# Simulation Evaluation of a Heterogeneous Web Proxy Caching Hierarchy

Mudashiru Busari Carey Williamson
*Department of Computer Science*
*University of Saskatchewan*
*Email:* carey@cs.usask.ca

## Abstract

*This paper uses trace-driven simulations to evaluate the performance of different cache management techniques for multi-level Web proxy caching hierarchies. In particular, the experiments consider heterogeneous cache replacement policies within a two-level caching hierarchy, and size-based partitioning across the levels of a caching hierarchy. Three different synthetic Web proxy workloads are used in the study, reflecting complete overlap, partial overlap, and no overlap in the workloads seen by the child-level proxies. The simulation results demonstrate that heterogeneous replacement policies and size-based partitioning each offer modest improvements in caching performance. The sensitivity of the results to the degree of workload overlap is also discussed.*

## 1. Introduction

Caching proxies have gained widespread popularity on the Internet [1, 4, 5, 6, 7, 13, 31]. Proxies function as intermediaries between Web clients (browsers) and Web servers, accepting client requests and forwarding them to Web servers only as neccessary. When a requested document is returned by a Web server, the proxy server sends the document to the client and stores a copy of the document in its local cache. Depending on client request patterns, the proxy may be able to satisfy future client requests directly from the cache without contacting the Web servers.

In recent years, multi-level proxy cache configurations have received increasing research attention [12, 14, 16, 18, 22]. In a hierarchical configuration, proxies at or near the end-user constitute the lowest level of the hierarchy, often with sibling-sibling relationships with one another. The lowest level proxies may have a child-parent relationship to a higher level proxy, usually a (geographically) regional proxy. A regional proxy can in turn connect to a higher level proxy, such as a national proxy [18]. A request that cannot be satisfied by a proxy node is typically sent to a nearby sibling or to the parent using an Inter-Cache Protocol [14, 21, 23, 28].

Interesting design issues arise with caching hierarchies, and performance tradeoffs exist. For example, the potential advantages of reduced server load, reduced network traffic, and reduced end-user latency may be offset by inter-cache communication overhead, delays incurred at each level of the hierarchy, and performance bottlenecks at higher level proxies [23, 24].

Empirical measurements suggest that Web proxy caching hierarchies are not that effective [18, 19, 24]. For example, the measurements reported in [19] for a three-level caching hierarchy show document hit ratios of 35-40% for a university-level Web proxy cache, hit ratios of 15-20% for a national-level Web proxy, and hit ratios of 5-10% for a root-level NLANR (National Laboratory for Applied Networking Research) cache. Thus a caching hierarchy can suffer "diminishing returns": the further up the hierarchy you go, the less likely you are to find the document of interest. In many cases, a request to the originating server is eventually needed to resolve the sequence of cache misses incurred.

The diminishing returns phenomenon makes sense intuitively, since the lower-level caches filter out many of the hits. As a result, the workload characteristics seen at higher-level caches become more "random" in nature. The only surprising aspect is that the diminishing returns occur *despite* the fact that higher-level caches are often larger (sometimes significantly larger) than the caches at the lower levels. These observations suggest that caching hierarchies are not that well-designed. Often, too much focus is placed on the performance of a proxy cache in isolation, rather than as part of an overall caching system [27].

Several researchers have suggested ways to improve the performance of caching hierarchies. Tewari *et al.* [25] suggested a distributed approach using metadata to track where copies of files are stored in the hierarchy. A similar technique is suggested by Povey *et al.* [22], where only the lower level caches are responsible for storing documents, while upper level caches maintain information about the contents of lower level caches. The collaborative method proposed by Yu *et al.* [30] employs a protocol that passes caching information down the proxy hierarchy for the lower level proxies to make better caching decisions. The Cache Array Routing Protocol (CARP) [26] is a form of distributed caching where multiple proxy servers are configured to appear as a single logical cache to the clients. In the "summary cache" scheme proposed by Fan *et al.* [14], each proxy stores a summary of URLs of documents cached at every other proxy so that misses can be sent to a proxy with a copy of the requested document, or otherwise directly to the Web server.

Our approach is different. As a starting point, we assume the existence of a traditional Web proxy caching hierarchy, such as that in [18], and then strive to improve its performance. The work is motivated by the observation that the workload characteristics differ across the levels of a caching hierarchy [18], due to filtering effects at lower-level caches [10, 27]. This observation suggests the use of different (i.e., heterogeneous) caching policies at different levels of a caching hierarchy.

In particular, this paper addresses two research questions:

- In a multi-level caching hierarchy, can overall caching performance be improved by using different cache replacement policies at different levels of the hierarchy (e.g., Least-Recently-Used at the lowest level and Least-Frequently-Used at the highest level)?

- In a multi-level caching hierarchy, can overall caching performance be improved by maintaining disjoint document sets at each level of the hierarchy (e.g., small documents at the lowest level, and large documents at higher levels)?

This work uses trace-driven simulation, and a synthetic Web proxy workload generator called ProWGen (developed in previous work [9]) that captures the salient characteristics of Web proxy workloads (e.g., one-time referencing, Zipf-like document popularity, heavy-tailed file size distribution, temporal locality). For simplicity, only a two-level caching hierarchy is considered, with three types of cache replacement policies: recency-based, frequency-based, and size-based. Synthetic workloads are used to investigate the performance of different combinations of replacement policies at different levels of the hierarchy, and then to investigate size-based document partitioning approaches.

The simulation results show that combining different replacement policies at different levels of the hierarchy improves performance. The results also indicate benefits for size-based partitioning, if small files are kept close to the clients.

The remainder of this paper is organized as follows. Section 2 presents the simulation model and methodology used in the experiments. Section 3 presents the simulation results. Finally, Section 4 concludes the paper.

## 2. Simulation model and methodology

### 2.1. Simulation model

The simulation experiments model a two-level hierarchical Web proxy configuration as shown in Figure 1. In the simulation model, requests from the aggregate workload are forwarded to the lower level proxies, and misses are forwarded to the upper level proxy. Misses from the upper level proxy are forwarded to the (simulated) Web servers. There are no interactions directly between the two lower-level proxies, and there are no cache consistency mechanisms in the modeled hierarchy, since the workload is assumed to consist only of static documents, with sizes and contents that do not change over time.

### 2.2. Web proxy workload model

A Web proxy workload generation tool called ProWGen [9] is used to synthesize an aggregate client workload, with workload parameters shown in Table 1. These values are based on empirically observed workload characteristics at the lowest level of a Web caching hierarchy [8, 18].

The statistical characteristics of the resulting workload produced are shown in Table 2. The generated workload closely matches the desired characteristics from Table 1. Figure 2 provides a graphical illustration of selected workload characteristics, including a Zipf-like document popularity profile (Figure 2(a)), and a heavy-tailed file size distribution (Figure 2(b) and (c)). A more detailed discussion of the validation of ProWGen and the workloads that it generates appears in [8, 9].

The aggregate workload is then split across the two lower-level proxies to model three different scenarios: *complete overlap*, *partial overlap*, and *no overlap*. Each scenario reflects a different degree of overlap (i.e., common URLs in the Web document request streams) in the workloads of the two lower level proxies.

The *complete overlap* scenario models the situation where the two lower-level proxies reside in "similar" organizations (i.e., the aggregate client sets behave similarly, in terms of the Web content requested). Thus the cache contents at the two lower level proxies are likely to be statistically similar, on average. This scenario is modeled by randomly dispatching each request in the workload to one of the lower-level proxies (equiprobably, at random).

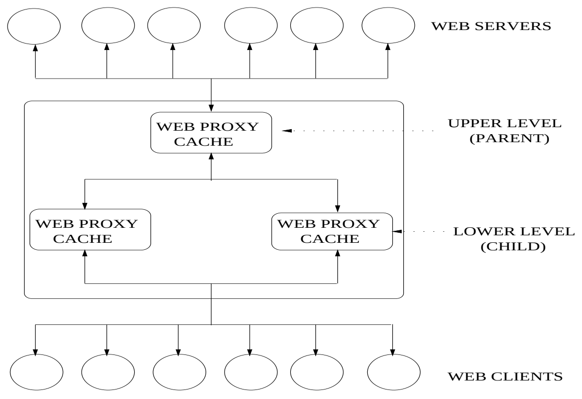


**Figure 1. Simulation model for two-level Web proxy caching hierarchy**

**Table 1. Parameter settings for ProWGen**

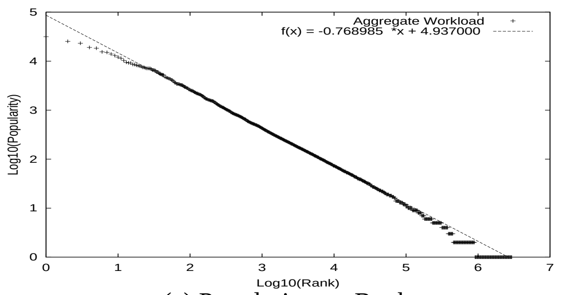
| Parameter | Value |
|---|---|
| Requests | 10,000,000 |
| Unique documents (% of total requests) | 30% |
| One-timers (% of unique documents) | 70% |
| Correlation (file size and popularity) | Zero |
| Zipf Slope | 0.75 |
| Mean of the lognormal distribution ($\mu$) | 7,000 |
| StdDev of lognormal distribution ($\sigma$) | 11,000 |
| Pareto tail index | 1.3 |
| Beginning of the tail ($k$) in bytes | 10,000 |
| Percentage of documents in the tail | 20% |
| LRU Stack Model for temporal locality | Dynamic |
| LRU Stack Size for temporal locality | 1,000 |

The *no overlap* scenario models Web proxies with entirely different document request streams (i.e., completely different client behaviours). This scenario is modeled by assigning requests for odd-numbered documents to one lower-level proxy, and requests for even-numbered documents to the other lower-level proxy.

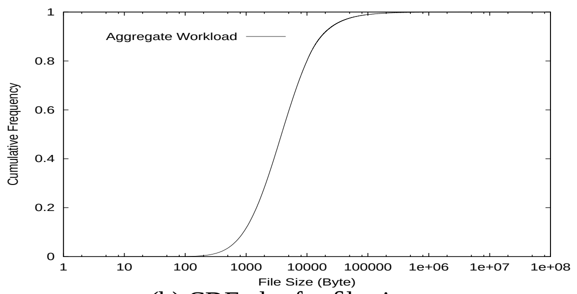
The *partial overlap* scenario represents an intermediate situation between the two extremes already discussed. In particular, we examined a scenario where there is 50% overlap in the workload of the two lower level proxies. This scenario is modeled by randomly choosing half of the documents to be shared (as in complete overlap), with the remaining documents split between the two child proxies on an odd-even basis.

**Table 2. Characteristics of synthetic workload**

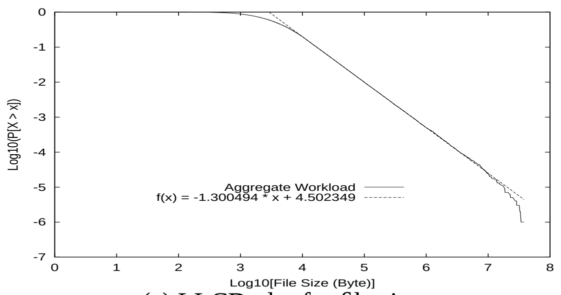
| Item | Value |
|---|---|
| Requests | 9,749,703 |
| Unique documents | 3,000,000 |
| One-timers | 2,099,045 |
| Total bytes of unique documents (GB) | 30 |
| Total transferred content bytes (GB) | 89 |
| Smallest file size (bytes) | 9 |
| Largest file size (bytes) | 51,600,679 |
| Mean file size (bytes) | 10,850 |
| Median file size (bytes) | 3,815 |
| Correlation (file size and popularity) | -0.004982 |
| Zipf Slope | -0.768985 |
| Pareto tail index | -1.300494 |



(a) Popularity vs. Rank



(b) CDF plot for file sizes



(c) LLCD plot for file sizes

**Figure 2. Popularity, CDF and LLCD plots**

## 2.3. Experimental design

The experimental methodology considers two main factors: *cache size* and *cache replacement policy*. Cache sizes range from 1 MB to 32 GB in the experiments, with all three caches (parent and two children) identical in size.

The replacement policy for the cache determines which document(s) to remove from the cache when more space is needed to store a new incoming document. Three cache replacement policies are considered: Least Recently Used (LRU), Least Frequently Used with Aging (LFU-Aging), and Greedy-Dual Size (GD-Size). In simple terms, LRU tries to keep recently active documents in the cache, LFU tries to keep popular documents in the cache (with *aging* to reduce the relative popularity of old popular documents), and GD-Size tries to keep small documents in the cache. These policies represent a broad range of candidate replacement policies (i.e., recency-based, frequency-based, and size-based), and are well-documented in the literature [2, 3, 11].

## 2.4. Performance metrics

Two performance metrics are used to evaluate cache performance: *document hit ratio* and *byte hit ratio*. The document hit ratio is the number of requests satisfied by a particular proxy's cache divided by the total number of requests seen by the proxy. The byte hit ratio is the volume of data (in bytes) satisfied by the proxy's cache divided by the total volume of data requested from the proxy.

Both metrics are required since Web documents can differ dramatically in size. The document hit ratio indicates the fraction of requests that are "offloaded" from the origin Web server by the presence of the proxy, while the byte hit ratio indicates the reduction in network traffic volume to and from the origin server.

A cache replacement policy might be designed to optimize one metric at the expense of the other. For example, caching only small documents might produce a high document hit ratio (since many documents fit in the cache), but a low byte hit ratio (since large documents must still be obtained from the origin Web server).

In general, the higher the document hit ratio and byte hit ratio are, the better a replacement policy is. Furthermore, the closer the "hit" is to the client, the lower is the (expected) document retrieval latency.

## 3. Simulation results

### 3.1. Heterogeneous replacement policies

The experiments in this section consider different replacement policies (LRU, LFU-Aging, and GD-Size) at the child and parent caches in the caching hierarchy.

Simulation results for the *complete overlap* scenario are shown in Figure 3. In this figure, the leftmost column of graphs shows document hit ratio results, while the rightmost column of graphs shows the corresponding results for byte hit ratios. Figures 3(a) and (b) show the results for the LRU policy at the child level, while Figures 3(c) and (d) show the results for LFU-Aging at the child caches, and Figures 3(e) and (f) show the results for GD-Size at the child caches. On each graph, the uppermost line shows the results for the child[1] cache, and the remaining three lines show the results for the parent cache, for each replacement policy considered.

In general, the child proxy caches have much higher hit ratios than the parent proxy. This observation is not surprising, given that the parent proxy only sees the requests that miss at the lower level caches (i.e, the request stream is filtered by the lower level proxies) [27, 29].

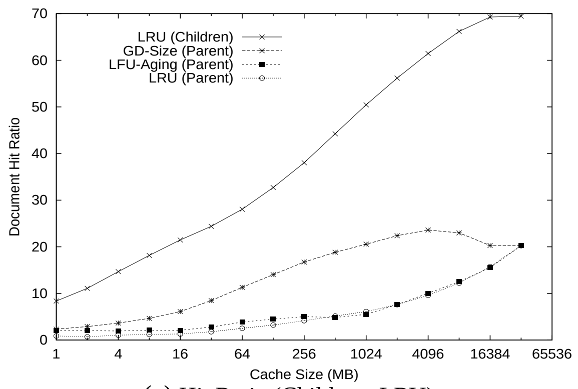
The graphs in Figure 3 illustrate some interesting differences in *marginal utility* (i.e., additional incremental value gained) when more cache space is added at either the child level or the parent level. This behaviour can be seen by noting the differences in the slopes of the hit ratio plots for the parent level and child level cache, as the cache sizes are increased (exponentially) from left to right. At some points, increasing the size of the child-level cache produces a sharp increase in hit ratio; at other points the graph is fairly flat. Similar observations apply for the parent level cache.

In some cases, the hit ratio results for the parent level cache *drop* as the cache size is increased. This non-monotonic behaviour happens because the child cache is also being increased in size, absorbing more hits, and reducing the number of requests to the parent level cache. In other words, the relative balance between "cold misses" (first request for a document) and "capacity misses" (subsequent request for a document that used to be in the cache, but has now been removed from the cache) changes as the cache sizes are scaled. This impact may be different at each level of the hierarchy.

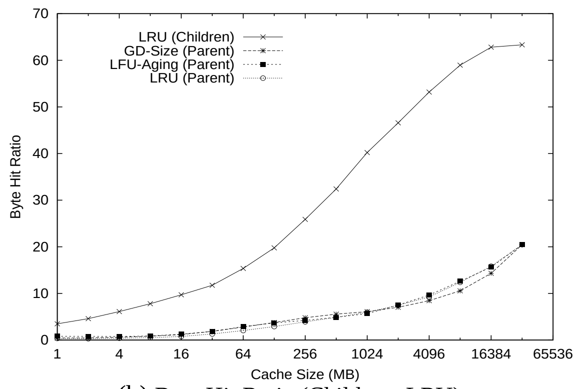
The exact shape of these curves depends, of course, on the nature of the workload: the temporal locality property, the Zipf-like referencing behaviour, and the size (in bytes) of the "document working set", relative to the cache size used. These marginal utility trends also depend on the cache replacement policy used, since the replacement policy at one level changes the workload characteristics for the next level of cache.

Overall, Figure 3 shows that the GD-Size policy at the parent cache provides a significantly better document hit ratio than either LRU or LFU-Aging at the parent cache. This document hit ratio advantage is a factor of two or more

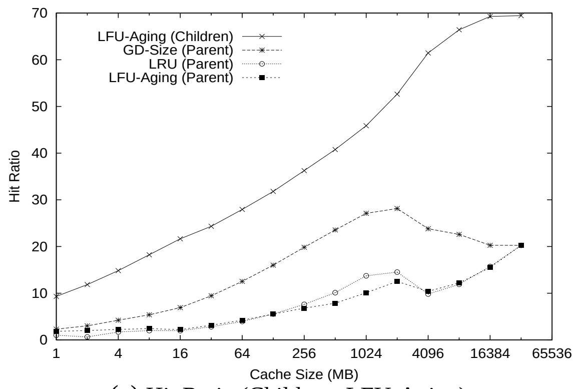
[1]Since the hit ratios are similar for each child cache, only the results for one child cache are shown, for clarity.



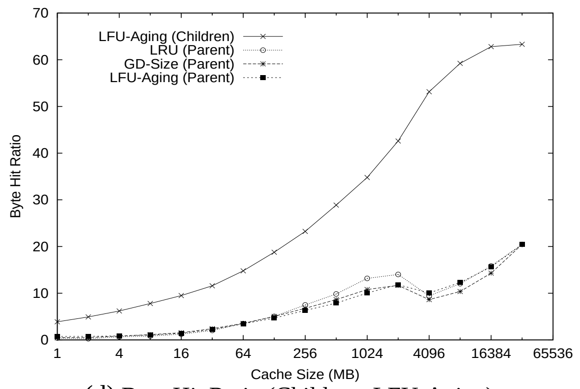
(a) Hit Ratio (Children: LRU)



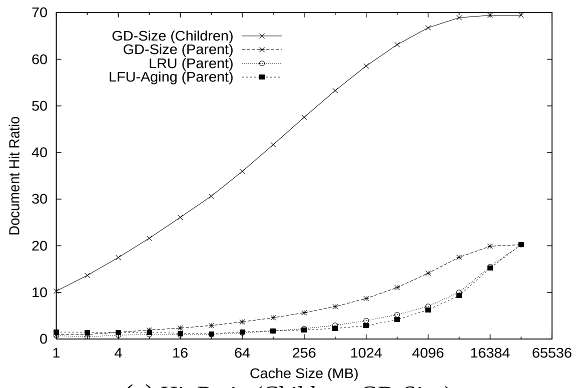
(b) Byte Hit Ratio (Children: LRU)



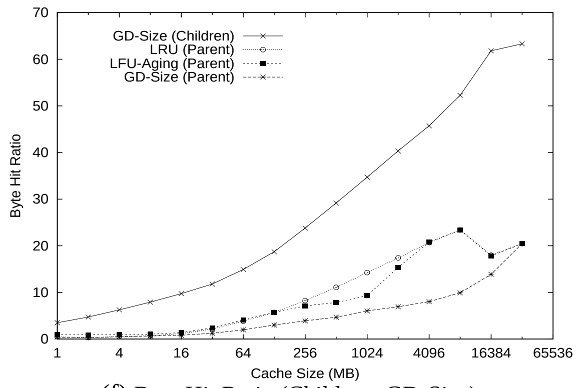
(c) Hit Ratio (Children: LFU-Aging)



(d) Byte Hit Ratio (Children: LFU-Aging)



(e) Hit Ratio (Children: GD-Size)



(f) Byte Hit Ratio (Children: GD-Size)

**Figure 3. Simulation results for different combinations of replacement policies in a two-level caching hierarchy (Complete overlap scenario)**

for most of the cache sizes considered, when the child level cache uses LRU (Figure 3(a)) or LFU-Aging (Figure 3(c)). Furthermore, this advantage does *not* come at the expense of the byte hit ratio (see Figure 3(b) and (d)), as is often the case with GD-Size [3]. The performance advantage of GD-Size at the upper level is less pronounced, but still present, when the child proxies use GD-Size (see Figure 3(e)). However, the document hit ratio advantage is compromised by a lower byte hit ratio (Figure 3(f)).

In summary, the potential advantage of heterogeneous caching policies is most evident in Figure 3(a). For a given cache size, the effectiveness (hit ratio) of a second-level cache can be doubled or tripled by using a size-based replacement policy that is different from that used at the first-level cache (LRU, for example).

### 3.2. Sensitivity of results to workload overlap

This section explores the sensitivity of the previous results to the degree of workload overlap between the two child-level proxies in Figure 1. The previous section assumed complete overlap in the workload: clients at either proxy are equally likely to access any given page in the Web document space. This section considers:

- a *partial overlap* scenario, in which 50% of the Web document space is common to all clients (accessed from any child proxy), and 50% is of regional interest only (accessed from only one child proxy)
- a *no overlap* scenario, in which the two child proxies handle reqeusts for completely disjoint document sets

Results for the *partial overlap* workload scenario are not shown here, for space reasons, but are available in [8]. These results follow the same trend as in the complete overlap case, except for a slight improvement in hit ratios for the child cache, and a noticeable drop in the hit ratios of the parent cache.

The explanation for this behaviour is the reduced overlap in the workloads of the two lower level proxies. That is, the 50% overlap in the workloads means that the left-most child proxy exclusively sees requests to 25% of the aggregate document set, the right-most child proxy exclusively sees requests to a different 25% of the aggregate document set, and the remaining 50% of the documents are (typically) seen by both proxies.

The partial overlap workload assumption has two important consequences. First, each child proxy sees only a subset (75%) of the total document space, which means fewer documents contending for cache space. Second, references to a particular document, as generated by ProWGen's document popularity and temporal locality models [8, 9, 20], may now be concentrated on a single child proxy, rather than randomly split across the two proxies. Again, this translates into better caching performance at the child proxies. On the other hand, the reduced overlap in the workload implies worse caching performance at the parent cache: the probability that a file requested by one child (and pulled into the parent cache from the origin server prior to delivering to the child) will also be requested later by the other child is about 50%, instead of almost 100% (ignoring one-timers) in the complete overlap scenario. Furthermore, repeated hits at the parent cache for such a document can only occur if there are repeated capacity misses at the child level.

The results for the no overlap scenario (not shown here) show the same trend: further improvement in the performance of the children, and a further decrease in the performance of the parent cache [8]. This trend is consistent, regardless of the replacement policies used.

For the no overlap scenario, the only role for the parent cache is to serve capacity misses from the lower level caches. In such a case, the GD-Size policy at the parent cache has the best performance, since it will store the most documents. The results show that the parent cache has its highest hit ratio when the size of the child cache is about 1-2 GB: about 10% of the total size of the Web content accessed. As the child caches grow larger, fewer capacity misses occur, and the relative benefit of the parent cache diminishes.

In summary, the effectiveness of caching hierarchies diminishes when there is little overlap in the Web workloads seen at the lowest-level proxies. For the scenarios and workloads studied here, the LRU or LFU-Aging policies at the lower level combined with GD-Size at the upper level always provide improvement in performance over an LRU-LRU combination. The GD-Size policy often provides twice the document hit ratio of other policies at the parent cache, without any penalty in byte hit ratio. Using GD-Size at both levels improves the document hit ratio, but sacrifices the byte hit ratio.

### 3.3. Size-based partitioning

The next caching strategy evaluated is *size-based partitioning*. That is, based on a specified size threshold $S$, the lower level caches are allowed to store only files smaller than $S$, while the upper level cache is allowed to store only files of size $S$ or larger. This simple policy provides a natural partitioning of the document space, using a minimal amount of information, which is directly available to Web servers and proxies in the HTTP response header.

For completeness, the converse of this policy is also considered, namely large files at the lower level, and small files at the upper level. With either of these approaches, distinct documents are maintained at each level of the hierarchy. Some replication of documents in multiple caches at the child level of the hierarchy is still possible.

The first design issue for size-based partitioning is the choice of the threshold size $S$. To understand the impact of different threshold sizes, three values are considered: 5,000 bytes, 10,000 bytes, and 100,000 bytes. For each chosen threshold size, heterogeneous replacement policies are studied. For space reasons, only the results for the partial overlap workload scenario and the LRU replacement policy at the lower level caches are presented here. Complete results are available in [8].

Figure 4 shows the results for a threshold size of 5,000 bytes. The top two graphs (Figures 4(a) and (b)) show the results when small files are kept at the lower level of the hierarchy, and large files at the upper level. The bottom two graphs (Figures 4(c) and (d)) are for the converse policy.

Figures 4(a) and (b) show that with size-based partitioning, the child caches have higher hit ratios than the parent cache (Figure 4(a)), but the parent cache achieves a much higher byte hit ratio (Figure 4(b)). The result for the parent cache is particularly interesting, in that it is able to achieve significant document hit ratios as well as byte hit ratios. This behaviour can be attributed to the high proportion of small files in the workload: about 60% of the requests are for files below this threshold size. The penalty for not keeping the large files (files $\geq$ 5,000 bytes) in the lower level cache is the lower byte hit ratios, as observed in Figure 4(b). The significant hit ratios and byte hit ratios achieved by the parent cache indicate that many references occur to files stored in its cache, and these files are responsible for a significant fraction of the total volume of data transferred.

Figure 4(a) shows that the GD-Size policy still provides the best document hit ratio at the parent cache, among the policies considered. However, its performance advantage over LRU and LFU-Aging has diminished significantly from that in Figure 3(a). Furthermore, it has a slight disadvantage in terms of byte hit ratio (Figure 4(b)) at large cache sizes.

The flattening of the hit ratio plots for the child caches beyond a cache size of 2 GB indicates a form of "cache ineffectiveness" beyond this point. That is, while increasing the cache size beyond 2 GB can improve the performance of the parent cache, it has no further benefit for the child caches. The reason for this is that the child cache is already large enough to accommodate all requested files smaller than 5,000 bytes, without any replacements required for the workloads generated. The hit ratios thus stabilize for this "infinite" cache size.

The performance results for the converse policy (i.e., keeping large files at the lower level and small files at the upper level) are shown in Figures 4(c) and (d). In these graphs, the parent cache shows consistently better document hit ratios than the child caches (Figure 4(c)). This is not surprising: the child proxies are not allowed to cache files smaller than 5,000 bytes, and these files account for a large fraction of the requests. While the observed document hit ratios at the children are low, the byte hit ratios (Figure 4(d)) are better than for the parent cache.

Figure 4(d) also shows that with the converse size-based partitioning approach, there are no noticable differences in byte hit ratio performance for the three different replacement policies considered at the parent cache (which is not allowed to cache large files). In other words, the performance impact of the size-based threshold scheme between levels of the proxy hierarchy is so dominant that the precise replacement policy used at the upper level is irrelevant. LRU, LFU-Aging, and GD-Size are equally effective (or ineffective) at the parent cache.

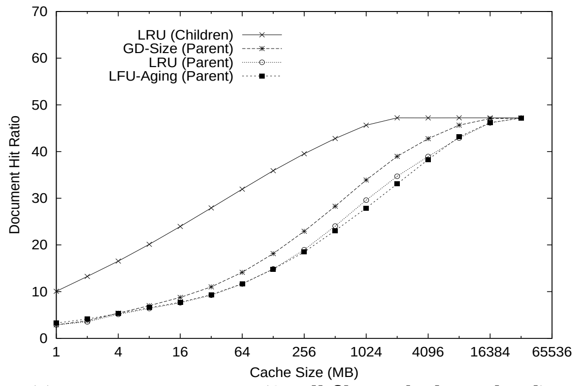
The results for size-based partitioning with a threshold size of 10,000 bytes (not shown here) show a significant improvement in performance for the lower level proxies, while the performance of the parent cache decreases [8]. This trend occurs because the percentage of files that can be cached at the lower level of the hierarchy increases: approximately 80% of the requests are for files smaller than 10,000 bytes. The byte hit ratio for the client caches shifts up noticeably. However, the drop in byte hit ratio for the parent cache is modest compared to the drop in document hit ratio because the large files cached at the upper level still contribute a significant fraction of the total bytes.

The results for the reversed threshold policy (i.e., keeping large files at the lower level, and files smaller than 10,000 bytes at the upper level) show that there is an increase in hit ratios for the parent cache, while the hit ratios for the child proxies decrease [8]. By keeping only large files in the lower level proxies, the byte hit ratio is still high, but the document hit ratio is low because of fewer references to the large files. The GD-Size policy provides the best document hit ratio at the upper level, without compromising the byte hit ratio. Again, the byte hit ratio results for the parent level cache are largely independent of the replacement policy used.

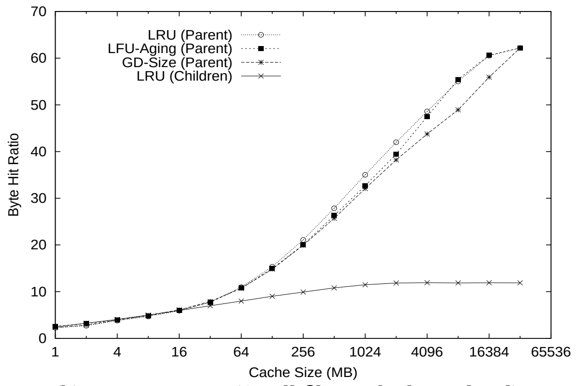
Increasing the size threshold further ($S = 100,000$ bytes, not shown here) continues the same trends indicated previously [8]. When small files are kept at the lower level, the document hit ratio at the parent drops drastically, though the byte hit ratio at the parent cache is still significant. Reversing the size threshold restriction improves hit ratios at the parent, but at the expense of the lower level proxies.

## 3.4. Summary

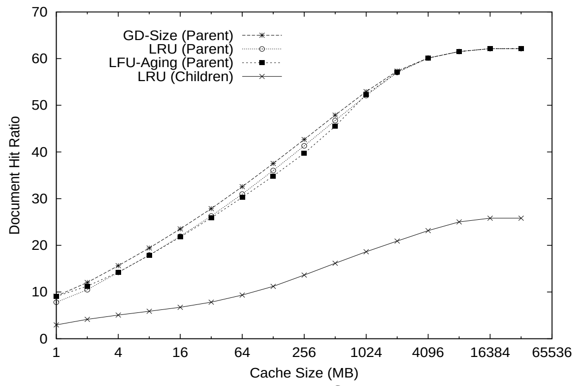
The foregoing experiments have illustrated the performance tradeoffs, in terms of document hit ratio and byte hit ratio, at both child-level and parent-level caches in a two-level Web proxy caching hierarchy. Several novel cache management strategies were explored, including heteroge-



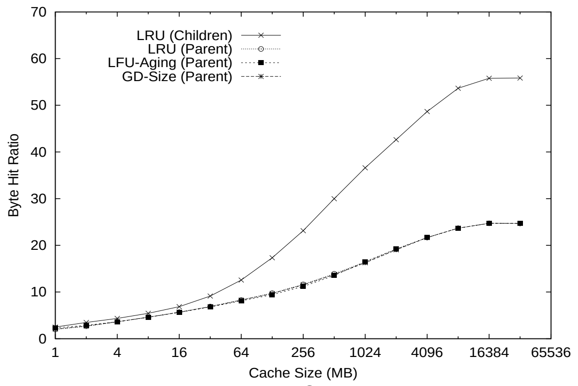
(a) Document Hit Ratio (Small files at the lower level)



(b) Byte Hit Ratio (Small files at the lower level)



(c) Document Hit Ratio (Large files at the lower level)



(d) Byte Hit Ratio (Large files at the lower level)

**Figure 4. Simulation results for size-based partitioning (Threshold = 5,000 Bytes)**

neous cache replacement policies, and the partitioning of Web document content across the levels of the caching hierarchy based on document size.

The simulation results suggest performance advantages for the use of heterogenous replacement policies across the levels of a caching hierarchy. Determining which combination of policies is "best", in terms of user-perceived response time, requires in-depth consideration of network capacity, network latencies, server load, HTTP, and TCP-level effects [15, 17]. Such a rigourous evaluation is beyond the scope of the current paper. Further discussion of this issue, and an approximate cost-benefit analysis of selected caching strategies, appears in [8].

## 4. Conclusions

This work used synthetic Web proxy workloads and trace-driven simulation to evaluate several different approaches to cache management for a two-level Web proxy caching hierarchy. In particular, the experiments consider heterogeneous replacement policies within the hierarchy, and size-based partitioning of documents across the levels of the hierarchy.

Simulation results show that combining different replacement policies at different levels of the hierarchy can improve the performance of a caching hierarchy. The best performance was typically provided by the use of LRU or LFU-Aging at the lower level, combined with GD-Size at the upper level. Using GD-Size at both levels provides a better document hit ratio, but sacrifices the byte hit ratio. For file partitioning, the simulation experiments show that size-based partitioning (with small files at the lower level of the hierarchy) can improve performance. However, the performance improvements are sensitive to the size threshold chosen, and to the degree of overlap in the workloads of the child-level proxies.

Future work will study network-level effects in Web proxy caching hierarchies. We also plan to extend our proxy workload generation tool to model document modifications, and extend our Web caching simulator to study cache consistency issues.

## Acknowledgements

The authors thank Rick Bunt, Anirban Mahanti, and the anonymous reviewers for their comments and feedback on earlier versions of this paper.

Financial support for this research was provided by TR*Labs* (Telecommunications Research Laboratories) in Saskatoon, and by NSERC research grant OGP0121969.

The ProWGen tool is available from
http://www.cs.usask.ca/faculty/carey/software/

## References

[1] G. Abdulla, E. Fox, M. Abrams, and S. Williams, "WWW Proxy Traffic Characterization with Application to Caching", Technical Report TR-97-03, Computer Science Department, Virginia Tech., March 1997.

[2] M. Arlitt, L. Cherkasova, J. Dilley, R. Friedrich, and T. Jin, "Evaluating Content Management Techniques for Web Proxy Caches", *ACM Performance Evaluation Review*, Vol. 27, No. 4, pp. 3-11, March 2000.

[3] M. Arlitt and C. Williamson, "Trace-Driven Simulation of Document Caching Strategies for Internet Web Servers", *Simulation Journal*, Vol. 68, No. 1, pp. 23-33, January 1997.

[4] M. Baentsch, L. Baum, G. Molter, S. Rothkugel, and P. Sturm, "World-Wide Web Caching: The Application Level View of the Internet", *IEEE Communications*, Vol. 35, No. 6, pp. 170-178, June 1997.

[5] M. Baentsch, L. Barum, G. Molter, S. Rothkugel, and P. Sturm, "Enhancing the Web's Infrastructure: From Caching to Replication", *IEEE Internet Computing*, Vol. 1, No. 2, pp. 18-27, March 1997.

[6] A. Bestavros, R. Carter, M. Crovella, C. Cunha, A. Heddaya, and S. Mirdad, "Application-Level Document Caching in the Internet", *Proceedings of the 2nd International Workshop on Services in Distributed and Networked Environments (SDNE'95)*, pp. 166-173, Whistler, BC, June 1995.

[7] J. Bolot and P. Hoschka, "Performance Engineering of WWW: Applications to Dimensioning and Cache Design", *Electronic Proceedings of 5th Inter. World-Wide Web Conference*, Paris, France, May 1996.

[8] M. Busari, *Simulation Evaluation of Web Caching Hierarchies*, M.Sc. Thesis, Department of Computer Science, University of Saskatchewan, June 2000.

[9] M. Busari and C. Williamson, "On the Sensitivity of Web Proxy Cache Performance to Workload Characteristics", *Proceedings of IEEE INFOCOM'2001*, pp. 1225-1234, Anchorage, Alaska, April 2001.

[10] H. Che, Z. Wang, and Y. Tung, "Analysis and Design of Hierarchical Web Caching Systems", *Proceedings of IEEE INFOCOM'2001*, pp. 1416-1424, Anchorage, Alaska, April 2001.

[11] P. Cao and S. Irani, "Cost-Aware WWW Proxy Caching Algorithms", *Proceedings of USENIX Symposium on Internet Technologies and Systems (USITS'97)*, pp. 193-206, December 1997.

[12] A. Chankhunthod, P. Danzig, C. Neerdaels, M. Schwartz and K. Worrell, “A Hierarchical Internet Object Cache”, *Proceedings of the 1996 USENIX Technical Conference*, pp. 153-163, San Diego, CA, January 1996.

[13] E. Cohen, B. Krishnamurthy, and J. Rexford, “Improving End-to-End Performance of the Web Using Server Volumes and Proxy Filters”, *Proceedings of ACM SIGCOMM’98 Conference*, pp. 241-253, Vancouver, BC, September 1998.

[14] L. Fan, P. Cao, J. Almeida, and A. Broder, “Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol”, *Proceedings of ACM SIGCOMM’98 Conference*, pp. 254-265, Vancouver, BC, September 1998.

[15] A. Feldmann, R. Caceres, F. Douglis, G. Glass, and M. Rabinovich, “Performance of Web Proxy Caching in Heterogeneous Bandwidth Environments”, *Proceedings of IEEE INFOCOM’99*, pp. 107-116, April 1999.

[16] J. Gwertzman and M. Seltzer, “An Analysis of Geographical Push-Caching”, *Proceedings of 5th IEEE Workshop on Hot Topics in Operating Systems*, pp. 51-55, Orcas Island, WA, May 1995.

[17] J. Heideman, K. Obraczka, and J. Touch, “Modeling the Performance of HTTP Over Several Transport Protocols”, *IEEE/ACM Transactions on Networking*, Vol. 5, No. 5, pp. 616-630, October 1997.

[18] A. Mahanti, C. Williamson, and D. Eager, “Traffic Analysis of a Web Proxy Caching Hierarchy”, *IEEE Network*, Vol. 14, No. 3, pp. 16-23, May/June 2000.

[19] A. Mahanti and C. Williamson, “Web Proxy Workload Characterization”, Technical Report, Department of Computer Science, University of Saskatchewan, February 1999.
http://www.cs.usask.ca/faculty/carey/papers/workloadstudy.ps

[20] A. Mahanti, D. Eager, and C. Williamson, “Temporal Locality and its Impact on Web Proxy Cache Performance”, *Performance Evaluation*, Vol. 42, No. 2-3, pp. 187-203, October 2000.

[21] N. Nishikawa, T. Hosokawa, Y. Mori, K. Yoshika, and H. Tsuji, “Memory-based Architecture for Distributed WWW Caching Proxy”, *Proceedings of World-Wide Web Conference*, pp. 205-214, April 1998.

[22] D. Povey and J. Harrison, “A Distributed Internet Cache”, *Proceedings of the 20th Australian Computer Science Conference*, Sydney, Australia, February 1997.

[23] F. Quaglia, B. Ciciani, and M. Colajanni, “An Analytical Comparison of Cooperation Protocols for Web Proxy Servers”, *Proceedings of MASCOTS’99*, pp. 174-181, College Park, MD, October 1999.

[24] P. Rodriguez, C. Spanner, and E. Biersack, “Web Caching Architectures: Hierarchical and Distributed Caching”, *Proceedings of the Fourth Web Caching Workshop*, San Diego, CA, pp. 37-48, March 1999.

[25] R. Tewari, M. Dahlin, H. Vin, and J. Kay, “Beyond Hierarchies: Design Considerations for Distributed Caching on the Internet”, *Proceedings of the 19th International Conference on Distributed Computing Systems*, Austin, TX, June 1999.

[26] V. Valloppillil and K. Ross, “Cache Array Routing Protocol v1.1”, Internet Draft, February 1998.

[27] D. Weikle, S. McKee, and W. Wulf, “Caches as Filters: A New Approach to Cache Analysis”, *Proceedings of MASCOTS’98*, Montreal, PQ, pp. 2-12, July 1998.

[28] D. Wessels and K. Claffy, “ICP and the Squid Web Cache”, *IEEE Journal on Selected Areas in Communication*, Vol. 16, No. 3, pp. 345-357, April 1998.

[29] D. Willick, D. Eager, and R. Bunt, “Disk Cache Replacement Policies for Network Fileservers”, *Proceedings of the 13th International Conference on Distributed Computing Systems (ICDCS)*, pp. 2-11, Pittsburgh, PA, May 1993.

[30] P. Yu and E. MacNair, “Performance Study of a Collaborative Method for Hierarchical Caching in Proxy Servers”, *Proceedings of World-Wide Web Conference*, pp. 215-224, April 1998.

[31] L. Zhang, S. Floyd, and V. Jacobson, “Adaptive Web Caching”, *Proceedings of the NLANR Web Caching Workshop*, Boulder, CO, June 1997.