Hint-based Acceleration of Web Proxy Caches

Daniela Roşu Arun Iyengar Daniel Dias
IBM T.J.Watson Research Center
P.O.Box 704, Yorktown Heights, NY 10598

Abstract

Numerous studies show that proxy cache miss ratios are typically at least 40%-50%. This paper proposes and evaluates a new approach for improving the throughput of proxy caches by reducing cache miss overheads. An embedded system able to achieve significantly better communications performance than a traditional proxy filters the requests directed to a proxy cache, forwarding the hits to the proxy and processing the misses itself. This system, called a Proxy Accelerator, uses hints of the proxy cache content and may include a main memory cache for the hottest objects. Scalability with the Web proxy cluster size is achieved by using several accelerators. We use analytical models and trace-based simulations to study the benefits and the implementation tradeoffs of this new approach. A single proxy accelerator node in front of a four-node Web proxy improves the cost-performance ratio by about 40%. Implementation choices that do not affect the overall hit ratio are available.

1 Introduction

Proxy caches for large Internet Service Providers can receive high request volumes. Numerous studies, such as [4, 6, 9], show that miss ratios are often at least 40%-50%, even for large proxy caches. Consequently, significant performance improvements may be achieved by reducing cache miss overheads.

Based on these insights and on our experience with Web server acceleration [12], this paper proposes and evaluates a new approach to improving the throughput of proxy caches. Software running under an embedded operating system optimized for communication, like the Web server accelerator described [12], takes over some of the Web proxy's operations. This system, which we call a *proxy accelerator,* (see Figure 1) forwards the hits to the Web proxy and processes the cache misses itself. Therefore, the Web proxy's miss-related overheads are reduced to receiving cacheable objects from the accelerator.

The information about proxy caches that is used in filtering the requests is called *hint information* and might not provide 100% accurate information. The hints are maintained based on information provided by proxy nodes and information derived from the accelerator's own decisions.

Besides hints, the accelerator may have a main memory cache. An accelerator cache can serve objects with considerably less overhead than from a proxy cache partly because of the embedded operating system on which the accelerator runs and partly because all objects are cached in main memory.

Previous research has considered the use of location hints in the context of cooperative Web proxies [5, 16, 17, 7]. Our system extends this approach to the management of a cluster-based Web proxy. In addition, our work extends the previous solutions to cache redirection [1, 10, 13, 11] by having the router (i.e., the proxy accelerator) off-load the proxy by processing some or all of the cache misses itself.

We use analytical models in conjunction with simulations to study the performance improvement enabled by hint-based acceleration. We evaluate the impact of several hint representation and maintenance methods on throughput, hit ratio, and resource usage.

Our analytical study reveals that the expected throughput improvement using a single accelerator node is greater than 100% for a four-node Web proxy when the accelerator can achieve about an order of magnitude better throughput than a Web proxy node for communications operations. Our simulations validate that with appropriate hint representations and update protocols, the proposed acceleration method can offload more than 50% of the WP load with no impact on the observed hit ratio.

Paper Overview. The rest of the paper is structured as follows. Section 2 describes the architecture of an accelerated Web proxy. Section 3 analyzes the expected throughput improvements with an analytical model. Section 4 evaluates the impact of the hint management on several Web proxy performance metrics with a trace-driven simulation. Section 5 discusses related work, and Section 6 summarizes our results.

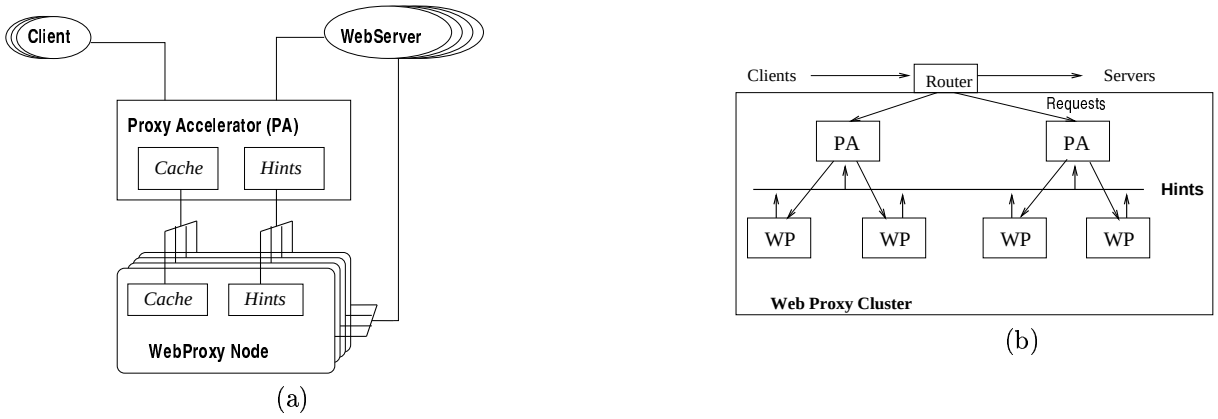

Figure 1: Interactions in a Web Proxy cluster with one or more Proxy Accelerators (PA).

2 Accelerated Web Proxy

This paper proposes a new approach to speeding up proxy caches by reducing their cache-miss overheads. In the new approach, the Web Proxy (WP) configuration includes an extended content-based router called a proxy accelerator (PA). A PA is typically built from stock hardware and runs under an embedded operating system optimized for communication. A PA can achieve an order of magnitude better throughput than a Web proxy node for communications operations.

The PA filters client requests, distinguishing between cache hits and misses based on hints about proxy cache content. In the case of a cache hit, the PA hands off the request to a WP node that contains the object. In the case of a cache miss, the PA sends the request to the Web site, bypassing the WP. When it receives the object from the Web site, the PA replies to the client, and if appropriate, pushes the object to a WP node.

The PA may include a main memory cache for storing the hottest objects. Requests satisfied from a PA cache never reach the WP, further reducing the traffic seen by the WP. The PA cache is populated with objects received directly from remote Web sites or objects pushed by WP nodes.

Figure 1 (a) illustrates the interactions of a PA in a four-node WP cluster. Notice that WP nodes still interact with Web sites. This occurs when the PA filter is not accurate, or the PA is overloaded and does not apply the content filter to all requests. For scalability, the system may include several PAs (Figure 1 (b)) and a router that balances the load among them. Each PA has hints about each of the WP nodes. Therefore, a PA can appropriately forward any of the incoming hits. However, in order to balance the WP load across the cluster, a PA can only push objects to a subset of the WP nodes.

Besides typical functionality, a WP node integrates procedures for hint maintenance and distribution, for handoff of TCP/IP connections and URL requests, and for pushing objects to PA caches.

Hint Representation. Previous research has considered multiple hint representations and consistency protocols. Representations vary from schemes that provide accurate information [17, 7] to schemes that trade accuracy for reduced costs (e.g. memory and communication requirements, look-up and update overheads) [5, 16].

In this study, we consider two representations – a representation that provides accurate information and a representation that provides approximate (inaccurate) information. The accurate-information scheme, called the *directory scheme*, uses a list of object identifiers including an entry for each object known to exist in the associated WP cache. In our implementation, the list is represented as a hash table [7, 17].

The approximate-information scheme is based on *Bloom Filters* [2] (see Figure 2). The filter is represented by a bitmap, called *hint space*, and several hash functions. To register an object in the hint space, the hash functions are applied to the object identifier, and the corresponding entries are set in the hint space. A hint look-up is positive if all of the entries associated with the object of interest are set. A hint entry is cleared when no object is left in the cache whose representation includes the entry. Therefore, the cache owner (i.e., the WP node) maintains a "collision" counter for each entry. When the WP has several nodes, the corresponding hint spaces can be represented independently or can be interleaved such that the corresponding hint entries are located in consecutive memory locations (see Figure 2 (b)).

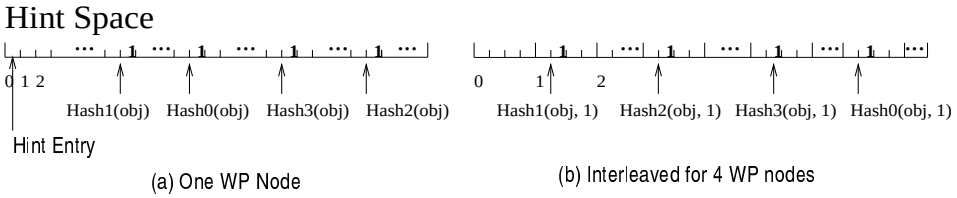

Figure 2: Hint representation based on Bloom Filters.

Hint Consistency. The hint consistency protocol ensures that the information at the PA site reflects the content of the WP cache. Protocol choices vary in several dimensions: the entity that identifies the updates, the protocol initiator, the rate of update messages, and the exchanged information. In the previously proposed solutions, updates are identified either by the WP (called *WP-only updates*) [5, 16] or the content-based router (PA) [13]. Besides these methods, we propose a method, called *eager hint registration*, in which both PA and WP identify and record hints. More specifically, when the PA forwards a connection or pushes an object to a WP node, it "eagerly" registers that the object exists in the corresponding cache. Consequently, a WP node sends updates only for cache removals, for additions of objects received from sources other than the PA, and for failures to cache objects sent by the PA.

With respect to the protocol initiator, previously proposed solutions address only WP-only updates. Updates are pulled by the hint owner (i.e., PA) [16] or pushed by the cache owner (i.e., WP) [5, 17, 7]. With respect to the update rate, message updates may be sent at fixed time intervals [16] or after a fixed number of cache updates [5]. The exchanged information may be incremental, addressing only the updates occurred since the previous message [7, 5, 16], or a copy of the entire hint space [5].

Proxy Accelerator Cache. The PA cache may include objects received from Web sites or pushed by the associated WP nodes. The PA cache replacement policy should maximize the hit ratio. Therefore, the policy may be different than the WP cache policy, which typically aims at minimizing the network-related costs. Therefore, policies based on hotness statistics and object sizes are expected to enable better PA performance than policies like Greedy-Dual-Size [3] which are more appropriate for the WP.

In the following sections, we evaluate the throughput potential of the proposed proxy acceleration method and the impact of several hint implementation choices. Insights about the impact of the cache management mechanism can be found in [14].

3 Throughput Improvements

The major benefit of the proposed hint-based acceleration of Web Proxies is throughput improvement. We analyze the throughput properties of this scheme with an analytic model by varying the number of WP and PA nodes, the PA functionality and cache size, and the WP's and PA's CPU speeds.

The system is modeled as a network of M/G/1 queues. The servers represent PAs, WP nodes and disks, and a Web site. The serviced events represent (1) stages in the processing of a client request (e.g. PA Miss, WP Hit, Disk Access) and (2) hint and cache management operations (e.g. hint update, object push). Event arrival rates derive from the rate of client requests and the likelihood of a request to switch from one processing stage to another. The likelihood of various events, such as cache hits and disk operation, are derived from several studies on Web Proxy performance [3, 15]. For instance, we consider a WP hit ratio of 40% and PA hit ratios up to 30%.

The overhead associated with each event includes a basic overhead, I/O overheads (e.g. connect, receive message, handoff), and operating system overheads (e.g. context switch). The basic overheads are identical for the PA and WP, but the I/O and operating system overheads are different. Namely, the PA overheads correspond to the embedded system used for Web server acceleration in [12] – a 200 MHz Power PC PA which incurs a $514\mu sec$ miss overhead (i.e., 1945 misses/sec) and $209\mu sec$ hit overhead (4783 hits/sec). The WP overheads are derived from [13] – a 300 MHz Pentium II which incurs a $2518\mu sec$ miss overhead (i.e., 397 misses/sec) and $1392\mu sec$ hit overhead (718 hits/sec). These overheads assume the following: (1) accurate hints with 1 hour incremental updates; (2) 8 KByte objects; (3) disk I/O is required for 75% of the WP hits; (4) one disk I/O per object; (5) four network I/Os per object (from WS, to client, to WP, from WP); (6) 25% of WP hits are pushed to the PA; (7) $30\mu sec$ WP context switch overhead. The costs of the hint-related operations derive from our implementation and the simulation-based study (see Section 4).

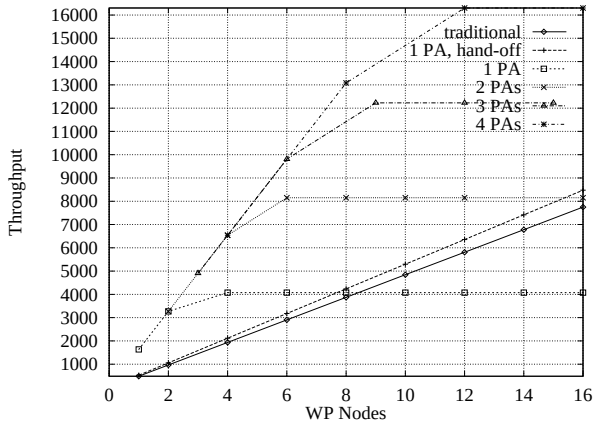

Figure 3: Variation of throughput with number of WP and PA nodes. 300 MHz, no cache PAs, 40% WP hit ratio, 300 MHz WP.

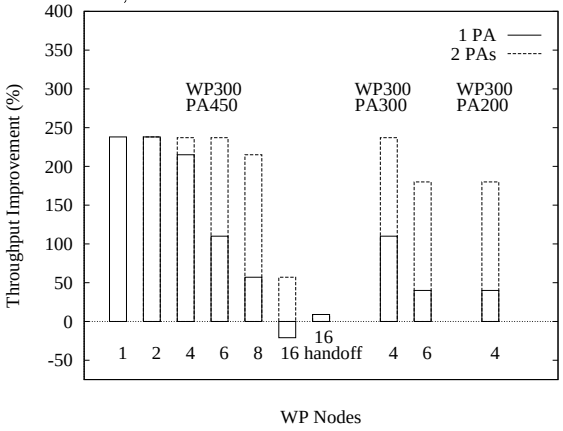

Figure 4: Variation of throughput improvement with number and speed of WP and PA nodes. No PA cache, 40% WP hit ratio.

Hint-based acceleration improves WP throughput. Figure 3 illustrates that the hint-based acceleration of a WP enables significant performance improvements. The plot presents the expected performance for a traditional WP-cluster and for clusters enhanced with one to four PAs. WPs and PAs have the same computation power (300 MHz), and PAs have no cache.

The plot shows that when not a bottleneck, the PA can significantly improve the performance. The cost-performance improvement achieved with a single PA is ≈38% in a 4-node WP and ≈55% in a 2-node WP. However, the plot highlights that for large WP clusters, the architecture should integrate several PAs. Reducing the PA load by having it hand off the client and server connections to be completed by a WP node (i.e., receive from server and send to client)

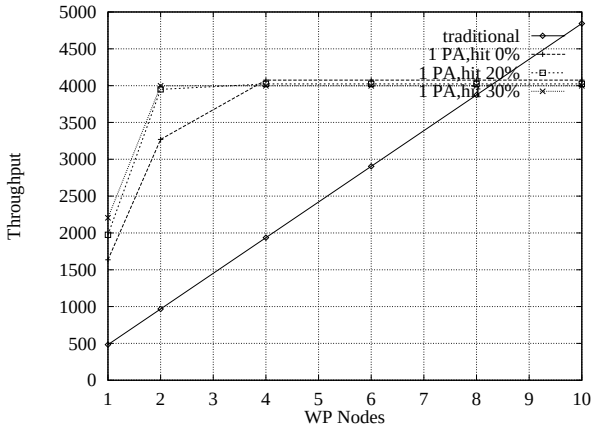

Figure 5: Variation of throughput with PA cache hit ratio. 300 MHz PA, 40% WP hit ratio, 300 MHz WP.

cannot yield significant performance improvements – only about 1% per WP node (see label *handoff*).

The performance benefits of the scheme increase with the PA's CPU speed. Figure 4 presents the expected throughput improvement for several cluster sizes and PA CPU speeds. For 300 MHz WP nodes, with only one 450MHz PA, the improvement is larger than 200% for WPs with no more than 4 nodes, and larger than 100% for 6-node WPs.

PA-level caches improve throughput when WP performance is moderate. Figure 5 illustrates the effect of a PA cache. In the context of a one or two-node WP, the PA cache enables about 20-30% throughput improvement. However, for larger WP clusters, the overhead of handling cache hits makes the PA become a bottleneck earlier (i.e., 2 WP nodes, 30% hits) than in a configuration with no cache. In addition, the overhead of object push from WP to PA affects the throughput when the WP is the bottleneck (see 2 WP nodes, 30% vs. 20% hits).

4 Impact of Hint Management

In this section, we evaluate how the hint representation and the consistency protocol affect the performance of an accelerated Web proxy. We focus on performance metrics, such as hit ratio and throughput, and on cost metrics, such as memory, computation, and communication requirements.

These performance and cost metrics are affected by various characteristics of the hint management mechanism (see Table 1). For instance, the look-up overhead affects the long-term PA throughput, and the false misses cause undue network loads and response delays. Table 2 summarizes some of the differences between representation methods and update protocols introduced in Section 2. For instance, the look-

up overhead is constant for the Bloom Filter-based scheme while for the directory scheme it depends on the context, namely on how well the hash table is balanced. Similarly, eager registration results in no false miss, while for WP-only registration the amount of false misses increases with the update period.

In this study, we identify and compare relevant trends of these representation methods and update protocols. The study is conducted with a trace-driven simulator and a segment of the Home IP Web traces collected at UC Berkeley in 1996 [8]. This traces include about 5.57 million client requests for cacheable objects over a period of 12.67 days. The number of objects is about 1.68 million with a total size of about 17 GBytes.

Factors. The factors considered in this study are the following: (1) PA and WP memory, (2) hint representation, (3) hint update protocol, and (4) update period. Namely, we consider configurations with 0.25-1 GByte PA memory and 1-10 GByte WP memory. For the Bloom Filter-based scheme, we vary the size of the hint space (0.5-10 MBytes) and the number of hash functions; each hash function returns the remainder of the hint space size divided by an integer obtained as a combination of four bytes in the object identifier. Due to limitations associated with trace encoding, object identifiers are composed of the 16-byte MD5 representation of the URL and a 4-byte encoding of the server name. Finally, we experiment with update periods in the range of (1 sec, 2 hours).

Fixed Parameters. We experiment only with "push" updates; in comparison to "pull" updates, "push" updates allow better control of hint accuracy [5] and lower I/O overheads. In addition, we experiment only with incremental updates, as we consider it more appropriate than "entire copy" updates in the context of the hint representations and system parameters considered in this study. The WP cache replacement policy is LRU. The PA cache replacement is "LRU with bounded replacement burst". According to this policy, an incoming object is not cached if this would cause more than a given number of objects to be removed from the cache. In comparison to traditional LRU, this policy achieves a larger PA hit ratio [14] and results in more predictable PA overheads.

4.1 Cost Metrics

Memory Requirements. The main-memory requirements of the hint mechanism derive from the hint meta-data. For the directory scheme, no meta-data is maintained at the WP, and at the PA, the requirements increase linearly with the population of the WP cache. For instance, in our implementation, with 68-byte entries, the meta-data is $\approx$70 MBytes for a 10 GByte WP cache. By contrast, for the Bloom Filter-based scheme, meta-data is maintained at both the PA and WP and is independent of the cache size. The PA meta-data is as large as the hint space multiplied by the number of WP nodes, and the WP meta-data is as large as the hint space multiplied by the size of the collision counter (e.g. 8 bits).

Computation Requirements. The computation requirements associated with our acceleration scheme result mainly from the request look-ups and the processing of update messages. In addition, minor computation overheads may be incurred at WP and PA upon each cache update.

Look-up Overhead. For the directory scheme, the look-up overhead depends on the distribution of objects to hash table buckets. For the Bloom Filter-based scheme, the look-up overhead is more predictable. It includes the MD5 computation and evaluation of related hash functions. On a 133 MHz PowerPC, the MD5 computation takes $\approx 22\mu sec$ per 56 byte string segment, and the evaluation of a hash function takes $\approx 0.6\mu sec$. With an interleaved implementation (see Figure 2 (b)), the look-up overhead is constant at $\approx 2.9\mu sec$. By contrast, with separate per-node representations, the worst case (i.e. miss) look-up overhead varies with the number of WP nodes and hash functions. For instance, for the four simple hash functions used in our Bloom Filter implementation, the overhead is $\approx 2.6\mu sec$ for one node and $9.5\mu sec$ for four nodes. Overall, the look-up overhead is lower than 10% of the expected cache hit overhead on a 300MHz PA (see Section 3).

Hint Consistency Protocol. The amount of computation triggered by the receipt of a hint consistency message depends on the number of update entries included in the message. Because of hint entry collisions, the Bloom Filter-based scheme results in about half the load of the directory scheme (with 160-byte object identifiers). The variation of the maximum and average message sizes presented in Figure 10 and Figure 11 illustrates this trend. These figures illustrate also the benefit of eager registration approach. For both hint schemes, on average, the eager hint registration reduces by half the computation loads. Similar trends are observed for the communication requirements of the consistency protocol. Note that the very large maximums observed in Figure 10 are due to few very large WP cache replacement bursts ($\approx$2300 objects).

4.2 Performance Metrics

Hit Ratio. The characteristic of the hint mechanism that most affects the (observed) hit ratio is the *likeli-*

Table 1: Impact of Hint Management Choices.

Characteristic	Affects
memory at WP	WP memory hit ratio
memory at PA	PA cache hit ratio
update period	long-term throughput
update period	WP/PA short-term throughput
look-up time	PA long-term throughput
false hits	WP load
false misses	network usage

Table 2: Selected Method Characteristics.

Characteristic	Representation	
	Directory	Bloom Filters
memory at WP	none	O(hint space)
memory at PA	O(objs WP)	O(hint space)
look-up overhead	context-dependent	constant
false hits	none	O(objs WP)
	Protocol	
	WP-only	Eager Reg.
false misses	period-dependent	none
PA computation	on-off, large bursts	smaller bursts

hood of false misses. Typically, false misses are due to hint representation and update delays. In our experiments, because of the selected hint representations, false misses are only the result of update delays.

As expected, the likelihood of false misses increases with the update period. Figure 7 illustrates this trend indirectly: the ratio of requests directed to the WP with a traditional update mechanism decreases with the increase of the update period. Eager hint registration reduces the number of observed false misses, with a significant impact on the directory scheme.

Another relevant hint mechanism characteristic is the *amount of hint meta-data at the WP*, which has a small impact on the overall hit ratio (i.e., hits in both PA and WP caches). The impact is small but more significant when the ratio of the meta-data in the WP memory is large, as for small WP cache sizes (Figure 6). PA meta-data has no significant impact because objects removed from the PA cache are pushed to the WP rather than being discarded.

Throughput. The extent of throughput improvement depends on how well the PA identifies the WP cache misses. This characteristic, reflected by the *likelihood of false hits*, is affected by the representation method and the consistency protocol.

Hint Representation. While the directory scheme has no false hits, for the Bloom Filter-based scheme, the likelihood of false hits increases as hint-space collisions increase. As illustrated in Figure 8, this occurs when the hint space decreases or the number of objects in the WP cache increases. We notice that the false-hit ratio is constant when the hint space increases beyond a threshold. This threshold depends on the WP cache size, and the effect is due to the characteristics of the selected hash functions. The likelihood of false hits also depends on the number of hint entries per object (for details see [14]).

Hint Consistency Protocol. The impact of update period on false hit ratio is more significant in configurations with frequent WP cache updates (e.g., systems with small WP caches). Figure 9 illustrates this trend for both hint schemes. For instance, for Bloom Filters, the difference between the false-hit ratios observed for 1-hour and for 1-sec updates is 0.07% for a 4 GByte cache and 0.04% for a 6 GByte cache. Comparing the two schemes, the relative impact of the update period is almost identical.

Summary. By trading off accuracy, the Bloom Filter-based scheme exhibits lower computation and communication overheads. However, the WP meta-data may reduce its memory-hit ratio. Hint look-up overheads are small with respect to the typical request processing overheads. The look-up overhead is more predictable for the Bloom Filter-based scheme. Eager hint registration reduces update-related overheads and prevents hit ratio reductions caused by update delays. False hit ratio (and, consequently, throughput) is more significantly affected by hint representation than by update period. For Bloom Filter-based schemes, this ratio increases exponentially with hint space contention.

5 Related Work

Previous research has considered the use of location hints in the context of cooperative Web Proxies [5, 16, 7, 17]. In these papers, the hints help reduce network traffic by more efficient Web Proxy cache cooperation. Extending these approaches, we use hints to improve the throughput of an individual Web proxy. In addition, we propose and evaluate new hint maintenance techniques that reduce overheads by exploiting the interactions between a cache redirector and the associated cache.

Web traffic interception and cache redirection relates our approach to architectures such as Web server accelerators [12], ACEdirector (Alteon) [1], DynaCache (InfoLibria) [10], Content Smart Switch [11], and LARD [13]. The ability to integrate a Proxy Accelerator-level cache renders our approach similar

to architectures in which the central element is an embedded system-based cache, such as a hierarchical caching architecture [10], or the front end of a Web server [12]. Our proxy accelerator extends the approach to bypassing the Web proxy proposed in [11]; besides non-cacheable objects, our accelerator can identify the misses and process them locally. Trading the throughput of the routing component, which is not the bottleneck resource, our approach can boost even more the throughput of the bottleneck proxy nodes. Furthermore, our approach is similar to the redirection methods used in architectures focused on content-based redirection of requests to Web Proxy nodes [1, 13]. However, our approach enables dynamic rather than fixed mappings of objects to WP nodes[1]. In contrast to the method in [13], redirection does not cause caching of multiple object replicas and is independent of client request patterns.

6 Conclusions

Based on the observation that miss ratios at proxy caches are often relatively high [4, 9, 6], we have developed a method to improve the performance of a cluster-based Web proxy by shifting some of its cache miss-related functionality to be executed on a Proxy Accelerator – an extended content-based router implemented on an embedded system optimized for communication. Consequently, the Web proxy can service a larger number of hits, and the Proxy Accelerator-based system can achieve better throughput than traditional Web proxies [1, 5, 16, 13] or systems in which the Web proxy is bypassed only for non-cacheable objects [11]. In addition, under moderate load, response time can be improved with a Proxy Accelerator main memory cache [12, 10].

Our study shows that a single Proxy Accelerator node with the same CPU speed as a Web Proxy node but with about an order of magnitude better throughput for communication operations [12] can improve the cost-performance ratio of a 4-node Web proxy by ≈38%. Considering the implementation, our study shows that eager registration is a "must" and that the Bloom Filter-based scheme is more appropriate than the directory scheme for large WP clusters or when PA and WP nodes have comparable power. In the future, we plan to extend the proposed Web proxy acceleration scheme to accommodate HTTP/1.1 traffic; the PA should maintain hints on the pool of client and Web site connections and should coordinate connection handoffs among the proxy nodes.

References

[1] Alteon Networks. New Options in Optimizing Web Cache Deployment. *white paper*.

[2] B. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of ACM*, pages 422–426, July 1970.

[3] P. Cao and S. Irani. Cost-Aware WWW Proxy Caching Algorithms. *USENIX Symposium on Internet Technologies and Systems*, 1997.

[4] B. M. Duska, D. Marwood, and M. J. Feeley. The Measured Access Characteristics of World-Wide-Web Client Proxy Caches. *USENIX Symposium on Internet Technologies and Systems*, 997.

[5] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol. *SIGCOMM'98*, 1998.

[6] A. Feldmann, R. Caceres, F. Douglis, G. Glass, and M. Rabinovich. Performance of Web Proxy Caching in Heterogeneous Bandwidth Environments. *IEEE INFOCOM'99*, pages 106–116, March 1999.

[7] S. Gadde, J. Chase, and M. Rabinovich. A Taste of Crispy Squid. *Workshop on Internet Server Performance*, 1998.

[8] Gribble, Steven D. UC Berkeley Home IP HTTP Traces. *http:// www.acm.org/ sigcomm/ITA*, July 1997.

[9] Gribble, Steven D. and Brewer, Eric A. System Design Issues for Internet Middleware Services: Deductions from a Large Client Trace. *USENIX Symposium on Internet Technologies and Systems*, 1997.

[10] A. Heddaya. DynaCache: weaving Caching into the Internet. *white paper*.

[11] E. Johnson. Increasing the Performance of Transparent Caching with Content-Aware Cache Bypass. *ArrowPoint Communications*, Mar.,1999.

[12] E. Levy, A. Iyengar, J. Song, and D. Dias. Design and Performance of a Web Server Accelerator. *IEEE INFOCOM'99*, 1999.

[13] V. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. Nahum. Locality-Aware Request Distribution in Cluster-based Network Servers. *International Conference on Architectural Support for Programming Languages and Operating Systems*, 1998.

[14] D. Ro¸su, A. Iyengar, and D. Dias. Hint-based Acceleration of Web Proxy Caches. *IBM Research Report RC21611*, Sept. 1999.

[15] A. Rousskov and V. Soloviev. A Performance Study of the Squid Proxy on HTTP/1.0. *WWW Journal*, (1-2), 1999.

[16] A. Rousskov and D. Wessels. Cache Digests. *http://squid.nlanr.net/Squid/CacheDigest*, 1998.

[17] R. Tewari, M. Dahlin, H. M. Vin, and J. S. Kay. Beyond Hierarchies: Design Considerations for Distributed Caching on the Internet. *International Conference on Distributed Computing Systems*, June 1999.

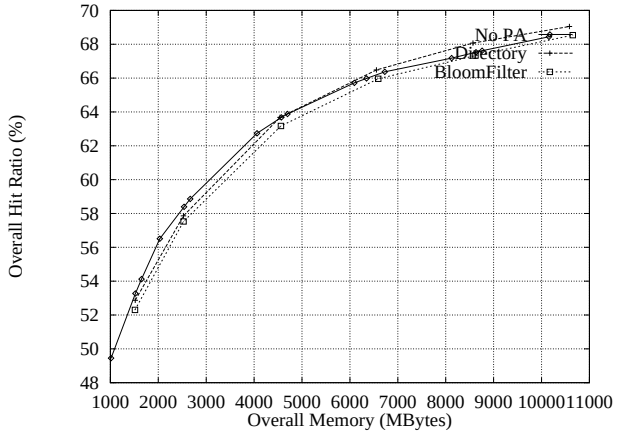

Figure 6: Variation of overall hit ratio with WP configuration. 512 MByte PA, Bloom Filter with 10 MByte hint space.

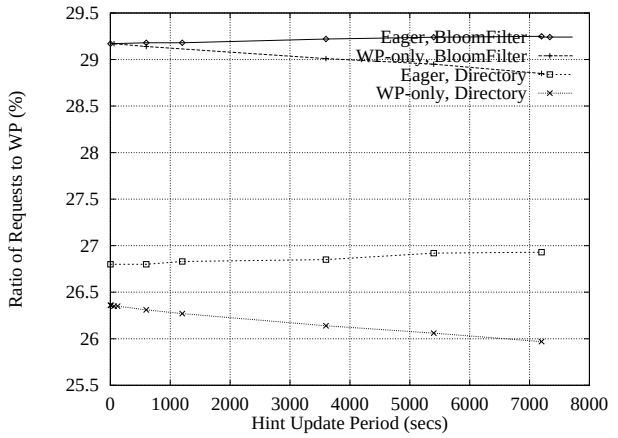

Figure 7: Variation of WP requests with update protocol and period. 256 MByte PA, 4 GByte WP, 4 MByte hint space.

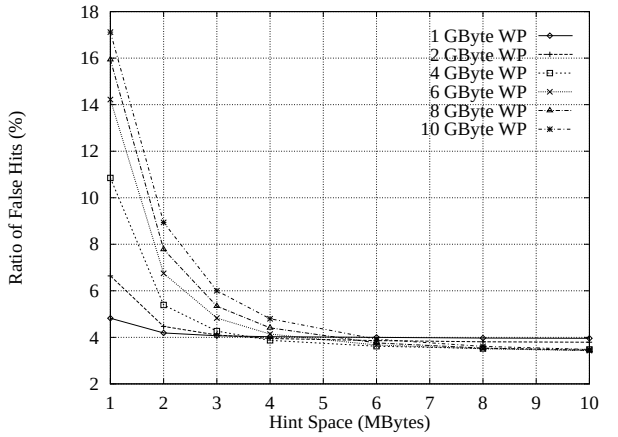

Figure 8: Variation of false hit ratio with hint space and WP cache. 512 MByte PA.

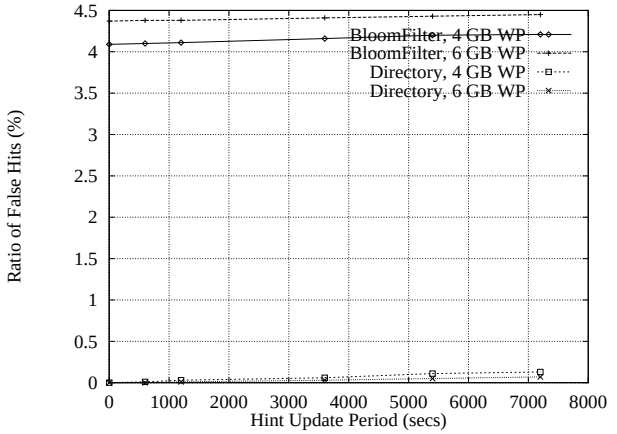

Figure 9: Variation of false hits with update period. 256 MByte PA, 4 MByte hint space.

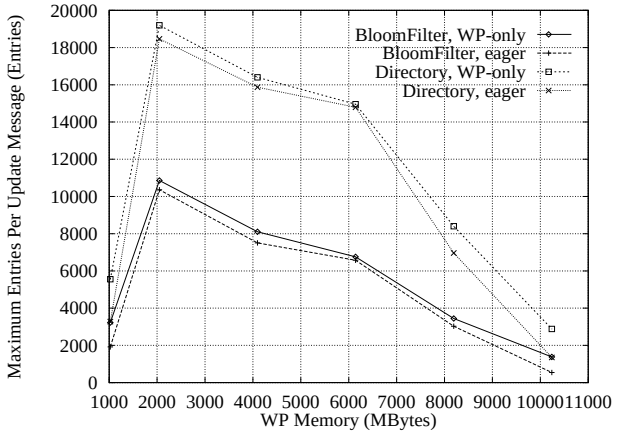

Figure 10: Variation of maximum number of entries per update message. 1 hour period, 256 MByte PA, 4 MByte hint space, 4 entries per object.

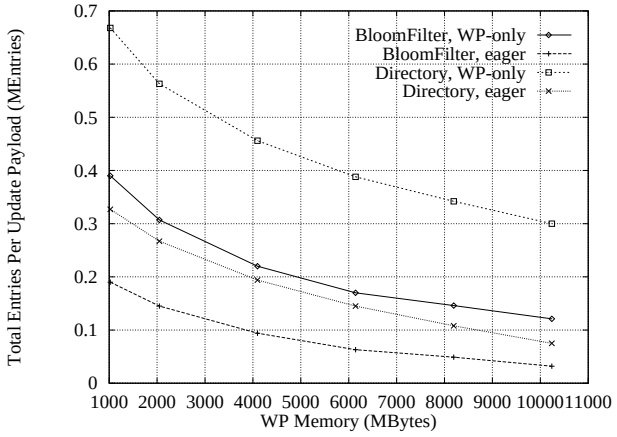

Figure 11: Variation of total number of update entries. 1 hour update period, 256 MByte PA, 4 MByte hint space, 4 entries per object.