the randomized response. However, the visualization time was comparable to the response time of Google in both situations.

4.2 Evaluation of the Adaptive TTL Consistency Algorithm

Because most Web sites do not provide the information of expiration time, the lifetime is predicted for each cache entry to avoid unnecessary consistency checking. However, the TTL parameters must be carefully adjusted. Otherwise, the predictive lifetime of cache entries may mislead the crawling analyzer to use the stale data.

In this experiment, the utilization rate of the stale cache entry and the processing time were calculated. The M value of the adaptive TTL approach was assigned to 0.5, and the lower bounds of lifetime for *.com*, *.net*, *.edu*, *.gov*, and other domains were assigned to 3, 8, 18, 27, and 8 days, respectively. The stale rates and the processing times were recorded after 1, 3, and 7 days from the first querying.

In Table 5, we can observe that there were many cache hits and most of the cached entries were not modified. After the first-time query, only three entries were updated because the server updated the expiry information in the first day. In the whole week, the data were not updated frequently. However, the stale data rate was 0 because the pages were indeed not modified on the servers although the servers reported the expiration.

Because "SARS" was the latest news issue before the experiment and the pages were updated frequently, the last-modified times for many pages were close to the access times. In addition, its predicted lifetime was short. Therefore, the overhead of consistency check became much higher. As shown in Table 6, the rates of stale entries used are 4.1% and 5.1% in three days and seven days. According to our observation, the Web pages about SARS appeared mostly in the *.gov* and *.org* domains, and the stale data were from these two domains. However, for over half of Web servers VISE could not determine from the IMS messages whether the pages were modified, these Web pages were still retrieved even if they were not modified. This injured the system performance.

4.3 Evaluation of Similar Queries

In this experiment, there were four similar and related terms that queried sequentially. It shows the effect of processing time when the same cache entries are referred by different querying terms. The processing time and the cache usage are shown in Table 7.

When similar terms are queried, the search engine will possibly return the same search results, and the probability of hit entries in the cache will become higher. Shaquille O'Neal is a famous basketball NBA player, and Shaq is his nickname. In Table 7, the hit ratio rises when similar terms are queried sequentially, and the processing time decreases accordingly.

4.4 Discussion

Because of the parallelized retrieval procedure and the incorporation of the reference cache, the query processing time is highly improved. Compared with the response time of querying Google, the visualization time in VISE is very close with a 90% cache hit ratio. As shown in the experimental results, VISE is more practicable for visualizing the query results.

Because stale data ratio of querying the frequently updated topics such as "SARS" is about 5%, the information provided is mostly up-to-date. When most similar entries are cached in the VISE interface, it is believed that the performance improvement makes the VISE interface more convenient to network users.

5. Conclusions

Because of the explosively growing amount of information on the World Wide Web, finding information for the user needs becomes a difficult problem. Search engines can help users retrieve information of interest from large amount of Web pages, but the list-based interface shows only limited information for search results.

In our previous research, VISE is proposed to provide a visualization interface for displaying the search results and the hyperlink reference

Table 5: The stale data rate for term "data mining".

Days after the first-time query	Processing Time (ms)	Amount of Cache Hit	Valid Hit	Update (Server)	Update (Predicted)	Number of Stale Entry Used
1	9856	183	180	3	0	0
3	131458	184	160	16	8	0
7	116213	184	170	2	12	0

Table 6: The stale data rate for term "SARS".

Days after the first-time query	Processing Time (ms)	Amount of Cache Hit	Valid Hit	Update (Server)	Update (Predicted)	Number of Stale Entry Used
1	12133	184	177	7	0	0
3	187413	187	122	16	49	5
7	95143	184	174	3	17	9

Table 7: The processing time and cache usage of four similar query terms.

Query Terms	Processing Time (ms)	Cache Access	Miss	Hit	
				Valid	Invalid
Shaquille O'Neal	264836	200	200	0	0
Shaq	251711	200	186	14	0
O' Neal	181625	200	164	36	0
NAB O'Neal	92581	200	114	86	0

relationships between the result entries. However, the original VISE design does not consider the optimization of query processing and the poor query-processing performance decreases its practicability.

In this paper, an improved query-processing kernel is presented. It includes a TTL-based caching scheme and a parallelized procedure. A reference cache is designed to store the retrieved Web pages and the hyperlink reference information. The adaptive TTL approach is used for the cache consistency algorithm, and different thresholds are defined for the estimated TTLs of different network domains.The query-processing time is highly reduced and comparable to the response time of normal Web search engines.

To evaluate the performance improvement of the caching scheme, we have conducted several experiments. The experimental results show that the query processing time is indeed improved after the caching scheme is incorporated. As shown in the experimental results, the TTL scheme effectively keeps the sate data rate under 5%, and the cache hit rate is kept around 90%.

Although the experiments are preliminary, we believe that the caching design improves the practicability of VISE to help users efficiently find the important information. In the future, more comprehensive experiments will be conducted to study other performance bottlenecks. In addition, prefetching is considered for further performance improvement. More improvements on VISE human-computer interaction are also in our future plan.

References

[1] B. Amento, W. Hill, L. Terveen, D. Hix, and P. Ju. “An Empirical Evaluation of User Interfaces for Topic Management of Web Sites”. In *Proceedings of the CHI99 Conference on Human Factors in Computing Systems*, pp.552–559, 1999.

[2] J. Carriere and R. Kazman. “Web Query: Searching and Visualizing the Web through Connectivity”. In *Proceedings of the 6th International World Wide Web Conference*, pp. 701–711, April 1997.

[3] R. Beale, R. J. McNab, and I. H. Witten. “Visualizing Sequences of Queries: A New Tool for Information Retrieval”. In *Proceedings of 1997 IEEE Conference on Information Visualization*, pp. 57–62, 1997.

[4] S. Mukherjea and Y. Hara. “Visualizing World-Wide Web Search Engine Results”. In *Proceedings of the 1999 IEEE International Conference on Information Visualization*, pp. 400–405, July 1999.

[5] T. M. Mann. “Visualization of WWW-Search Results”. In *Proceedings of the International Workshop on Web-Based Information Visualization (WebVis’99)*, pp. 264–268, 1999.

[6] H.-C. Yang, M.-C. Tzeng, and C.-Z. Yang. “A Web Interface for Visualizing Web Search Engine Results”. In *Proceedings of the 2001 National Computer Symposium, Workshop on Internet and e-Commerce*, Vol. I, pp. 151–159, December 2001.

[7] J. Gwertzman and M. Seltzer, “World Wide Web Cache Consistency”. In *Proceedings of the 1996 USENIX Technical Conference*, San Diego, CA, pp. 141–152, 1996.

[8] L. Y. Cao and M. T. Özsu. “Evaluation of Strong Consistency Web Caching Techniques”. *World Wide Web*, Vol. 5, No. 2 pp. 95–123, 2002. Kluwer Academic Publishers.

[9] X. Chen and P. Mohapatra. “Lifetime Behavior and its Impact on Web Caching”. In *IEEE Workshop on Internet Applications, 1999*, pp. 54–61, 1999.

[10] J. Cho and H. Garcia-Molina. “The Evolution of the Web and Implications for an Incremental Crawler”. In *Proceedings of the 26th International Conference on Very Large Data Bases*, pp. 200–209, 2000.

[11] S. Glassman. “A Caching Relay for the World Wide Web”. In *Proceedings of the first International Conference on the World Wide Web*, pp. 165–173, May 1994.

[12] M. Abrams, C. R. Standridge, G. Abdulla, S. Williams, and E. A. Fox. “Caching Proxies: Limitations and Potentials”. In *Proceedings of the 4th International WWW Conference*, July 1995.

[13] S. Williams, M. Abrams, C. R. Standridge, G. Abdulla, and E. A. Fox. “Removal Policies in Network Caches for World-Wide Web Documents”. In *Proceedings of ACM SIGCOMM 96*, pp.293–305, Augast 1996.

[14] J. Dilley, and M. Arlitt, “Improving Proxy Cache Performance-Analyzing Three Cache Replacement Policies”. *IEEE Internet Computing*, Vol. 3, No. 6, pp. 44–50, November/December 1999.

[15] L. Rizzo and L. Vicisano, “Replacement Policies for a Proxy Cache”. *IEEE/ACM Transaction on Networking*, Vol. 8, No. 2, pp. 158–170, April 2000.

[16] C. Aggarwal, J. L. Wolf, and P. S. Yu, “Caching on the World Wide Web”. *IEEE Transaction on*

Knowledge and Data Engineering, Vol. 11, No.1,
pp.94–107, January/Febrary 1999.