Eliminating the I/O Bottleneck in Large Web Caches

Alex Rousskov and Valery Soloviev
Computer Science Department
North Dakota State University
Fargo, ND 58105-5164
{rousskov,soloviev}@plains.NoDak.edu

Abstract

This paper presents a technique for eliminating the disk bottleneck in large Web Caches. Our approach objective is twofold. First, the presented algorithm substantially decreases disk activity during peak server load. Second, it maintains the hit ratio at the level of traditional caching policies. We evaluate the performance of the algorithm using trace driven simulations based on access logs from several top-level Web caches.

1 The Problem

Caching servers or *caching proxies* are now standard tools for handling the exponential growth of the Web traffic. Individual caching proxies can boost their performance by joining cache *hierarchies*. Several large hierarchies or *cache meshes* are currently operational [1, 2].

In a cache mesh, intermediate servers have to serve data to all proxies they cooperate with. To be effective, an intermediate cache server must handle gigabytes of traffic per day and maintain a large storage of cached documents. The traffic volume is rapidly increasing with the Web growth and as new servers join the hierarchy. Due to the system overhead, it may take about four disk I/Os to cache or retrieve a single document. It is no surprise that the disk subsystem becomes a serious bottleneck in a large caching proxy [3, 4].

In the nearest future, multimedia data such as audio and video clips are expected to become a major part of the Internet traffic. Caching proxies could be naturally used for caching and *smoothing* the delivery of delay-sensitive media. This new media, along with the Web traffic growth, will increase the burden on caching proxies, especially on their disk subsystem.

Current caching schemes rely on algorithms originally designed for *in-memory* caches in database and file systems [5, 6]. Thus, traditional algorithms ignore performance problems associated with maintaining large *disk-resident* archives of cached documents.

This paper demonstrates that traditional caching algorithms create excessive load on a caching server and do not scale with the increase in the volume of the Web traffic. We present a new caching algorithm designed to minimize I/O activity on a server while maintaining the hit ratio at the level of traditional algorithms.

2 Traditional Approach

Many Web caching algorithms have been proposed. Most algorithms adopt techniques found in database and file systems [5, 6]. A few recent studies take the specifics of the Web traffic into account [7]. All existing algorithms optimize the hit ratio and ignore I/O activity.

A common pattern among traditional algorithms is to store *every* incoming cachable document while purging the previously cached ones to free disk space. Thus, there is a persistent flow of data to the disk-resident cache. Clearly, the volume of disk traffic is proportional to the Web traffic. That is, when the number of requests to a caching proxy increases, so does the number of documents written to the disk cache. Disk queues grow exponentially and so does disk response time [4]. This results in an I/O bottleneck system.

The I/O bottleneck leads to the performance degradation of the entire caching proxy. When the disk subsystem cannot handle incoming requests, the response time of a single request increases. A longer response time means more concurrent requests in the system. The latter leads to shortages in buffer memory, CPU slices, file descriptors, etc. As the queuing theory suggests, these resource shortages increase *exponentially* when the server load is high. Thus, the performance problems are most severe during the peak server load.

To compensate for the increase in the load during peak hours, the system must have excessive disk, memory, and CPU resources. These resources are not utilized for the rest of the time.

3 Proposed Algorithm

This section presents an algorithm designed to eliminate the I/O bottleneck in large caching proxies. Our design objective is twofold. First, we want to substantially decrease disk activity during peak server load. Second, we want to maintain the hit ratio at the level of other caching algorithms.

The traditional approach is to write *every* new cachable document to disk. In a large hierarchy, the majority of Web documents are requested from an intermediate caching proxy only once. Thus, these documents are written to the disk but are never read back. If we could predict future requests, we would never write such documents and would drastically reduce disk activity.

A precise prediction of the Web traffic is, of course, not feasible. However, our analysis shows that most documents requested today were requested in the past as well. Thus, by maintaining an *active set* of previously requested documents, we could "predict" the future traffic. We can rebuild the active set only once per day when the server load is negligible. We propose `Static Caching` algorithm that works as follows.

- Once per day, when the server load is negligible, scan the *log file* of a caching proxy to determine a set of URLs (names) of previously accessed documents.
- Form an active set by selecting URLs of most *valuable* documents among those scanned in the first step. A value of a document is determined by its contribution towards the total number of hits weighted by the document size. Such a value would guarantee an optimal hit ratio if the future traffic matches the past precisely.
- If a document from the active set is not currently cached, then it can be either prefetched when the active set is formed or fetched during the first corresponding request. We assume the first scenario.
- During the day, the active set of URLs remains unchanged or *static*. A request to a cached document from the set produces a *hit*. A request to a document outside of the active set gives a *miss*.

`Static Caching` shifts all major disk and CPU activities from peak load hours to the time when a caching proxy is idle. During the day, the disk activity is triggered by *hits* and *updates* of active documents only. The former are essential to maintain a high hit ratio. The latter are rare and have negligible impact. Thus, there is no excessive disk activity during peak load.

As a side effect, the CPU load is minimal during peak hours. A single hash table lookup is required to process a request. No work on maintaining the hash table or any other meta data is required. A single hash table is all that has to be stored in memory buffer. Thus, memory requirements are also minimal. Other benefits of the approach include a potential for exchange of active sets among cooperative caching proxies to optimize the cooperation and for compression of cached documents.

4 Performance Analysis

Using trace driven simulations, we have compared the performance of the `Static` algorithm with the `LRU-TH` algorithm [8]. `LRU-TH` and its variations are traditionally used in caching proxies [9, 3]. Due to cache sizes exceeding daily traffic volume, many other known algorithms will mimic the behavior of `LRU-TH`. In our previous work with *primary* Web servers, we observed a similar performance of many algorithms on large caches [10].

We used traces from six *root* servers of a large cache hierarchy maintained by a National Laboratory for Applied Network Research [1]. Traces were produced by the Squid caching proxy, a freeware successor of Harvest [9]. Squid is currently the best freeware proxy accounting for about 70% of all European caches [11]. Traces' duration was about two months. Each server studied has distinct traffic patterns. For example, the `SV` server handled mostly international traffic and processed more than 6 GB per day (more than 600,000 requests). On the other hand, the `LJ` server had an open access list and processed almost 2 GB per day (about 150,000 requests). NLANR servers had 6-8GB allocated for disk cache.

For each URL request, Squid logs the time of the request, the URL, the size of the document, and the performed action [1]. The last modification date of a document is not logged, but updates can be usually detected by analyzing the action field. Furthermore, for every request, we compared the size of the requested file with the one cached by an algorithm. A change in the file size indicates a modification and results in a "miss".

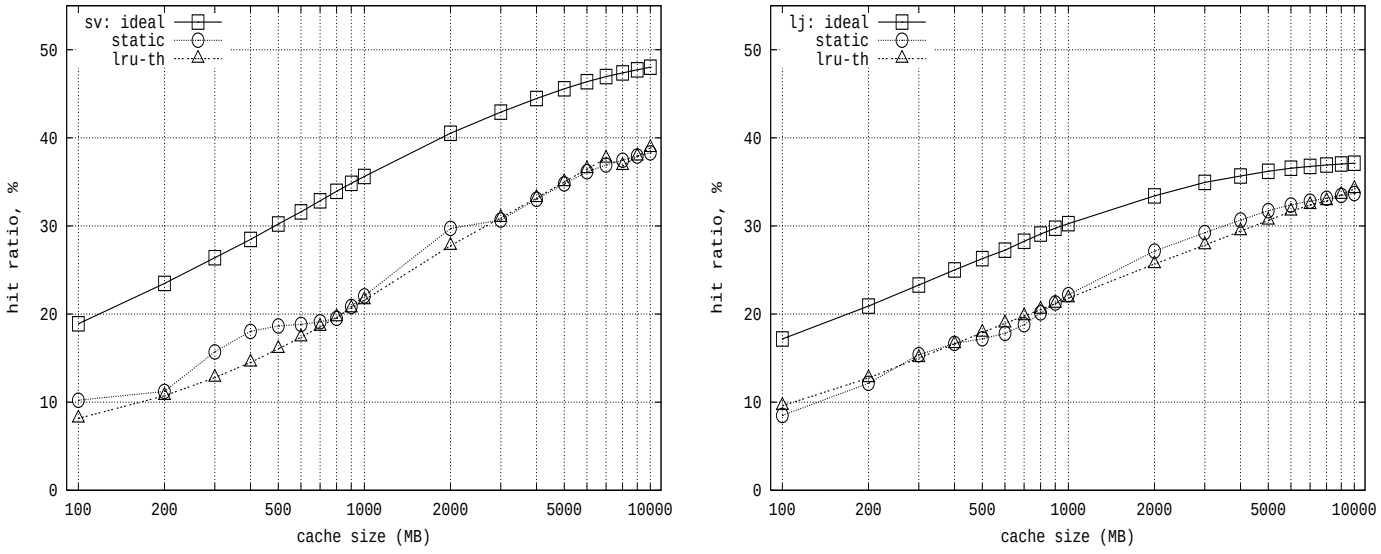

Figure 1: `Static` vs. `LRU-TH` on `SV` and `LJ` servers

Figure 1 compares the performance of the `Static` algorithm with the `LRU-TH` algorithm on the `SV` and `LJ` servers. Performance on other servers studied is similar. `LRU-TH` is shown with the *best* threshold for each cache size. Solid lines show an upper bound on `Static` performance in a hypothetical case when the future is 100% predictable (i.e., we know what *old* URLs will be accessed again; there are still *new* URLs that can be cached by `LRU` but cannot be cached by `Static`).

Clearly, elimination of excessive I/O load by `Static` did not sacrifice the hit ratio. We also measured the number of disk I/Os produced by the algorithms. `Static` consistently reduced daily disk traffic by about 35%. The savings would become even bigger when only peak load is considered. Such a reduction would eliminate disk bottleneck in a real proxy environment.

5 Conclusions and Future Work

The expansion of the Web and increasing presence of multimedia information result in the I/O bottleneck on caching proxies. Traditional caching algorithms ignore the performance of the I/O subsystem and do not scale with the increase in traffic volume and intensity. We have presented the `Static Caching` algorithm which focuses on saving disk bandwidth. `Static Caching` eliminates the I/O bottleneck while preserving a high hit ratio.

Currently, we are investigating the application of `Static` algorithm to *leaf* proxy caches installed in large universities and corporations. Leaf caches handle more *dynamic* traffic than intermediate caching proxies. An interesting *symbiosis* of the `Static` algorithm and a traditional dynamic caching policy may be needed to maintain a high hit ratio with minimum resource requirements.

Acknowledgements

This work was supported, in part, by NSF grants `OSR-95-53368` and `RIA IRI-94-09845`.

References

[1] National Laboratory for Applied Network Research.
the lab: `http://www.nlanr.net/`, cache project: `http://ircache.nlanr.net/`

[2] Europe Caching Hierarchy
`http://www.terena.nl/projects/choc/`

[3] Carlos Maltzahn, Kathy Richardson, and Dirk Grunwald. Performance Issues of Enterprise Level Web Proxies. *In Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, Seattle, June 1997
`http://www.cs.Colorado.edu/ carlosm/sigmetrics.ps.gz`

[4] Alex Rousskov and Valery Soloviev. On Performance of Caching Proxies. Submitted for publication.
`http://www.cs.ndsu.nodak.edu/~rousskov/research/cache/squid/profiling/papers/`

[5] E.O'Neil, P. O'Neil, and G. Weikum. The LRU-K Page Replacement Algorithm For Database Disk Buffering. *In Proceedings of the ACM SIGMOD Int. Conf. on Management of Data*, Washington, May 1993.
`http://paris.cs.uni-sb.de/public_html/papers/LRU-k_report.ps.Z`

[6] R. Karedla, J. Love, and B. Werry. Caching Strategies to Improve Disk Performance. *IEEE Computer*, pp. 38-46, v.27(3), March 1994.

[7] J. Pitkow and M. Recker. A Simple Yet Robust Caching Algorithm Based on Dynamic Access Patterns. *In Proceedings of the Second International WWW Conference*, Chicago, October 1994.
`http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/DDay/pitkow/caching.html`

[8] M. Abrams, C. Stanbridge, G. Abdulla, S. Williams, and E. Fox. Caching Proxies: Limitations and Potentials. *In Proceedings of the Fourth International Conference on the WWW*, Boston, December 1995.
`http://ei.cs.vt.edu/~succeed/WWW4/WWW4.html`

[9] A. Chankhunthod, P. Danzig, C. Neerdaels, M. Schwartz, and K. Worrel. A Hierarchical Internet Object Cache. *USENIX Annual Technical Conference*, San Diego, January 1996.
`http://excalibur.usc.edu/cache-html/cache.html`

[10] Igor Tatarinov, Alex Rousskov, and Valery Soloviev. Static Caching in Web Servers. In proceedings of *the IEEE International Conference on Computer Communications and Networks, Las Vegas, September 1997*.
`http://www.cs.ndsu.nodak.edu/~tatarino/pubs/static.ps`

[11] European Caching Task Force Survey Results:
`http://w3cache.icm.edu.pl/survey/results/`