RC 21945 (98662) 2 February 2001 **Computer Science**

IBM Research Report

Web Proxy Acceleration

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

Research Division
Almaden - Austin - Beijing - Haifa - T. J. Watson - Tokyo - Zurich

Web Proxy Acceleration

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 miss ratios at forward proxy are typically at least 40%-50%. This paper proposes and evaluates a new approach for improving the throughput of Web proxy systems by reducing the overhead of handling cache misses. Namely, we propose to front-end a Web proxy with a high performance node that filters the requests, processing the misses and forwarding the hits and the new cacheable content to the proxy. Requests are filtered based on hints of the proxy cache content. This system, called Proxy Accelerator, achieves significantly better communications performance than a traditional proxy system. For instance, an accelerator can be built as an embedded system optimized for communication and HTTP processing, or as a kernel-mode HTTP server. Scalability with the Web proxy cluster size is achieved by using several accelerators. We use analytical models, trace-based simulations, and a real implementation to study the benefits and the implementation tradeoffs of this new approach. Our results show that a single proxy accelerator node in front of a four-node Web proxy can improve the cost-performance ratio by about 40%. Hint-based request filter implementation choices that do not affect the overall hit ratio are available. An implementation of the hint management module integrated in Web proxy software is presented. Experimental evaluation of the implementation demonstrates that the associated overheads are very small.

Keywords: proxy cache, hints, Web performance

1 Introduction

Proxy caches for large Internet Service Providers can receive high request volumes.

Numerous studies show that miss ratios are often at least 40%-50%, even for large proxy caches [7, 9, 12]. Consequently, a reduction of the cache miss overheads can result in significant throughput improvements for proxy caches experiencing high request volumes.

Based on this insight and on our experience with Web server acceleration [15], this paper proposes and evaluates a new approach to improving the throughput of Web proxy systems. Namely, we propose to front-end a Web proxy system with a high performance node that filters the requests, processing the misses and forwarding the hits and new cacheable content to the proxy. The Web proxy's miss-related overheads are reduced to receiving the cacheable objects pushed by the accelerator over permanent connections. Consequently, the proxy can use the released resources to service an increased number of hits. Figure 1 illustrates the interactions of the proxy accelerator with Web proxy nodes, clients, and Web servers.

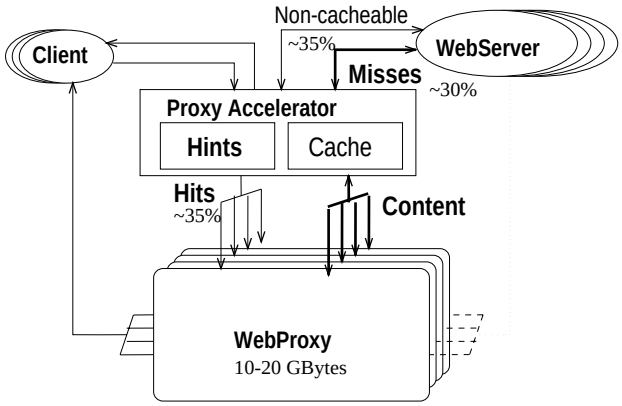

Figure 1: Accelerated Web Proxy Architecture.

The accelerator is an extended HTTP server running under an embedded operating system optimized for communication, like the Web server accelerator described [15], or under a general-purpose operating system in kernel-mode at service-interrupt level, like the Netfinity Web Server Accelerator[16]. The accelerator has a main memory cache, mainly used to store content to be forwarded to the Web proxy.

The accelerator filters the requests based on a summary of the proxy cache content, called *hints*. The hints are maintained based on information provided by proxy nodes and information derived from the accelerator's own decisions. The hint representation and the update mechanism are important factors for the performance gains enabled by the proposed proxy acceleration method. This is due to the related requirements for system and network resources and to the intrinsic tradeoff between resource usage and information accuracy.

Previous research has considered the use of location hints in the context of cooperative Web proxies [8, 19, 22, 10]. Our system applies this approach to the management of a cluster-based Web proxy. In addition, we experiment with new hint maintenance mechanisms, specifically designed for the interaction between a front end and a cluster of homogeneous nodes.

Our work extends the set of solutions to proxy cache redirection [1, 13, 14, 17]. Most

relevant is the comparison with the proposal in [14], which has the front end identifying and processing the requests for non-cacheable content. Extending this functionality, a proxy accelerator processes all of the requests for content not cached at the Web proxy, including the cacheable content not currently in the proxy cache.

Building on top of previous research on cluster-based network services [4, 2], our proxy acceleration method overcomes the intrinsic limitations of content-based routing components by integrating several accelerator nodes in front of a Web proxy cluster.

In this paper, we use analytical models in conjunction with simulations to study the potential for performance improvement of our proxy acceleration scheme. We evaluate the impact of several hint representation and maintenance methods on throughput, hit ratio, and resource usage. In addition, we present and evaluate the performance impact of an actual implementation of the hint-management component in Web proxy application.

Our analytical study reveals that the throughput improvement can be significant. For instance, when the accelerator can achieve about an order of magnitude better throughput than a Web proxy node for communications operations, such as the embedded system considered in [15], a single accelerator node can boost the throughput of a four-node Web proxy by about 100%. 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. In addition, the evaluation of our implementation shows that the overhead incurred with hint management has no significant impact on the client-perceived response times and throughput.

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 presents the implementation of the hint-management module in a Web proxy application and evaluates its performance impact. Section 6 discusses related work, and Section 7 summarizes our results.

2 Accelerated Web Proxy

Typical high-performance Web proxy systems consist of a cluster of nodes running a Web proxy application and a front-end node that distributes the requests to the application nodes. Front end-based solutions to improving the performance of Web proxy systems are restricted mainly to balancing the load based on criteria like number of active connections and CPU utilization. Because of the inherently unpredictable content mixtures in Web proxy caches, typical content-based routing policies that distinguish origin servers and/or object types are less beneficial. However, a content-based scheme distinguishing between cacheable and non-cacheable content has been proved to bring significant performance improvements to Web proxy systems when combined with off-loading the processing of requests for non-cacheable content from proxy nodes to the front end [14].

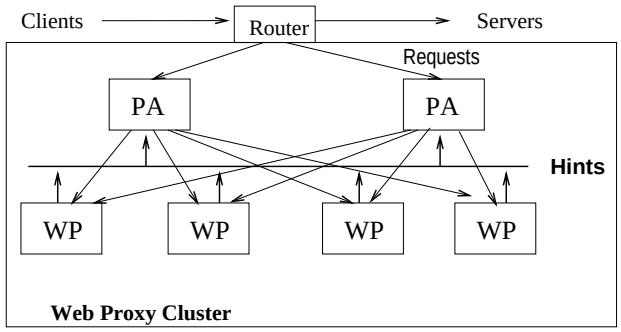

Figure 2: Interactions in a Web Proxy (WP) cluster with several Proxy Accelerators (PA).

In this paper, we propose to further offload proxy node overheads by extending the functionality of the front-end. Namely, the extended front-end, called *proxy accelerator* (PA), is a content-based router that can distinguish between cache hits and misses based on information about proxy cache content. In the case of a cache hit, the PA forwards the request to a Web proxy (WP) node that contains the object either by TCP splicing or handoff [17, 21]. In the case of a cache miss, the PA sends the request to the origin 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 which is selected based on some load balancing criteria.

The enabling element for the new front-end functionality is the PA's information about the content of the WP cache, henceforth called *hints*. Possible solutions for hint representation and update mechanism will be discussed later in this section and evaluated by trace-based simulation in Section 4.

The PA includes a main memory cache for storing content waiting to be pushed to the WP. In addition, the PA cache can store duplicates of objects in the WP cache. Requests satisfied from a PA cache never reach the WP, further reducing the load of the WP.

Figure 1 illustrates the interactions of a PA in a four-node WP cluster. PA and WP nodes are connected by permanent TCP connections for request handoff [17], hint updates, and content push. Notice that WP nodes still interact with Web sites. This occurs when the PA filter is not accurate, or cache objects have to be revalidated.

For scalability, the system may include several PAs (Figure 2) 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. The policy for new content distribution may be as simple as round-robin.

A PA can be built from stock hardware and runs under an embedded operating system optimized for communication. Alternatively, the PA can be built as an extended kernel-mode HTTP server. Either way, a PA can achieve an order of magnitude better throughput than a Web proxy node for communications operations.

A WP node would typically run software extended 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 several hint representations and consistency protocols. Hint representations vary from schemes that provide accurate information [22, 10] to schemes that trade accuracy for reduced costs (e.g. memory and communication requirements, look-up and update overheads) [8, 19].

In this study, we consider two representations – one that provides accurate information and one 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 WP cache. In our implementation, the list is represented as a hash table [10, 22].

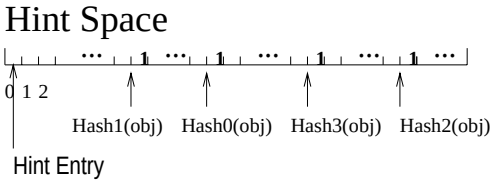

Figure 3: Hint representation based on Bloom Filters.

The approximate-information scheme is based on *Bloom Filters* [5]. The filter is represented by a bitmap, henceforth called *hint space* (see Figure 3), and several hash functions. To register an object in the hint space, the hash functions are applied to the object identifier, and the corresponding bits, which are generically called *hint 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, which is updated whenever a related object is added or removed from the cache. When a collision counter reaches its maximum value, it is no longer updated. The entire hint space is recomputed when the number of overflowed counters reaches a threshold. The PA maintains a Bloom Filter for each of the WP nodes.

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 (henceforth called *WP-only updates*) [8, 19] or by the content-based router (PA) [17]. Both methods have a significant likelihood of wrong request redirection decisions. The first method may result in false misses because of the inherent delay of updates. The second method may result in false hits because information about objects removed from the cache is not propagated to the router. False misses are also possible because entries in the router directory are discarded periodically due to space limitations.

In this paper, we propose and evaluate a method, called *eager hint registration*, which attempts to address the drawbacks of previous methods. Namely, both PA and WP identify and record hints. When the PA pushes an object to a WP node, it "eagerly" registers that the object exists in the corresponding cache. As a side effect, the method enables a reduction of the update traffic sent by a WP node. Namely, a WP node does not send update notifications for objects received from the PA (a small exception, due to the asynchronous

communication between PA and WP, is discussed in Section 5).

With respect to the protocol initiator, previously proposed solutions address only WP-only updates. Updates are pulled by the hint owner (i.e., PA) [19] or pushed by the cache owner (i.e., WP) [8, 22, 10]. With respect to the update rate, previous proposals consider update messages sent at fixed time intervals [19] or after a fixed number of cache updates [8]. The exchanged information may be incremental, addressing only the updates occurred since the previous message [10, 8, 19], or a copy of the entire hint space [8].

Proxy Accelerator Cache. The PA cache may include objects received from Web sites or pushed by the associated WP nodes. In selecting the PA cache replacement policy, we have to consider that all objects in the PA cache also exist in the WP cache. Consequently, the PA cache replacement policy should not be focused on minimizing the network-related costs, as shown appropriate for WP caches. Policies that maximize the hit ratio based on hotness statistics and object sizes are expected to enable better PA performance than policies like Greedy-Dual-Size [6].

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

3 Expected 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 CPUs 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 recent studies on cache performance and Web content models [23, 14, 6, 18].

The overhead associated with each event includes a basic compute 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 [15] – 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 [17] – 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 8 KByte objects.

The following parameters are fixed for this study: (1) 40% non-cacheable content ratio and 25% miss ratio, (2) TCP handoff is used by the PA for client connections upon positive hint look-up; (3) directory-based hints with 1 hour incremental updates; (4) disk I/O is required

for 75% of the WP hits; (5) one disk I/O per object; (6) four network I/Os per object transmission; (7) 25% of WP hits are pushed to the PA, when the PA caches hot objects; (8) $30\mu sec$ WP context switch overhead. Unless otherwise specified, the PA nodes have 450 MHz CPUs, and the WP nodes have 300 MHz CPUs. The costs of the hint-related operations are derived from our implementation and the simulation-based study (see Section 4).

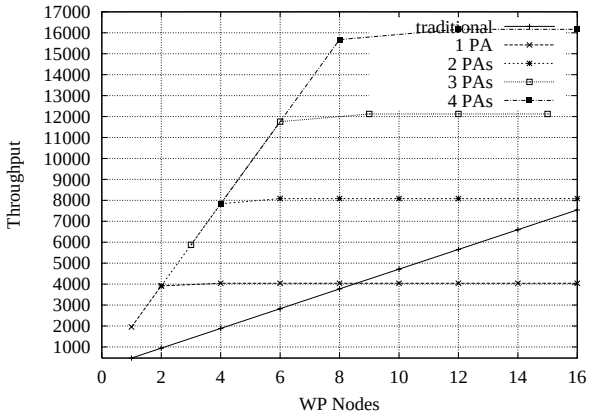

Figure 4: Throughput with number of WP and PA nodes. (no service from PA cache)

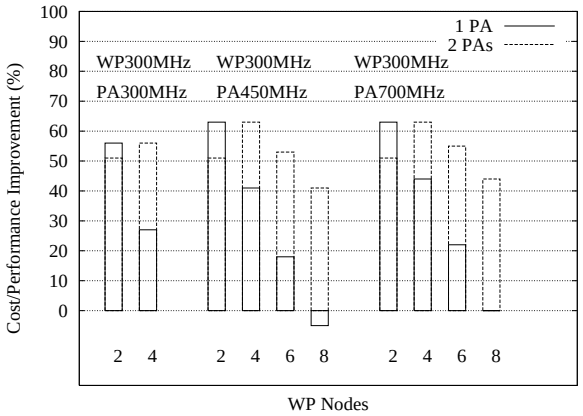

Figure 5: Cost-performance improvement with number and speed of WP and PA nodes. (no service from PA cache)

Hint-based acceleration improves throughput and cost-performance ratio. Figure 4 presents the expected performance for a traditional WP-cluster and for clusters enhanced with up to four PAs when the PAs do not service objects from their cache. The plot illustrates that the hint-based acceleration enables significant WP throughput improvements. For instance, a single PA node can increase the performance of a 2-node WP about three times and that of a 4 node WP about two times. With two PAs, a 4-node WP can achieve higher throughput than a 16-node WP. However, consistent with previous studies on content-based routing [4, 2, 21], the plot illustrates that a PA becomes a bottleneck for large WP clusters.

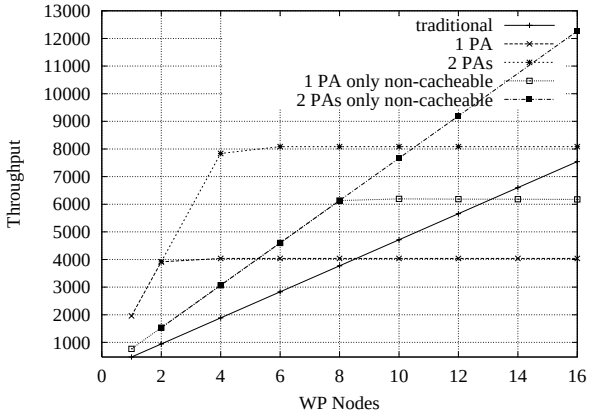

Figure 6: Throughput of hint-based acceleration and non-cacheable redirection. (no service from PA cache)

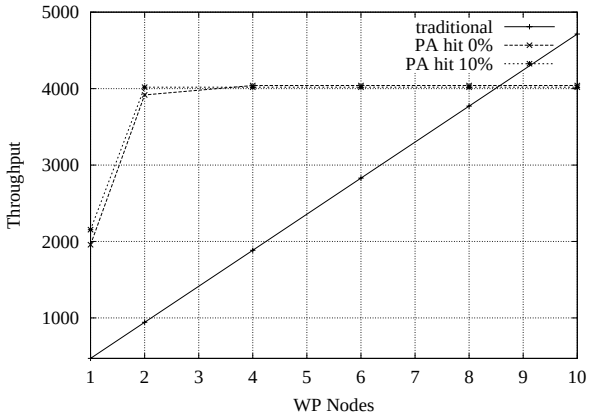

Figure 7: Throughput when PA services objects from its cache.

The cost-performance improvement enabled by the proposed scheme is relevant. For instance, Figure 5 illustrates that if we assume that the cost of a PA is almost the same as for a WP node, the cost-performance improvement achieved with a single PA is ≈60% in a 2-node WP and ≈40% in a 4-node WP. The performance benefits of the scheme increase with the difference between WP and PA's CPU speeds, as illustrated by Figure 5 for the case of 700 MHz PA CPU's. Figure 5 also illustrates that a 2-node accelerator benefits relatively large WP configurations. For instance, the cost-performance improvement with a 2 PAs is lower than with 1 PA for a 2-node WP.

Hint-based acceleration can benefit more than the redirection of non-cacheable objects. Figure 6 illustrates that, for appropriately sized WP clusters, the proposed hint-based acceleration can enable better performance than the previously proposed policy of redirecting only the non-cacheable objects [14]. For a workload with only 25% miss ratio, a 2-node WP cluster can attain more than double throughput levels if the front-end can identify and process the request for cache misses along with the requests for non-cacheable content.

PA-level caches benefit low-performance WPs. Figure 7 illustrates that the effect of

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

a PA cache is marginal in particular when the ratio of non-cacheable content is significant, like in our selection of experiment parameters.

To summarize, the hint-based Web Proxy acceleration may lead to significant performance and cost-performance improvements when the PA nodes can achieve a much higher throughput network I/O than the WP nodes. The new scheme improves upon the previously proposed redirection of non-cacheable content for appropriately sized WP clusters.

4 Impact of Hint Management

In this section, we evaluate how the hint representation and the consistency protocol can 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 depend on various characteristics of the hint management mechanism (see Table 1). For instance, the look-up overhead affects the long-term PA throughput. Also, the false miss decisions of the PA filter cause undue network loads and response delays. Table 2 summarizes some of the differences between the representation methods and the 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 [11]. The 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

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

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 (this restriction makes impossible an accurate evaluation of the resource requirements associated with the Bloom Filter-based scheme). Finally, we experiment with update periods in the range of 1 sec to 2 hours.

Fixed Parameters. We experiment only with "push" updates; in comparison to "pull" updates, "push" updates allow better control of hint accuracy [8] 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 the 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. This policy is expected to perform well based on studies that show that small objects have a large contribution to the hit ratio [18, 3]. For both caches, garbage collection is invoked when a new object has to be accommodated.

4.1 Cost Metrics

Memory Requirements. The main memory requirements of the hint mechanism derive from the hint meta-data and the message buffers used for the hint consistency protocol.

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 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

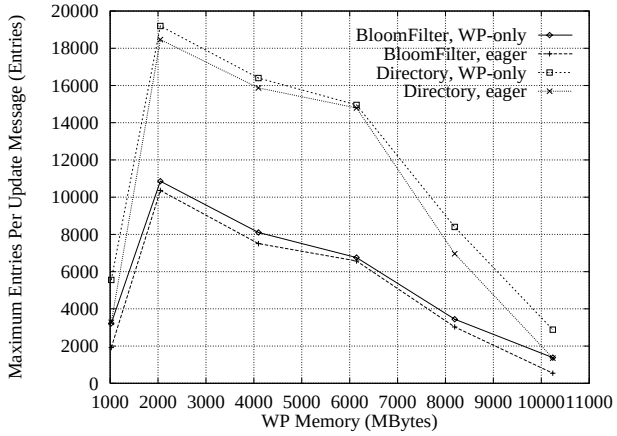

Figure 8: Variation of maximum number of entries per update message. 1 hour period, 256 MByte PA, 4 MByte hint space, 4 entries per object.

space multiplied by the size of the collision counter (e.g. 8 bits).

Hint Consistency Protocol. Message buffer storage space required at the WP site for the hint consistency protocol increases with the rate of WP cache updates and the period of the consistency protocol. For the directory scheme, an update entry has unpredictable length, as it includes the full name of the object. For the Bloom Filter-based scheme, an update corresponds to a hint entry update, represented by a 5-byte data structure; a cache update can result in as many hint entry updates as hash functions in the filter.

Computation Requirements. The computation requirements associated with our acceleration scheme result mainly from request look-up 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, depending mainly on the filter parameter. More specifically, the overhead includes the transformation of the object name into one or more integer values, the computation of the hash functions, the selection, and the test of the corresponding bitmap entries. On a 332 MHz PowerPC, the transformation of the object name into an MD5 representation takes $\approx 9\mu sec$ per 55 byte string segment. For the four simple hash functions used in our Bloom Filter implementation, the overhead of hash function computation and bitmap look-up is $\approx 1\mu sec$. 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 8 and Figure 9 illustrates this trend. These figures also illustrate the benefit of the eager registration approach. For both hint schemes, on average, the eager hint registration reduces by half the computation loads. Similar trends

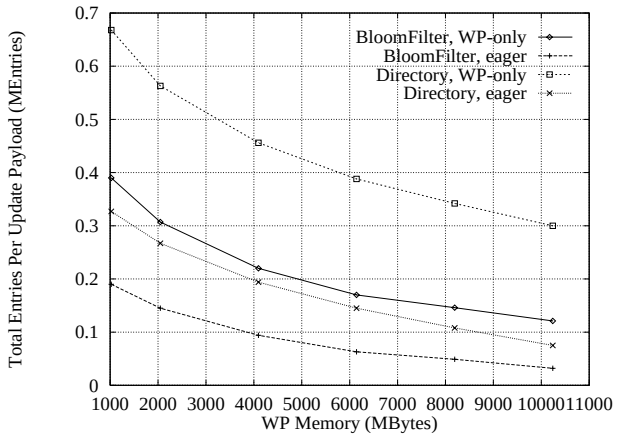

Figure 9: Variation of total number of update entries. 1 hour update period, 256 MByte PA, 4 MByte hint space, 4 entries per object.

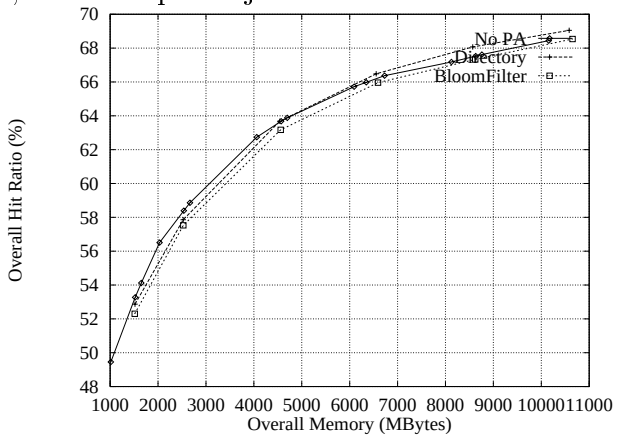

Figure 10: Variation of overall hit ratio with WP configuration. 512 MByte PA, Bloom Filter with 10 MByte hint space.

are observed for the communication requirements of the consistency protocol. Note that the very large maximums observed in Figure 8 are due to a few very large WP cache replacement bursts (≈2300 objects).

4.2 Performance Metrics

Hit Ratio. The characteristic of the hint mechanism that most affects the (observed) hit ratio is the *likelihood 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 11 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 less impact on the Bloom-Filter scheme because of collisions.

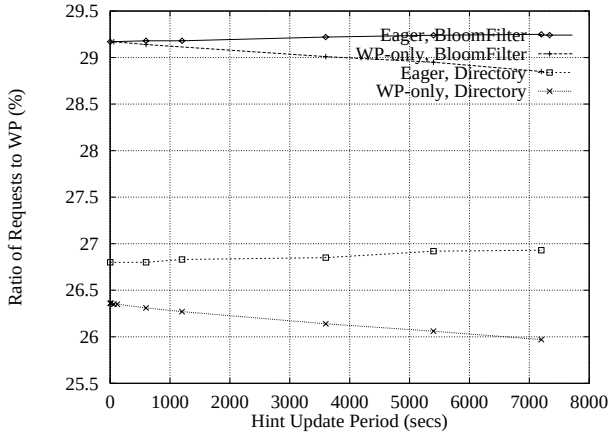

Figure 11: Variation of WP requests with update protocol and period. 256 MByte PA, 4 GByte WP, 4 MByte hint space.

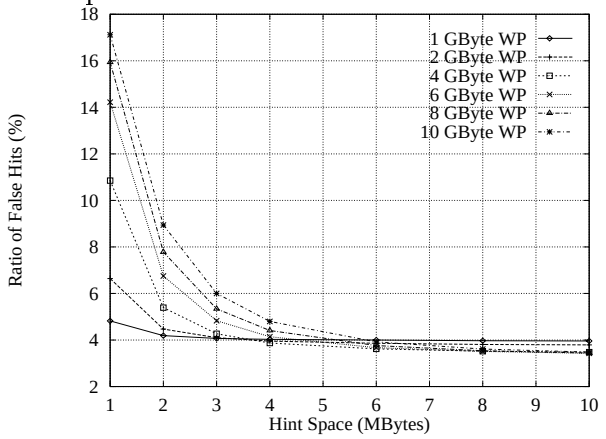

Figure 12: Variation of false hit ratio with hint space and WP cache. 512 MByte PA.

The *amount of hint meta-data at the WP* can have a relevant impact on the overall hit ratio (i.e., hits in both PA and WP caches) when the ratio of the meta-data in the WP memory is large, as for small WP cache sizes (see Figure 10). 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 with the hint-space collision. As illustrated in Figure 12, 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. Figure 13

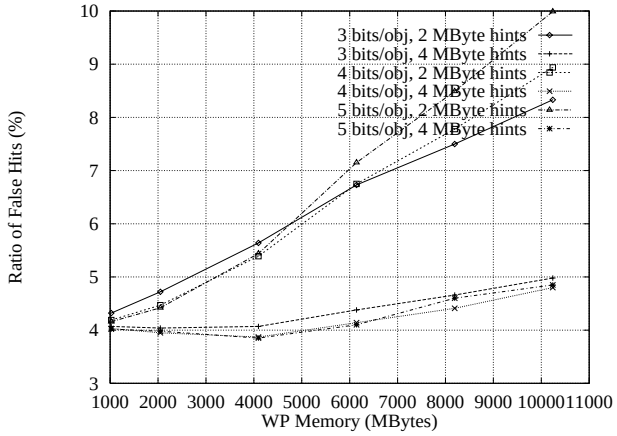

Figure 13: Variation of false hit ratio with hint entries per-object. 512 MByte PA.

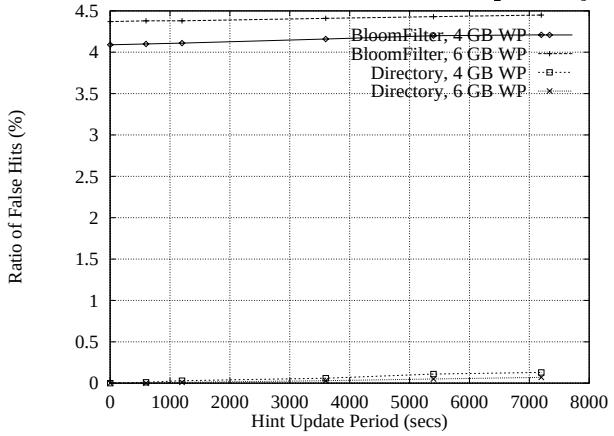

Figure 14: Variation of false hits with update period. 256 MByte PA, 4 MByte hint space.

illustrates the variation observed in our experiments with filters with 3-5 hash functions. While, typically, the ratio of false hits decreases with the number of hint entries per object, when hint space occupancy is high, inversions may occur due to the collisions.

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 14 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.

Request Response Time. Besides the overhead of hint look-up and update, the accelerated WP architecture can affect the average request response time by the reduction of the hit ratio in the WP and PA main-memory caches. Besides the amount of meta-data in the available main memory, the hit ratios depend on the cache replacement policies. Considering the PA cache, Figure 15 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

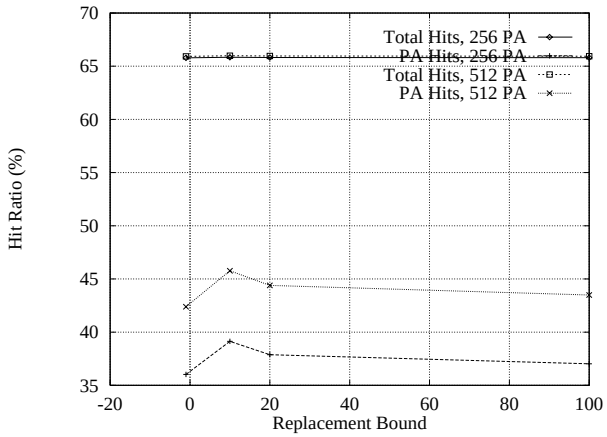

Figure 15: Variation of hit rate with replacement bound. '-1' for unbounded replacements. Bloom Filter, 6 GByte WP, 4 MByte hint space.

maximum (e.g. 3 vs. 2300 in our experiments).

Summary. By trading off accuracy, the Bloom Filter-based scheme exhibits lower computation and communication overheads. However, the meta-data at the WP node may reduce the memory-hit ratio. While, in general, the hint look-up overheads are small with respect to the typical request processing overheads, the overheads are 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 Implementation of Hint Management in Web Proxy Nodes

The implementation of the hint-based accelerated Web proxy infrastructure is built by extending a thread-based Web proxy application and a kernel-mode HTTP server with functionality for hint management and content push. The implementation of the hint mechanism is based on Bloom Filters with eager registration and periodic updates. The implementation makes possible a flexible system configuration. Namely, the Web proxy node can interact with multiple accelerators and the accelerator node can interact with multiple proxy nodes.

Hint Representation. The size of the hint space is customizable at initialization time. The collision counters take up one byte. The Bloom Filter has six hash functions, four applied to the object name and two to the origin server IP address. The object name, $s_0 s_1 ...$, is transformed into an integer value by evaluating the polynomial $\sum_i s_i X^i$ for a fixed prime number. The overhead of name transformation is $\approx 0.95 \mu sec$ for the first 10 bytes, and $\approx 0.7 \mu sec$ for each additional 10 bytes, measured on a 332 MHz PowerPC. Note that,

although less effective than MD5 in avoiding collisions, this solution is characterized by lower computation overheads, which clearly benefit the proxy accelerator performance.

The hash functions are equal to $mod(HintSpace)$ applied to the resulting 4-byte integer and to three permutations of its bytes. Similarly, the $mod(HintSpace)$ function is applied to the 4-byte integer representing the IP address and to one permutation of its bytes. The overhead of computing these hash functions is $\approx 5.5 \mu sec$.

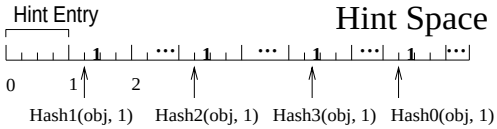

Figure 16: Hint representation as interleaved Bloom filters for 4 WP nodes.

When the WP has several nodes, the corresponding Bloom Filters are represented at the PA by interleaved bitmaps. In this representation, the corresponding bits all of the filters are placed in consecutive memory locations (see Figure 16). In comparison to the original representation in which each Bloom Filter is represented in a separate memory segment, the interleaved representation is characterized by predictable look-up overheads, independent of the cluster size. With the interleaved representation, the look-up overhead for a 4-bit entry is constant at $\approx 1.28 \mu sec$, on a 332 MHz PowerPC. For the original representation, the overhead varies from $\approx 1.19 \mu sec$ for a single node look-up to $\approx 4.39 \mu sec$ for four node look-up.

Hint Update Protocol. The hint management implementation uses the eager hint registration protocol introduced in Section 2. Namely, an accelerator registers hints as it pushes objects towards a proxy node. A proxy node collects the hint updates and periodically sends them to each accelerator such that an accelerator will not receive updates resulting from cache updates for content it has pushed to the WP node. An exception is made when a hint entry is set by a push operation when the previous clear has not reached the accelerator before it pushed the object (and set the entry in its representation). If not appropriately handled, this situation may lead to false misses. To identify this type of situation, proxy updates are labeled, and the pushed content sent by an accelerator carry the label of the most recent update enacted by the proxy.

Web Proxy Software Extension. The extension of the Web proxy software has three components. The first component represents the update interface invoked by the proxy threads handling client requests when objects are added or permanently removed from the cache (cache operations that remove objects only for accommodating their newer versions should not result in hint updates). The calling thread indicates the object name, origin server, and the corresponding cache operation. The resulting hint updates, if any, are recorded in update buffers. When an update buffer gets full, it is released for transmission. Update buffers are organized in an *active set* and a *waiting set*. Updates are recorded in buffers of the active set. When such a buffer gets full, it is replaced with one in the waiting set. The active set includes a buffer for each active accelerator and the local proxy

node. Updates are registered in the active buffer associated with the node that originated the operation. More specifically, updates resulting from a cache-add operation are registered in the buffer of the proxy node when it has received the content from the origin Web server; otherwise, the update is registered in the buffer of the accelerator that pushed the object to the proxy cache. Updates for cache-remove operations are registered in the buffer of the proxy node. This scheme prevents replication of update representations when the proxy interacts with several accelerators and permits the use of multicast if available.

The second component represents the proxy accelerator interface. This component consists of three threads: one thread listens for incoming connections from accelerators, one thread handles accelerator requests for transmission of the entire hint space, and the third thread handles the periodic updates. Accelerators may request a complete copy of the hint space at their initialization. The periodic update thread disseminates to the registered accelerators the update buffers released for transmission. If no buffer is released, the thread transmits the currently active batch update buffer.

The third component of the Web proxy extension is the push handler. This component receives the new content and stores it in the cache. The content is sent in HTTP format as a POST request. While this requires the proxy node to duplicate the header processing that was already done in the accelerator, it decouples the implementations of the proxy cache and the accelerator. In order to hide the latency of the related I/O operations, multiple push connections can be concurrently active between accelerator and proxy.

Experimental Evaluation. In the remainder of this section, we present an experimental evaluation of the overhead of hint management. The experimental evaluation is done with the Web Polygraph proxy performance benchmark [20]. Each component of the benchmark, *polyclt*, *polysrv*, Web Proxy, and proxy accelerators run on separate nodes of a switched Ethernet-based LAN.

The selected traffic model is meant to stress the hint-related component of the Web proxy application. Namely, the request model is characterized by a 75% miss ratio. The content size is fixed to 2 bytes, which results, with the headers, in about 370 bytes per request. Requests are issued by the *polyclt* node in best-effort mode, meaning that a new request is sent as soon as the reply for the previous one is received. Each experiment consists of 100,000 requests.

During experiments, the Web proxy configuration is varied from original (i.e., no hint collection and updates), to hint collection but no updates, and to both hint collection and updates to one and two accelerators. During these experiments, the role of accelerators is reduced to receiving hint updates; requests are received directly by the Web proxy and, therefore, they do not pass through the accelerator. The update period is 5 min.

The Web Polygraph statistics about the response times observed by the client show that the distributions of hit and miss response times are almost identical up to the 98-percentile for all the tested configurations. For the 99-percentile, the miss response times experienced with the configurations with active hint management differ by less than 1% from the performance of the original configuration. For hit response times, the 99-percentile difference is less than 10% of the performance in the original configuration. The throughput observed by the

client is unchanged for the experiment with no active accelerators and ≈1.8% lower for the experiments with active accelerators.

With Web Proxy internal monitoring instrumentation, we collected statistics about the overhead of hint collection with one active proxy accelerator. This overhead includes hint computation and collection into batch update buffers. The statistics show a minimum of $31\mu sec$, a median of $62\mu sec$, and a 95-percentile of $123\mu sec$. The above results demonstrate that hint collection and update protocol have negligible impact on the performance of a Web proxy node.

6 Related Work

Previous research has considered the use of location hints in the context of cooperative Web Proxies [8, 19, 10, 22]. 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 [15], ACEdirector (Alteon) [1], DynaCache (InfoLibria) [13], Content Smart Switch [14], and LARD [17]. 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 [13], or the front end of a Web server [15]. Our proxy accelerator extends the approach to bypassing the Web proxy proposed in [14]; 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, 17]. However, our approach enables dynamic rather than fixed mappings of objects to WP nodes[1]. In contrast to the method in [17], redirection does not cause caching of multiple object replicas and is independent of client request patterns.

7 Conclusions

Based on the observation that miss ratios at proxy caches are often relatively high [7, 12, 9], 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, 8, 19, 17] or systems in which the Web proxy is bypassed only for non-cacheable objects [14]. In addition, under moderate load, response time can be improved with a Proxy

Accelerator main memory cache [15, 13].

Our study shows that a single Proxy Accelerator node with about an order of magnitude better throughput for communication operations than a proxy node [15, 16] can improve the cost-performance ratio of a 4-node Web proxy by ≈35%. 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. The implementation of the accelerated Web proxy node demonstrates that the overhead added by hint management is not significant in comparison to typical proxy node overheads.

References

[1] Alteon Web Systems, ACEdirector Web Switch, in: *http://www.alteonwebsystems.com/products/acedirector2-data.shtml* (1999).

[2] M. Aron, D. Sanders, P. Druschel, W. Zwaenepoel, Scalable Content-aware Request Distribution in Cluster-based Network Servers in: *Proc. Annual Usenix Technical Conference* (2000).

[3] P. Barford, A. Bestavros, A. Bradley, M. Crovella, Changes in Web Client Access Patterns. Characteristics and Caching Implications in: *World Wide Web Journal, Special Issue on Characterization and Performance Evaluation* 2(1-2) (1999).

[4] A. Bestavros, M. Crovella, J. Liu, D. Martin, Distributed Packet Rewriting and its Application to Scalable Server Architectures in: *Proc. 6th International Conference on Network Protocols* (1998).

[5] B. Bloom, Space/time trade-offs in hash coding with allowable errors. in: *Communications of ACM* 13(7) (1970), 422-426.

[6] P. Cao, S. Irani Cost-Aware WWW Proxy Caching Algorithms in: *Proc. USENIX Symposium on Internet Technologies and Systems* (1997).

[7] B. M. Duska, D. Marwood, M. J. Feeley, The Measured Access Characteristics of World-Wide-Web Client Proxy Caches in: *Proc. USENIX Symposium on Internet Technologies and Systems* (1997).

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

[9] A. Feldmann, R. Caceres, F. Douglis, G. Glass, M. Rabinovich, Performance of Web Proxy Caching in Heterogeneous Bandwidth Environments in: *Proc. IEEE INFOCOM'99* (1999).

[10] S. Gadde, J. Chase, M. Rabinovich, A Taste of Crispy Squid in: *Proc. Workshop on Internet Server Performance* (1998).

[11] S. D. Gribble, UC Berkeley Home IP HTTP Traces in: *http://www.acm.org/sigcomm/ITA* (1997).

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

[13] A. Heddaya, DynaCache: Weaving Caching into the Internet in: 3-rd International WWW Caching Workshop (1998).

[14] E. Johnson (ArrowPoint Communications) Increasing the Performance of Transparent Caching with Content-Aware Cache Bypass (1999).

[15] E. Levy, A. Iyengar, J. Song, D. Dias, Design and Performance of a Web Server Accelerator in: *Proc. IEEE INFOCOM'99* (1999).

[16] IBM Netfinity Web Server Accelerator V2.0, in: http://www.pc.ibm.com/us/solutions/netfinity/server_accelerator.html (2000).

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

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

[19] A. Rousskov, D. Wessels, Cache Digests in: *http://squid.nlanr.net/Squid/CacheDigest* (1998).

[20] A. Rousskov, D. Wessels, High Performance Benchmarking with Web Polygraph in: *http://ircache.nlanr.net/Polygraph/doc/papers/paper01.ps.gz* (1999).

[21] J. Song, E. Levy, A. Iyengar, D. Dias, Design Alternatives for Scalable Web Server Accelerators, in: *Proc. IEEE International Symposium on Performance Analysis of Systems and Software* (2000).

[22] R. Tewari, M. Dahlin, H. M. Vin, J. S. Kay, Beyond Hierarchies: Design Considerations for Distributed Caching on the Internet in: *Proc. International Conference on Distributed Computing Systems* (1999).

[23] A. Wolman, G. M. Voelker, N. Sharma, N. Cardwell, A. Karlin, H. M. Levy, On the scale and performance of cooperative Web proxy caching in: *Proc. 17th ACM Symposium on Operating Systems Principles* (1999).