RC21611(97267)28SEP1999 Computer Science/Mathematics

IBM Research Report

Hint-based Acceleration of Web Proxy Caches

Daniela Rosu, Arun Iyengar, and Daniel Dias

IBM Research Division
T.J. Watson Research Center
Yorktown Heights, New York

Research Division
Almaden · T.J. Watson · Tokyo · Zurich

Hint-based Acceleration of Web Proxy Caches

Daniela Rosu   Arun Iyengar   Daniel Dias

IBM T.J.Watson Research Center, P.O.Box 704,
Yorktown Heights, NY 10598
{drosu,aruni,dias}@watson.ibm.com

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, which we call 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 of this scheme and the potential impacts of hint-related mechanisms. Our proxy accelerator results in significant performance improvements. A single proxy accelerator node in front of a four-node Web proxy improves throughput by about 100%.

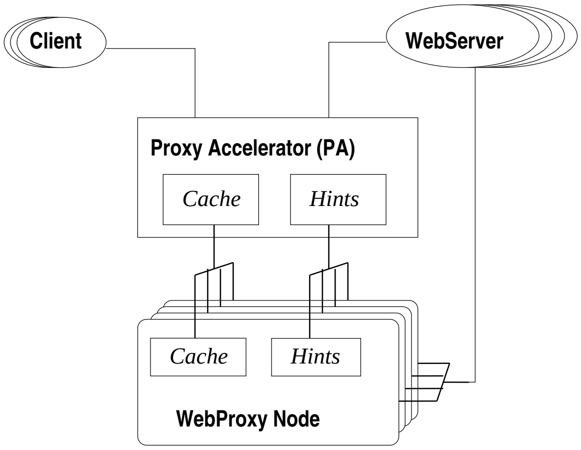
1 Introduction

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.

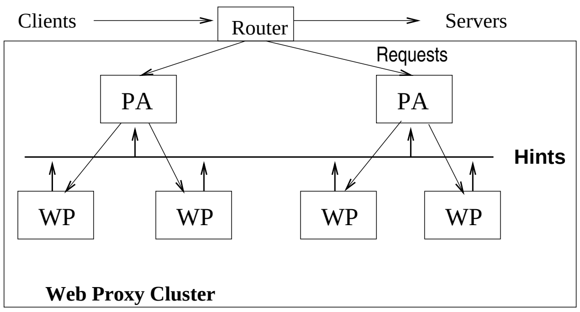
Based on these insights and on our experience with Web server acceleration [11], this paper proposes and evaluates a new approach to improving the throughput of proxy caches. Software running under an embedded operating system optimized for communication takes over some of the Web proxy cache's operations. This system, which we call a *proxy accelerator,* is able to achieve significantly better performance for communications operations than a traditional proxy node [11]. Using information about the Web proxy cache content, the accelerator filters the requests, forwarding the hits to the Web proxy and processing the cache misses itself. Therefore, the Web proxy miss-related overheads are reduced to receiving cacheable objects from the accelerator.

Information about proxy caches that is used in filtering the requests is called *hint information.* The hint representation might trade off accuracy for resource usage. Therefore, the hints 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, which is used to store the hottest objects in the proxy 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.



(a)



(b)

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

Figure 1 illustrates the interactions of the proxy accelerator (PA) with Web proxy nodes, clients, and Web servers. Requests to the proxy site initially go through the accelerator, which reduces load on the Web proxy node by using hints and its own cache. For large proxy clusters, the flow of requests can be distributed across several accelerators, each with hints about all nodes in the cluster. In the new architecture, a traditional proxy node is extended with procedures for hint maintenance and distribution [5, 16], for transfer of TCP/IP connections and URL requests [13], and for processing of transferred requests and pushed objects.

Previous research has considered the use of location hints in the context of cooperative Web proxies [5, 16, 18, 14]. Our system extends this approach to the management of a cluster-based Web proxy. In addition, our work extends the approach to cache redirection proposed by previous research [12, 10, 13] by having 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. The main results of our study are the following:

- 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. An accelerator similar to the Web server accelerator described in [11] can achieve such performance;
- the overall hit ratio is not affected by the hint mechanisms when the amount of hint meta-data in the Web proxy cache is small; and
- the overhead of hint maintenance is about 50% lower when the proxy accelerator registers hint updates based on its knowledge of request distributions.

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 mechanism on several Web proxy performance metrics with a trace-driven simulation. Section 5 discusses related work. Finally, Section 6 summarizes the main results and conclusions.

2 Accelerated Web Proxy Architecture

This paper proposes a new approach to speeding up proxy caches by reducing cache miss overheads. In the new approach, the traditional Web Proxy (WP) configuration is extended with a proxy accelerator (PA). The PA runs under an embedded operating system that is optimized for communication. Such operating systems can communicate via TCP/IP using considerably fewer instructions than if a general-purpose operating system is used. The hardware is typically stock hardware.

It may be difficult or impossible to develop an efficient implementation of a WP on such an embedded operating system. For example, the operating system used for the Web server accelerator described in [11] has limited library and debugging support which makes general software development difficult. In addition, there is no multithreading or multitasking. Pauses due to disk I/O can therefore ruin performance. There is thus a need for a PA that can communicate with a WP running under a conventional operating system.

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 forwards 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. The PA may receive the object from the Web site, reply to the client, and if appropriate, push the object to a proxy node in order to allow the proxy node to cache the object. Alternatively, the PA may forward the client and Web site connections to a WP node, which will receive the object and reply to the client.

In addition to hints, 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 can be populated with objects received directly from remote Web sites or objects pushed by WP nodes.

Figure 1 illustrates the interactions of a PA in a four-node WP cluster. The system may include one or several PAs, depending on the expected load and the performance of WP nodes. When multiple PAs exist in the system, they all include hints on each of the WP nodes, and they push objects and forward connections to all or only a subset of these nodes.

In the remainder of this section, we present several methods and policies that may be used to implement the hint-based redirection and the PA cache.

Hint Representation. Previous research has considered multiple hint representations and consistency protocols. Representations vary from schemes that provide accurate information [18, 14, 7] to schemes that provide information with a certain degree of inaccuracy [5, 16]. Typically, the performance of a hint scheme (i.e., the accuracy) is inversely correlated to its costs (i.e., memory and communication requirements, look-up and update overheads).

In this study, we consider two representations – a hint representation that provides accurate information, referred to as the *exact hint scheme*, and a representation that provides approximate (inaccurate) information, referred to as the *approximate hint scheme*. The *exact* scheme consists of 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. Similar representations have been considered in schemes proposed for collective cache management [7, 18].

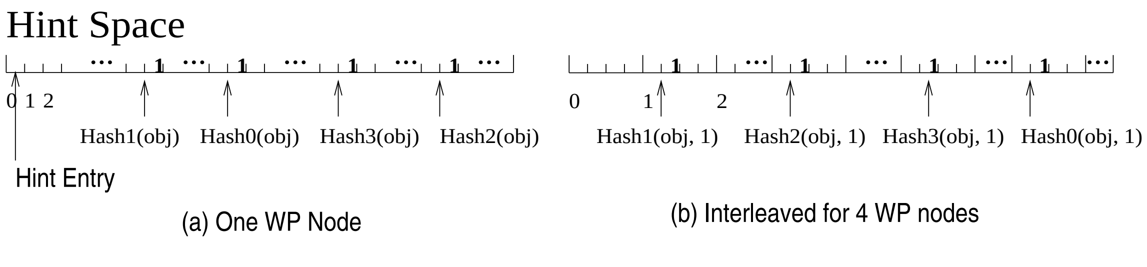


Figure 2: Hint representation based on Bloom Filters.

The *approximate* 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 hint-space entries are set. 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 there is no object whose representation includes it. Therefore, the cache owner (i.e., the WP node) has to maintain an auxiliary data structure that tracks the number of related objects for each hint entry.

When the WP has several nodes, the hint spaces corresponding to each of these nodes can be represented independently or can be interleaved such that the corresponding hint entries are located in consecutive memory locations (see Figure 2 (b)). The two methods have different look-up and update overheads.

Hint Consistency. The hint consistency protocol ensures that the hints at the PA site accurately reflect the content of the WP cache. Protocol choices vary in three dimensions: protocol initiator, rate of update messages, and exchanged information. With respect to the protocol initiator, previously proposed solutions are based on either pull [16] or push [5, 18, 7] paradigms. In a pull protocol, the initiator is the hint owner (i.e., the PA), and in a push protocol, the initiator is the cache owner (i.e., the WP node). While a pull protocol shields the PA from the arrival of unexpected hint updates, a push protocol permits better control of hint accuracy [5].

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]. With respect to the exchanged information, both complete and incremental information may be considered. In the complete information case, the cache owner maintains a copy of the hint space and sends it at protocol-specific intervals of time. In the incremental information case, the cache owner sends only information that describes the additions and removals incurred since the previous update. To the best of our knowledge, the incremental method has been considered in all of the related studies [7, 5, 16]. The complete approach is mentioned in [5] as an alternative for situations in which the incremental update message is larger than the hint space itself.

In our implementation, we extend the traditional hint consistency methods by having the PA actively involved in hint registration. More specifically, when the PA forwards a connection or pushes an object to one of the WP nodes, it can immediately register that the object is part of the corresponding cache. Consequently, a WP node only has to send updates for cache removals, additions of objects received from sources other than the PA, and failures to cache objects sent by the PA.

This method, which we call *eager hint registration*, has two major benefits. First, the method releases significant PA and WP resources by reducing the amount of information transferred and processed on behalf of the hint consistency protocol. Second, the method decouples the likelihood of erroneous hint information from the update period, therefore enabling the system to overcome overload situations by reducing the update frequency.

Proxy Accelerator Cache. The PA cache may include objects received from other Web sites or pushed by the associated WP nodes. Due to the interactions between the PA and the WP cache, the PA cache replacement policy may be different than the WP cache policy. For instance, while efficient WP policies often consider network-related costs such as byte hit ratio or fetching cost [3], the PA policy should emphasize hit ratio. Therefore, policies based on hotness statistics and object sizes are expected to enable better performance than policies like Greedy-Dual-Size [3]. In our implementation, we use an LRU policy with bounded replacement burst. This policy aims to maintain a large number of cached objects by limiting the number of objects that may be removed in order to accommodate an incoming object.

If the limit is reached and the available space is not sufficient, the incoming object is not cached at the PA.

In the following sections, we evaluate the throughput potential of the proposed proxy acceleration method and the impact of several implementation choices.

3 Throughput Improvements

The major benefit of the proposed hint-based acceleration of Web Proxies is throughput improvement. The throughput properties of this scheme are analyzed with an analytic model. The system is modeled as a network of M/G/1 queues. The queue servers represent the PA, the WP nodes and disks, and the WS. The serviced events represent (1) the various stages in the processing of a client request (e.g. PA Hit, PA Miss, WP Hit, WP Disk Access) and (2) the hint and cache management operations (e.g. hint update, object push).

The event arrival rates are determined by 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, 17, 15]. For instance, we consider a WP hit ratio of 40% and PA hit ratios up to 30%.

The overhead associated with each event results from basic overhead, network I/O overheads (e.g. connect, receive message, transfer connection), and operating system overheads (e.g. context switch). In our study, 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 [11] – 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, 1] – 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; (6) 25% of WP hits are pushed to the PA when WS replies are received by the WP; (7) $30\mu sec.$ WP context switch overhead. The costs of the hint-related operations are based on our implementation and simulation-based study (see Section 4).

In the following, we present the main insights from our study in which we vary the number of WP and PA nodes, the PA functionality and cache size, and the WP's and PA's CPU speeds.

Hint-based acceleration improves WP throughput. Figure 3 illustrates that the hint-based acceleration of WPs enables significant performance improvements. The plot presents the expected performances for a traditional WP-cluster (labeled *WP-only*) and for clusters enhanced with one to four PAs. Notice that a single PA with a 4-node WP results in almost the same performance as a traditional 8-node WP, and two PAs enable about 10% performance improvement for a 16-node WP. The plot shows that in order to accommodate large WP clusters, the architecture should allow us to integrate several PAs.

The plot also presents the performance with a PA that on a miss, transfers the client and

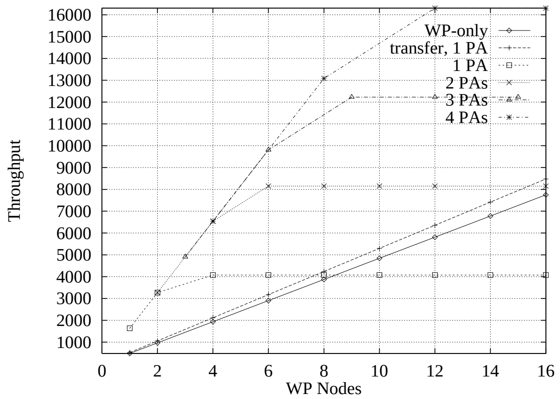


Figure 3: Variation of throughput with number of WP nodes and PAs. 300 MHz no cache PAs, 40% WP hit ratio, 300 MHz WP nodes.

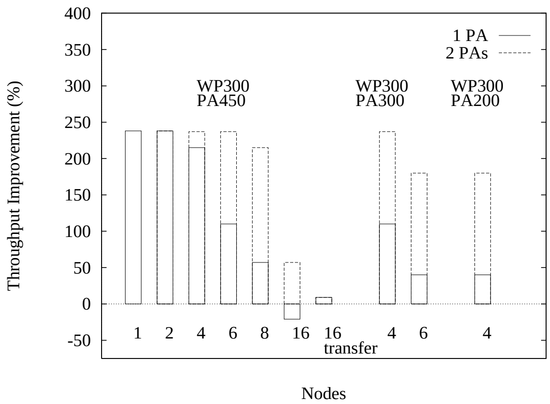


Figure 4: Variation of throughput improvement with number of WP and PA nodes and speeds. No PA cache, 40% WP hit ratio.

Web server connections to be completed by a WP node (see line labeled *transfer*). While such PA functionality accommodates large WP clusters with a single PA, the performance improvement is minimal (with our settings, only about 1% per WP node).

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

Combining content-based and load-balancing routing improves scalability. When the number of requests increases to the extent that the PA is the bottleneck, the achievable performance can be improved if the PA applies the hint-based method to only a fraction of the requests, routing the rest of the requests with a load balancing policy such as round-robin. Figure 5 illustrates the benefit of such an adaptive policy when the load balancing-based routing takes only $30\mu sec.$ per request. The throughout can be improved by more than 10% (e.g. 4-node WP, 10% load balancing-based routing). However, the optimal share of traffic to be routed by the load-balancing policy depends on the WP configuration. For instance, the plot illustrates that for small WP clusters, a 10% share results in better performance than 20% and 30% shares.

PA-level caches improve throughput when WP performance is moderate. Figure 6 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 than in a configuration with no cache.

To summarize, the hint-based Web Proxy acceleration may lead to more than 100% performance improvements when the PA has the same CPU speed as the WP nodes but runs under an embedded operating system highly optimized for network I/O [11]. In the following section, we evaluate the effect of various hint-related mechanisms and policies on the

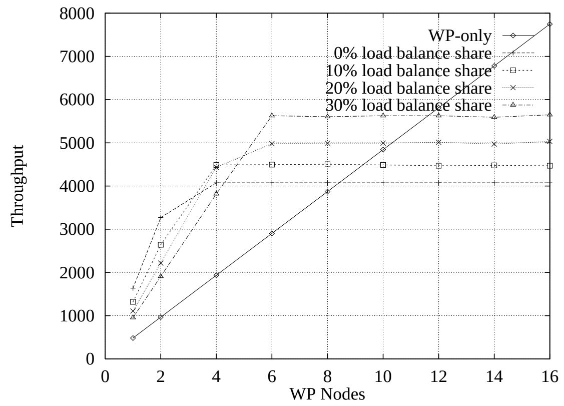


Figure 5: Variation of throughput with share of requests routed by load balancing policy. One 300 MHz no cache PA, 40% WP hit ratio, 300 MHz WP node.

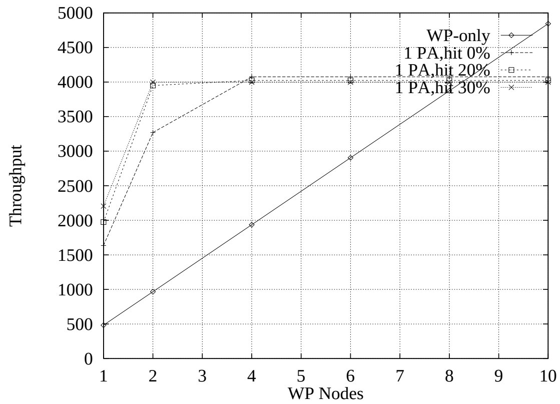


Figure 6: Variation of throughput with PA cache hit ratio. 300 MHz PA, 40% WP hit ratio, 300 MHz WP node.

performance of an accelerated WP using trace-driven simulations.

4 Impact of Hint and Caching Mechanisms

In this section, we evaluate how the hint representation, hint consistency protocol, and the PA cache replacement policy 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.

The study is conducted with a trace-driven simulator. The traces used in our study are a segment of the Home IP Web traces collected at UC Berkeley in 1996 [8]. This segment includes 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.

The main factors considered in this study are the following (see Table 1):

- *PA and WP Memory. Memory space* designates the entire memory and disk space allocated to the cache, the hints, and the related meta-data. We experiment with PA and WP memory sizes comparable to those considered in studies related to cooperative caches [5, 16, 4].
- *Hint Representation.* We experiment with the exact and the approximate hint schemes described in Section 2. For the approximate scheme, we vary the size of the hint space and the number of hash functions (i.e. hint entries used to represent an object). The range of hint space size includes the limits considered in [16] and some of the limits considered in [5] (i.e., eight times the expected number of objects in the cache). The object identifiers used by the approximate hint scheme are composed of the 16-byte MD5 representation of the URL and a 4-byte encoding of the server name. Each hash function calculates hash values from the remainder of the hint space size divided by an integer obtained as a combination of four bytes in the object identifier.

Table 1: Factors and Related Sets of Values.

Factor	**Values**		
System Configuration	stand-alone WP		
	one node WP with PA		
PA Memory	0–644 MBytes		
WP Memory	1–10 GBytes		
Replacement Policy	replacement burst:	10 objects–unlimited	
(LRU)			
Hint Representation	no hints		
(incremental updates)	exact hints		
	approximate hints	– hint space:	0.5–10 MBytes
		– hash functions:	3–5
Hint Consistency Period	w/ and w/o eager registration	– period:	0-2 Hours

- *Hint Consistency Protocol.* We experiment with the incremental, periodic push approach in the context of eager hint registration (see Section 2) and of the traditional update mechanism [5, 16].
- *PA Cache Replacement Policy.* We experiment with an LRU, bounded replacement policy (see Section 2). The bound varies from 10 replacements per new object to no bound.

4.1 Cost Metrics

Memory Requirements. The main memory requirements of the hint mechanism are related to the hint meta-data. For the exact hint scheme, no meta-data is maintained at the WP, and the PA requirements linearly increase with the population of the WP cache. For instance, in our implementation using 68-byte entries, the meta-data is about 9 MBytes for a 1 GByte WP cache and about 70 MBytes for a 10 GByte cache. By contrast, for the approximate hint scheme, meta-data is maintained at both the PA and WP and is independent of the cache size. At the PA, the meta-data is as large as the hint space multiplied by the number of WP nodes. At a WP node, the 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 hint-based acceleration of Web Proxies mainly result from request look-ups and the processing of update messages. In addition, minor computation overheads are incurred at each WP cache update.

Look-up Overhead. For the hash table-based representation of exact hints used in our experiments, the look-up overhead depends on the distribution of objects among the hash table buckets. For approximate hints, the look-up overhead includes the MD5 computation and evaluation of related hash functions. On a 133 MHz PowerPC, the computation of the MD5 representation takes about 22$\mu sec.$ per 56 byte string segment, and the evaluation of a hash function takes about 0.6$\mu sec.$ With an interleaved implementation (see Figure 2 (b)), the look-up overhead is constant at about 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 four hash functions, the overhead is about 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 burst of computation triggered by the receipt of a hint consistency message may affect the short-term PA throughput. The amount of computation depends on the number of update entries in the message, and the approximate scheme result in about half the load of the exact scheme (with 160 bytes per entry). Figure 14 and Figure 14 illustrate this trend for both maximum and average message sizes. These figures also illustrate the benefit of eager registration approach. For both hint schemes, on average, the eager hint registration reduces by half the computation loads.

Communication Requirements. The communication requirements derive from the hint consistency protocol and the PA cache policies. The exact scheme is characterized by larger communication requirements than the approximate scheme both with respect to the average and maximum message size (see Figure 15 and Figure 16). For the approximate scheme, the differences among the requirements of representations with different hint entries per object diminishes as the hint space contention increases and the accuracy diminishes (Figure 15).

4.2 Performance Metrics

Hit Ratio. The main characteristic of the hint mechanism that affects the hit ratio is the *likelihood of false misses.* Given the specifics of the Web interactions, false miss results in undue network loads and response delays.

In general, false misses result from characteristics of hint representation and delays in updating hints. Because of our selection of hint representations, in our experiments, false misses are only the result of late updates. Figure 8 illustrates that with a traditional update mechanism, the larger the period, the smaller the ratio of requests directed to the WP. This is because the likelihood of false misses increases with the period. This figure also illustrates that eager hint registration reduces the number of observed false misses. By decoupling the hit ratio from the frequency of cache updates, eager registration more significantly benefits systems with higher update frequencies, such as those with small WP caches.

Another characteristic of the hint mechanism that affects the hit ratio is the *amount of hint meta-data.* Figure 7 presents the observed hit ratios in several configurations which differ in the amount and the location of meta-data. The hit ratio decreases when the ratio of meta-data in WP memory increases (see the *exact* and *approx* lines). The impact of the meta-data at the PA is not significant because objects removed from the PA cache are pushed to the WP rather than being discarded.

Throughput. Throughput improvement is achieved primarily by reducing the overhead incurred by a WP for processing cache misses. The extent of this overhead reduction depends on how well the PA identifies the WP cache misses. This characteristic is reflected by the *likelihood of false hits* and is affected by the hint representation method and consistency protocol.

Hint Representation. While the exact scheme has no false hits, for the approximate scheme, the likelihood of false hits increases as hint-space contention increases. As illustrated in Figure 9, 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 hash functions. Figure 10 illustrates that for the approximate scheme, the likelihood of false hits depends also on the number of hint entries per object. Namely, when the hint space is small or the number of objects is high, a representation with smaller number of entries per object enables better performance. For instance, while for a 4 Mbyte hint space a representation with four bits per object offers the best performance across the entire range of WP cache sizes, for a 2 MByte hint space, the best performance for large WP cache sizes is offered by the three-bit representation.

Hint Consistency Protocol. The impact of the update period on the false hit ratio is more significant in configurations with frequent WP cache updates (e.g., systems with small WP caches). This is also true for false misses. Figure 11 illustrates this behavior. For instance, the difference between the false hit ratio with hourly updates and with updates every second is 0.07% for a 4 GByte cache and 0.04% for a 6 GByte cache. Comparing the two hint representations, the impact of the update period is almost identical; the differences between one hour and one second update periods for the exact scheme is 0.06% for a 4 GByte cache and 0.03% for a 6 GByte cache.

Response Time. The hit ratio of the PA cache is a measure of how many requests can receive faster replies. Besides memory availability, the PA hit ratio is influenced by the cache replacement policy. Figure 12 illustrates that the bounded replacement policy enables better PA hit ratios without reducing the overall hit ratio. For instance, limiting the replacement burst to 10 objects enables more than 5% larger PA hit ratios than with unlimited burst (see *bound* = *-1*). This is because the average replacement burst is significantly smaller than the maximum (e.g. 3 vs. 2300 in our experiments).

Summary. By trading off accuracy, the approximate hint scheme exhibits lower computation and communication overheads. However, the corresponding hint meta-data at the WP may reduce the overall hit ratio. Look-up overheads are small with respect to the typical overhead of a reply to a request. Eager hint registration reduces update-related overheads and prevents hit ratio reductions caused by update delays. The ratio of false hits is significantly dependent on hint representation and update period. For approximate hints, the ratio of false hits 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, 14, 18]. In these papers, the hints help reduce network traffic by more efficient Web Proxy cache cooperation. For instance, [5, 16] use a Bloom filter-based representation, while [14, 18] experiment with directory-based representations. Extending these approaches, we use hints to improve the throughput of an individual Web proxy. In addition, we propose and evaluate 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 [11], ACEdirector (Alteon) [12], Web Gateway Interceptor (MirrorImage), DynaCache (InfoLibria) [10], 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 (DynaCache, Web Gateway Interceptor), or the front end of a Web server [11]. Furthermore, our approach is similar to the redirection methods used in architectures focused on content-based redirection (ACEdirector and [13]). However, the hint-based redirection proposed in this paper enables dynamic rather than fixed object-to-WP node mappings (ACEdirector). Also, in contrast to the method in [13], redirection decisions use information which is updated upon cache replacements and is independent of client request patterns.

6 Conclusions

Based on the observation that hit 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 an embedded system optimized for communication. This embedded system, called Proxy Accelerator, uses hints of the Web proxy cache content to identify and service cache misses. The Web proxy cache miss overhead is reduced to receiving the object from a Proxy Accelerator over a dedicated connection and caching it. As a consequence, the Web proxy can service a larger number of hits, and the Proxy Accelerator-based system can achieve better throughput than traditional Web proxies [12, 5, 16]. In addition, under moderate load, response time can be improved with a Proxy Accelerator main memory cache [11, 10].

Exploiting the benefits of an embedded operating system, the Proxy Accelerator is similar to the Web Server Accelerator proposed in [11]. In addition, because of reduced load on the Web Proxy node, the Proxy Accelerator enables better performance than related Transparent Proxy Caching and Web Cache Redirection solutions [12, 13].

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 [11] can improve the throughput by about 100% in a Web Proxy with four nodes and more for a proxy with fewer nodes. In addition, the simulation-based study provides the following insights on the implementation of a Proxy Accelerator-based system:

- In the absence of false misses, the amount of hint meta-data on the Web proxy is the primary hint representation characteristic that affects overall performance.

- A low hint update rate does not affect the overall hit ratio significantly provided the accelerator is performing eager hint registration based on its own decisions of request and object distribution. Eager hint registration reduces also the overhead of processing hint updates, therefore reducing the short-term impact of hint management.
- A replacement policy for the accelerator's cache that optimizes the hit ratio, such as the bounded replacement policy, improves the response time with little impact on the network traffic of the overall Web proxy.

In the future, we plan to extend the proposed Web proxy acceleration scheme to accommodate HTTP 1.1 traffic. Towards this end, the PA also has to maintain hints on the pool of client and Web Server connections and coordinate the connection transfer among the proxy nodes. Connection-related hints have shorter lifetimes than object-related hints. In addition, the performance enabled by currently used connection caching policies developed for single-node Web proxy caches may significantly diminish in the context of cluster-based Web proxies. Consequently, further research is required in order to achieve optimal performance in the presence of HTTP 1.1.

References

[1] G. Banga and J. C. Mogul. Scalable kernel performance for Internet servers under realistic loads. *USENIX Symposium on Operating Systems Design and Implementation*, June 1998.

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

[12] A. Networks. New Options in Optimizing Web Cache Deployment. *white paper.*

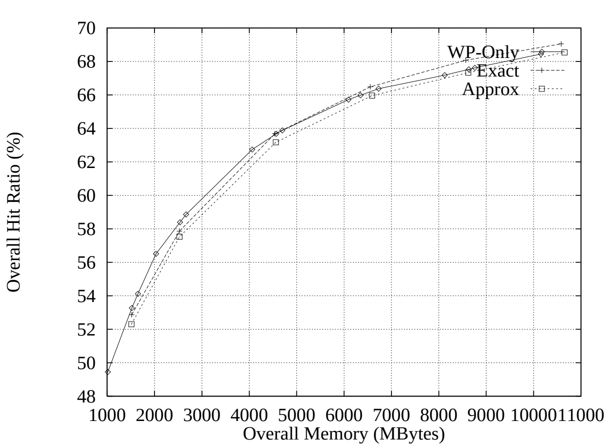


Figure 7: Variation of overall hit ratio with WP configuration. 512 MByte PA cache, approximate hints with 10 MByte hint space.

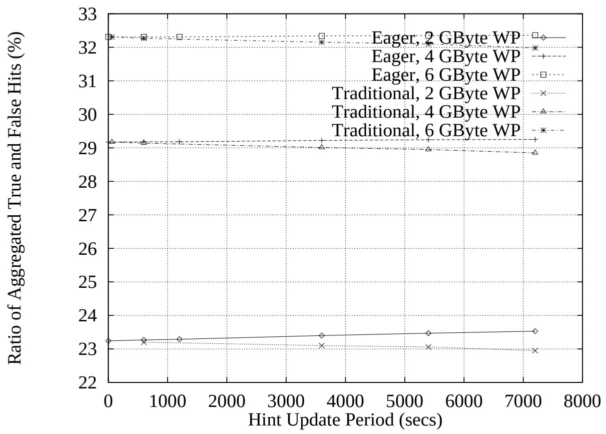


Figure 8: Variation of ratio of requests forwarded to WP with the update period. Approximate hints, 256 MByte PA cache, 4 MByte hint space.

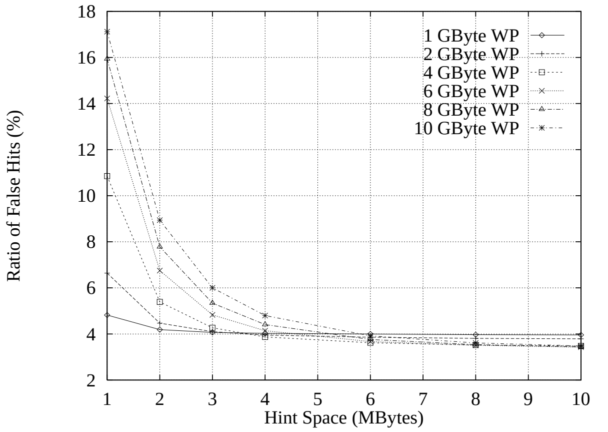


Figure 9: Variation of false hit ratio with hint space and WP cache.

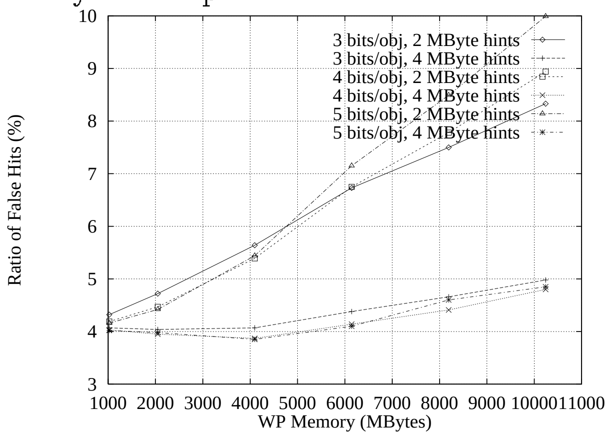


Figure 10: Variation of false hit ratio with hint bits per-object. 512 MByte PA cache.

[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] M. Rabinovich, J. Chase, and S. Gadde. Not All Hits Are Created Equal: Cooperative Proxy Caching Over a Wide-Area Network. *The 3rd International WWW Caching Workshop*, 1998.

[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] A. Rousskov, D. Wessels, and G. Chisholm. The First IRCache Web Cache Bake-off. *http://bakeoff.ircache.net*, Apr. 1999.

[18] R. Tewari, M. Dahlin, H. M. Vin, and J. S. Kay. Beyond Hierarchies: Design Considerations for Distributed Caching on the Internet. *Technical Report CS98-04, Department of Computer Sciences, UT Austin*, 1998.

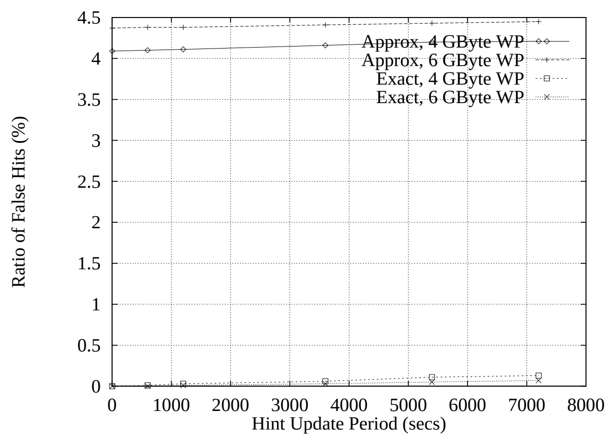


Figure 11: Variation of false hit ratio with period of hint updates. 256 MByte PA cache, 4 MByte hint space.

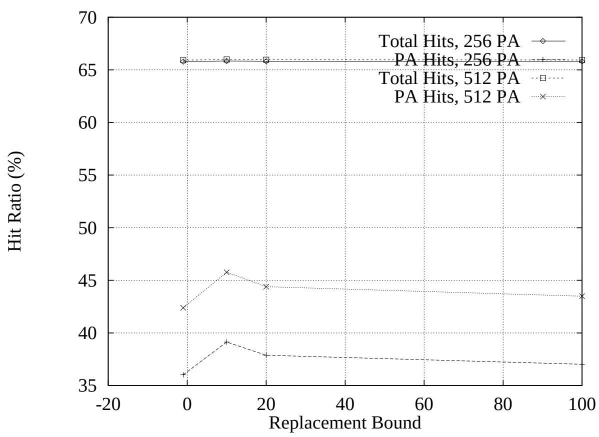


Figure 12: Variation of hit rate with replacement bound (-1, unbounded replacements). Approximate hints, 4 MByte hint space, 6 GByte WP cache.

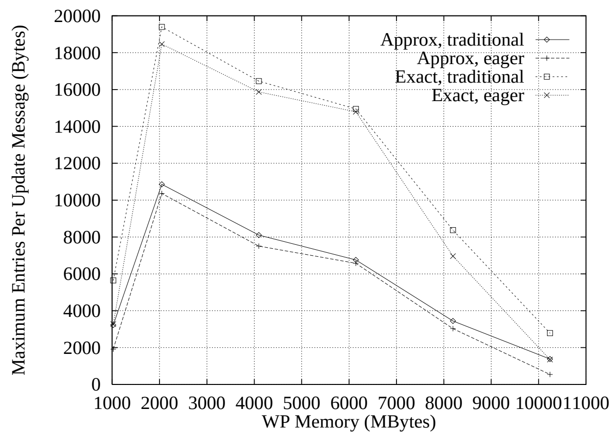


Figure 13: Variation of maximum number of entries per update message. 1 hour updates, 256 MByte PA cache, 4 MByte hint space, 4 bits per object.

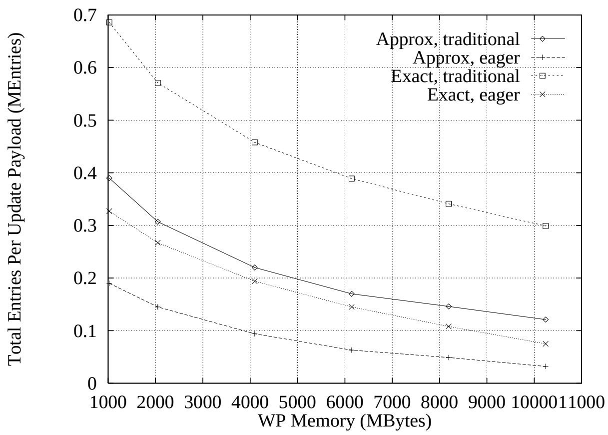


Figure 14: Variation of total number of update entries. 1 hour updates, 256 MByte PA cache, 4 MByte hint space, 4 bits per object.

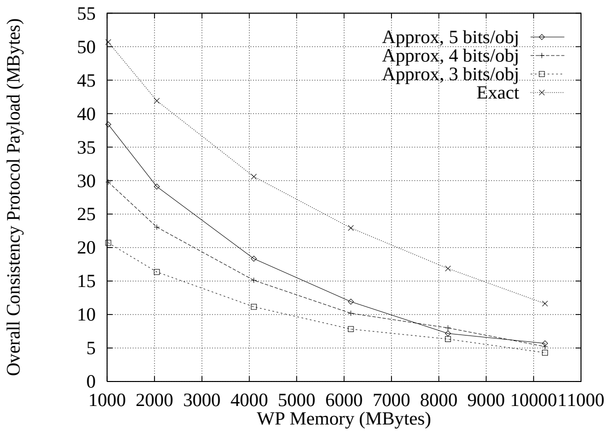


Figure 15: Variation of total hint consistency traffic with hint representation. 512 MByte PA cache, 4 MByte hint space, 1 sec. updates.

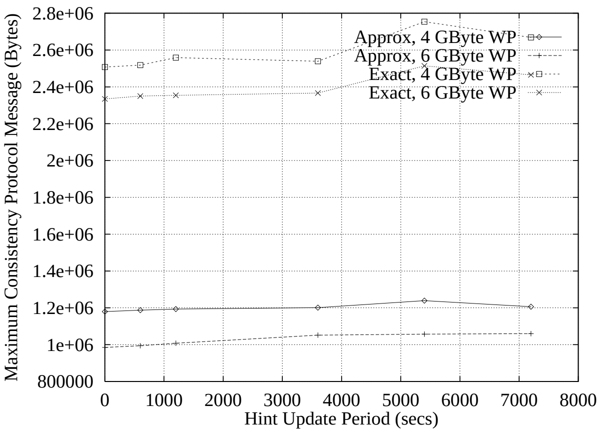


Figure 16: Variation of maximum update message size with update period. 256 MByte PA cache, 4 MByte hint space.