Proceedings of the Fourth International Web Caching Workshop, San Diego, California
April, 1999, pages 159-169

Achieving Load Balance and Effective Caching in Clustered Web Servers

Richard B. Bunt, Derek L. Eager, Gregory M. Oster, and Carey L. Williamson

Department of Computer Science, University of Saskatchewan, Saskatoon, Canada

{bunt,eager,oster,carey}@cs.usask.ca

Abstract

This paper considers policies for distributing requests in clustered Web servers, wherein multiple server machines are configured to function as a single high(er) performance Web server. We evaluate various load distribution policies with respect to both their ability to achieve good load balance (the primary goal) and also to their impact on the effectiveness of per-machine caching. Trace-driven simulation is employed, with workload traces from two heavily-loaded (3-8 million requests per day) commercial Web servers.

Our results show that use of current state information is necessary in achieving good load balance only when the achievable per-request bandwidth is not strongly network or client limited. Use of current state information is not found to be necessary with respect to achieving good cache behaviour. Load distribution based on a static hashed assignment of the URL space is found to yield very similar cache performance to load distribution based on current cache contents. We also find that it is possible to achieve both good cache behaviour and good load balance, but it requires use of policies that take both objectives into consideration and that make use of information concerning current server loads.

1 Introduction

Continued growth in Web usage is causing a number of performance problems, including slow response times, network congestion, and denial of service. There are a number of approaches to addressing these problems. One approach is based on caching copies of Web objects closer to the requesting clients. This may be controlled by servers that "push" popular Web objects out to other cooperating servers [4, 13], or may be triggered by individual client requests passing through client and proxy caches [5, 7, 20, 21, 22]. Another approach is to use prefetching to reduce response times, by hiding server and network latency [6, 18].

A complementary approach is to make the Web server more powerful through the use of a clustered architecture, in which multiple machines function cohesively as a single Web server (e.g., [1, 3, 8, 9, 10, 11, 12, 14, 15, 19, 24]). Such an architecture offers multiple benefits: first, more machines means more capacity to handle requests; second, reliability and availability are improved since the server can continue to operate should some machines go down; third, a consistent external interface can be preserved; and fourth, the design should be more easily scalable. If incoming requests can be distributed among the server nodes in a reasonable fashion, then the overall performance (and scalability) is improved. Distribution of requests (i.e., load distribution), however, has further implications. One important consideration is the effect of load distribution on cache performance within the cluster, since poor cache performance could reduce benefits of load balance.

The goal of this work is to investigate fundamental performance issues within the server cluster; in particular, the benefits of using current state information (both cache contents and server loads) in load distribution. In this paper, we evaluate the impact of load distribution policies on both the load balance that is achieved across the machines in the cluster, and caching performance within the cluster. We consider configurations in which each server machine has a main-memory Web object cache, and load distribution is done by the cluster itself, rather than by individual clients [17, 23]. Trace-driven simulation, with traces from two heavily-loaded (3-8 million requests per day) commercial Web sites, is used to conduct our experiments.

Our main conclusions are as follows:

- Use of current server state information is necessary for good load balance only if the achievable per-request bandwidth is not strongly network or client limited.
- Use of current server state information is not necessary for good cache behaviour. Load distribution based on a static hashed assignment of the URL space is nearly as effective as load distribution based on current cache contents.
- Load distribution based only on load balance considerations can negatively impact the performance achieved with per-machine Web object caches. Similarly, load distribution based only on cache affinity can negatively impact load balance. Achieving both good cache performance and good load balance is possible, but it requires the use of policies that take both objectives into consideration, and that make use of information concerning current server loads.

This work is similar in nature to several previous studies of clustered Web servers (e.g., [1, 8, 9, 10, 12, 15, 19, 24]), with two important differences. First, our work focusses on the performance impacts of the amounts and types of information employed in load distribution, using a methodical exploration of the design space. Second, we focus on load balance and cache performance as primary performance metrics, rather than server or connection throughput. Further discussion of closely related work [12, 19] is deferred until Section 5.

The remainder of the paper is organized as follows. Section 2 describes the load distribution policies that we consider, our trace-driven simulation methodology, and the empirical workload traces that we employ. We first focus solely on load balancing performance; these results are presented in Section 3. In Section 4 we consider the impact of load distribution on caching performance, and give results for load distribution policies that integrate considerations of both caching and load balancing. Section 5 discusses the relationship of this work to other work in this area, and Section 6 concludes the paper.

2 Methodology

We consider load distribution policies that differ in the amounts and types of information employed. In particular, we consider the use of information concerning load balance, and information concerning the contents of the per-machine Web object caches. Three choices are considered with respect to the use of information concerning load:

- Random distribution (RANDOM): no information is utilized; in the absence of cache considerations, and assuming homogeneous server machines, load distribution is random (specifically, by Bernoulli trial, with equal probabilities of going to each server)
- Round-Robin distribution (ROUND-ROBIN): only information on past routing decisions is utilized; in the absence of cache considerations, load distribution is round-robin
- Load-based distribution (LOAD): information on the current load at each server is utilized; in the absence of cache considerations, the most lightly loaded server is selected.

Three choices are also considered with respect to use of information regarding cache contents:

- no information is utilized; load distribution is determined solely according to load balance considerations
- information on the "cache affinities" of the incoming requests is utilized, but these cache affinities are estimated based only on a static hashed assignment of the URL space among the servers
- information on the cache affinities of the incoming requests is utilized; these cache affinities are determined using knowledge of the current contents of each server cache.

In Section 3, policies that attempt only to balance load, with no consideration of cache affinities, are considered. Section 4 considers policies that utilize cache affinity in addition to information on current loads.

Trace-driven simulation is used to evaluate the performance that is achieved with each of the load distribution policies that we consider. We assume that the servers within the simulated cluster are homogeneous. A simple model of server operation is employed, in which the bandwidth (e.g., in Kilobytes (KB) per second) achieved in servicing a request, as determined both by server capabilities and by possible network and/or client-side limitations, is characterized by two parameters. These parameters are the maximum bandwidth that can be delivered to an individual request (B_{indiv}), and the maximum aggregate bandwidth (B_{total}) that a single server can deliver to all the requests that it is attempting to serve concurrently. As motivated from a formula used in simple queueing models of computer systems [16], we model the achieved per-request bandwidth (e.g. in KB/sec) at a server that is concurrently servicing N requests as:

$$\frac{1}{\frac{1}{B_{indiv}} + \frac{N-1}{B_{total}}}$$

yielding an achieved aggregate bandwidth over all of the requests in service at that server of

$$\frac{N}{\frac{1}{B_{indiv}} + \frac{N-1}{B_{total}}}$$

Note that with this model, the service time for a request is a direct function of the request size, and the current service rate. The service rate, in turn, is a dynamic function of the number of simultaneously active connections, as well as the per-server and per-request bandwidth constraints defined above. These constraints reflect the impacts of client-side network bandwidth limitations, connection setup costs, TCP slow-start effects, and the overall contention for server resources.

The workload for our simulations was drawn from empirical traces, obtained from access logs at two busy commercial Internet Web sites. Both sites employed a clustered Web server architecture. The information extracted from the access logs for each request includes a timestamp, an identifier for the host that generated the request, the object requested, the type of object requested, and the object size in bytes.

The general characteristics of our traces are given in Table 1. Trace 1 and Trace 2 were both obtained from the same site. Although there are some significant differences between them, the traces were found to conform surprisingly well to the workload characteristics described in [2]. Thus many of the observations about caching performance reported in [2] are likely to apply with these data sets as well, although the request rates of 3-8 million requests per day are significantly higher. In the interests of space, we focus on Trace 1, the most recent trace, in the remainder of the paper. Results for Trace 2 and Trace 3 are qualitatively similar. Except for some statistics that we present for the entire 7 days of the trace, our simulations use the first 24 hours of trace data for a "warmup period", before measurements are taken.

Figure 1 shows the overall traffic profile for the trace. The workload shows distinct daily and weekly patterns: busiest during the daytime hours, less busy during the evening and nighttime hours, and least busy on weekends. The peak request rate observed in this trace, over one minute observation intervals, was 21,351 requests per minute; the average was 7,912 requests per minute (131.9 req/sec). The peak byte request rate observed, over one minute observation intervals, was 1,496 KB/sec, with an average of 530 KB/sec.

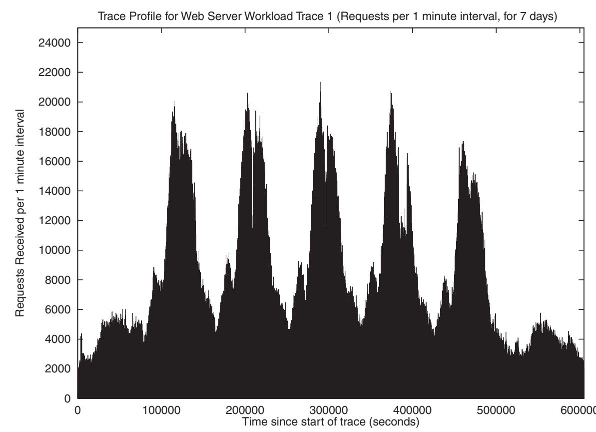


Figure 1: *Traffic Profile for the Web Server Workload in Trace 1 (Requests per minute, full trace)*

3 Load Balancing

In this section we consider only the objective of balancing the load across servers (with no consideration of the resulting caching performance). Section 3.1 describes the metrics employed. These metrics are used in comparisons of alternative load distribution policies in Section 3.2. Section 3.3 evaluates the impact of varying numbers of servers on our principal results. The impact on cache performance is considered in Section 4.

3.1 Metrics

Our comparisons of the load balancing performance that is achieved with various load distribution poli-

Item	Trace 1	Trace 2	Trace 3
Duration	7 days	14 days	7 days
Start Date	June 14, 1998	August 10, 1997	October 17, 1996
Total Requests	56,458,479	42,742,552	38,467,771
Average Requests/Day	8,065,497	3,053,039	5,495,396
Average Object Size Requested (bytes)	5,716	9,350	8,831
Total MBytes Transferred	307,785	381,123	323,963
Average MBytes/Day	43,969	27,223	46,280

Table 1: *Summary of Trace Characteristics (Raw Data)*

cies are based on a new measure that we call the "Load Balance Metric" (LBM). We also give some results for average "inflation factors", for the purpose of establishing the correlation between the LBM and end-user performance. The inflation factor for a request is defined as the factor by which the total service time of the request is increased beyond what it would be were there no contention for server resources; i.e., if the request was serviced at the maximum per-request bandwidth.

In our definition of the LBM, we consider two variants: "LBM Requests" and "LBM Bandwidth". The former is concerned with server load imbalances (as measured by differences in the numbers of requests being served concurrently), while the latter is concerned with differences in the achieved per-request bandwidth (as caused by differing numbers of requests contending for server resources). With either variant, a peak-to-mean ratio of server loads is calculated periodically in the simulation (with the results presented here, after every 10 seconds of simulated time), based on the current server states as observed at that sampling point. The LBM is then formed by taking the weighted average of these peak-to-mean ratios.

Defining *load* as the number of requests being served concurrently, for LBM Requests, and as the inverse of the current per-request bandwidth (equal to the maximum per-request bandwidth if the server is idle), for LBM Bandwidth, the LBM is defined more formally as follows:

- $load_{i,j}$ – *load* of server i (of n servers) at the j^{th} sampling point (of m such points)
- $peak\_load_j$ – highest *load* on any server at the j^{th} sampling point
- peak-to-mean ratio:

$$\frac{peak\_load_j}{(\sum_{i=1}^{n} load_{i,j})/n}$$

- LBM (weighted average of the instantaneous peak-to-mean ratios):

$$\frac{\sum_{j=1}^{m}\left(\frac{peak\_load_j}{(\sum_{i=1}^{n} load_{i,j})/n} \times \frac{\sum_{i=1}^{n} load_{i,j}}{n}\right)}{\sum_{j=1}^{m}\sum_{i=1}^{n} load_{i,j}/n}$$

which is simply:

$$\frac{\sum_{j=1}^{m} peak\_load_j}{(\sum_{j=1}^{m}\sum_{i=1}^{n} load_{i,j})/n}$$

Note that the value of the LBM can range from 1 to at most n, the number of servers. Small values of the LBM indicate better load balancing performance (smaller peak-to-mean load ratios) than large values[1]. LBM Bandwidth is generally a preferable metric to LBM Requests, since it reflects variations in end-user performance more directly (i.e., it focusses on service rate received by a request, rather than on number of requests in service). Note that when load is measured simply by the number of requests, for example, there (incorrectly) appears to be a load imbalance when one request is being served at one server, and the other server is idle. With either variant of the LBM, however, the use of a weighted average ensures that emphasis is given to those sampling points at which server loads are the heaviest, and thus where load balance is the most important.

3.2 Performance Comparison

Figure 2 presents results that illustrate the impact of the use or non-use of current state information in making load balancing decisions, and the dependence of this impact on the extent to which the

[1] A direct physical interpretation of the LBM Requests metric is as follows: On a two-server system, an LBM of 1.0 indicates perfect load balance. A load imbalance where one server queue holds 60% of active requests and the other server has 40% yields an LBM of 1.2 ($60/((60+40)/2) = 1.2$). Similarly, a 70%/30% division of the requests among the servers yields an LBM of 1.4 ($70/((70+30)/2) = 1.4$).

per-request bandwidth is constrained, for a 2-server system. The x-axis of each plot is the maximum per-server bandwidth in KB/sec, and the y-axis is the maximum per-request bandwidth expressed as a percentage of the per-server bandwidth. The z-axis shows the performance surface obtained from our simulations, as a function of these two server parameters. Figure 2(a), for example, shows the relative improvement in the load balancing performance of the Load distribution policy as compared to the Random distribution policy, while Figure 2(b) compares the Load distribution policy to the Round-Robin distribution policy.

A number of observations are evident from these plots. First, relative LBM values are well correlated with relative end-user performance as measured by average service time inflation factors (Figures 2(c) and 2(d)). Second, the Load distribution policy never does worse (on average) than the Random or Round-Robin distribution policies. The relative improvement in the LBM value can be as high as 30%, depending on the server configuration. Third, for an under-resourced server (i.e., low values of the per-server bandwidth), all server nodes have large backlogs, and there is little improvement over a random dispatch policy. Fourth, for an over-resourced server (i.e., high values of the per-server bandwidth), all server nodes have short request queues, and there is little improvement over a random dispatch policy. Fifth, when the per-request bandwidth constraint is small (e.g., slow modem users), there is still little advantage over a random dispatch policy, since all servers will have large backlogs, even when the per-server bandwidths are high. Finally, there is a region in the middle of Figures 2(a) and 2(b) where a load-based request distribution policy can provide significant (15-25%) improvement in the LBM. Similar observations apply for inflation factors (Figures 2(c) and 2(d)).

3.3 Scalability

We study the scalability of our load balancing results by varying the number of nodes in the clustered server architecture. On each configuration, we evaluate two request-based distribution policies, Load and Round-Robin, using the LBM Bandwidth and LBM Requests measures of load balancing performance.

Results for both the Load policy and the Round-Robin policy are shown in Tables 2 and 3. Table 2 shows results for a fixed *per-server* bandwidth (700 KB/sec); the aggregate server bandwidth thus increases with the number of server nodes. Table 3 shows results for a fixed *aggregate* server bandwidth (1,400 KB/sec). As expected, the values of the LBM Request metric increase with the number of server nodes, since it is more difficult to achieve perfect load balancing in larger systems. This observation applies to both the fixed per-server and fixed aggregate bandwidth scenarios. The LBM Bandwidth metric, however, decreases for large numbers of server nodes when the per-server bandwidth is fixed. This is explained by the fact that queue sizes are very small in this case and thus all requests receive service at roughly the same rate, even though some server nodes may be idle while others have one or two requests.

The results in Table 2 for the Round-Robin policy indicate that it does almost as well as the more complex Load policy on larger systems, when the bandwidth per server is fixed. This observation is somewhat uninteresting, however, since it represents a very lightly loaded system in which queue sizes are very small and load balancing is not an issue. When the aggregate server bandwidth is fixed, however, Round-Robin does not do as well as Load on larger systems (Table 3). The reason for this behaviour is the diminishing average request queue size per server; using load information can exploit small differences in queue sizes amongst servers, whereas Round-Robin cannot. The relative importance of using information on current server loads for load balancing purposes increases with the number of servers.

4 Cache Performance Considerations

We assume that each server in the cluster has a local in-memory cache into which referenced Web objects are copied. The cache is managed according to a frequency-based replacement policy, as suggested in [2]. Dynamic files (e.g., as produced as output from CGI scripts) are not cached. Since different requests for the same object may go to different servers, the same object may be present simultaneously in more than one cache. Cache misses may occur because of first-time references, references to dynamic content, limited cache capacity, and invalidation due to object modification.

If cache contents are ignored in load distribution, copies of popular Web objects will find their way into many caches, perhaps forcing out other popular objects. Arguably, the total cache space is used more effectively when fewer replicas exist at once (and they exist in the right place). Further, each replica must initially be created through a separate

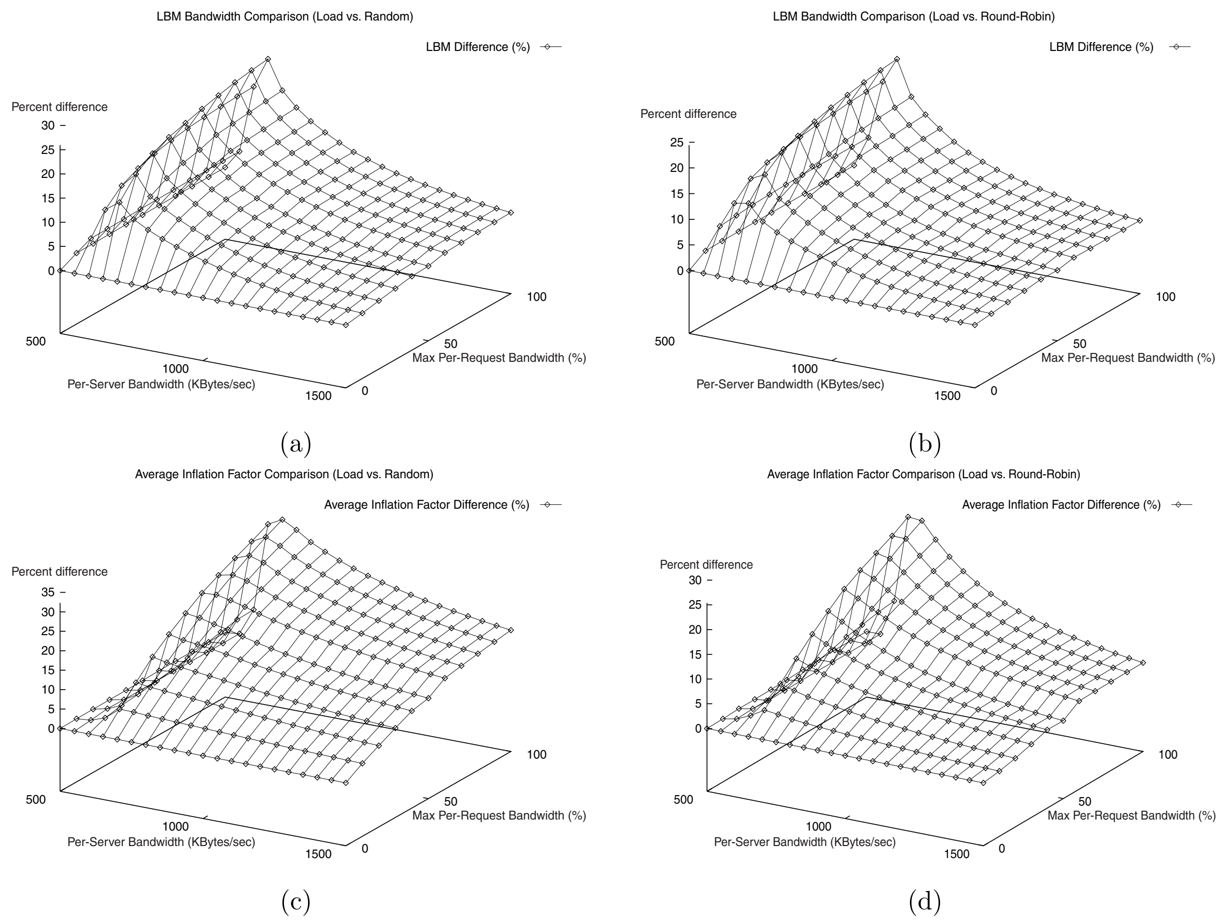


Figure 2: *Load Balancing Performance of Load vs. Random vs. Round-Robin (2 servers): (a) LBM Bandwidth (Load vs. Random), (b) LBM Bandwidth (Load vs. Round-Robin), (c) Average Inflation Factor (Load vs. Random), (d) Average Inflation Factor (Load vs. Round-Robin)*

cache miss. These considerations motivate load distribution by cache affinity. A key issue, however, is whether cache performance and load balance are fundamentally at odds with each other.

In this section we first study the impact on cache performance of ignoring cache affinity in load distribution. We then consider the performance (with respect to both cache hit ratios and load balance) of load distribution policies that take cache affinity into account.

4.1 Impact of Load Distribution on Cache Hit Ratios

Ignoring cache contents in the load distribution policy can have a considerable impact on cache performance. This can be seen in a comparison of cache hit ratios with varying numbers of servers (Figure 3), with each server allocated an equal share of the aggregate cache space, with the Load distribution policy.

Each of the curves in Figure 3 shows the effect on the hit ratio (for objects in (a), and for bytes in (b)) of increasing the aggregate and per-server cache sizes for a particular number of servers (1, 2, 4, or 8) in the cluster. Configurations with a fixed aggregate cache size (but varying numbers of servers and thus varying per-server cache sizes) can be compared by examining points with the same x-coordinate, but on different curves. Configurations with a fixed per-server cache size (but varying numbers of servers and thus varying aggregate cache sizes) can be compared by matching points on the curve for one server to those with twice the aggregate cache space on the curve for two servers, four times the aggregate cache space on the curve for four servers, and eight times the aggregate cache space on the curve for eight servers. (Note that the x-axis is on a log scale.)

Number of Servers	Load LBM		Round-Robin LBM	
	Bandwidth	Requests	Bandwidth	Requests
2	1.059	1.142	1.248	1.406
4	1.090	2.048	1.176	2.302
8	1.064	3.856	1.085	3.980
16	1.030	7.247	1.033	7.287
32	1.013	14.007	1.013	14.020

Table 2: *Scalability of Load Balancing Performance for Load and Round-Robin Policies (per-server bandwidth of 700 KB/sec, per-request bandwidth constraint 50% (350 KB/sec))*

Number of Servers	Load LBM		Round-Robin LBM	
	Bandwidth	Requests	Bandwidth	Requests
2	1.059	1.142	1.248	1.406
4	1.177	1.688	1.706	2.217
8	1.294	2.200	2.236	3.087
16	1.406	2.608	2.738	3.897
32	1.508	2.881	3.116	4.705

Table 3: *Scalability of Load Balancing Performance for Load and Round-Robin Policies (fixed aggregate server bandwidth of 1400 KB/sec, per-request bandwidth constraint of 50% of the per-server bandwidth for the 2 server case, and 100% of the per-server bandwidth for the remaining cases)*

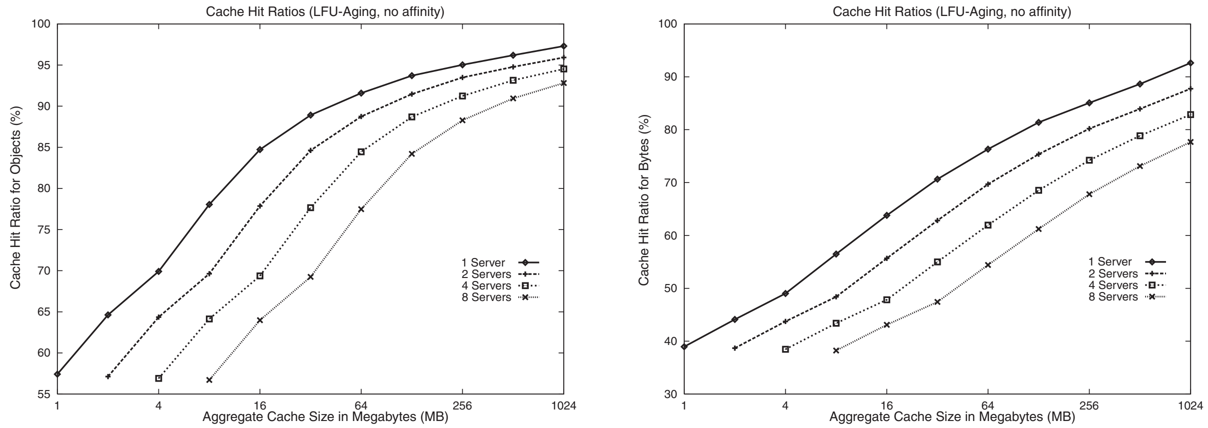
As can be seen, distributing a fixed total amount of cache space over an increasing number of servers has an adverse impact on cache performance. This is not surprising, however, owing to the replication of objects that occurs with multiple caches, and load distribution using the Load policy. What is perhaps more surprising, is that cache performance degrades slightly with an increase in number of servers, even when the per-server cache size is held constant. This degradation is particularly evident for the byte hit ratio, and, to a lesser extent, for the object hit ratio with larger aggregate cache sizes. An explanation for this phenomenon lies in the splitting of the reference stream when multiple servers are present. With just one server (and one cache), for example, an object that is referenced several times within a reasonably short time interval following its initial reference will likely still be in the cache for the references subsequent to the first. With eight servers (and eight caches), however, these subsequent references may be directed to different servers owing to load balancing considerations, resulting in misses at the respective caches. More generally, the locality properties of cache reference stream(s) are adversely affected by splitting.

4.2 Use of Cache Affinity Information in Load Distribution

The effects of employing cache affinity information in distribution decisions are investigated first in the context of the Load policy. The degree to which affinity information is considered is controlled by a policy parameter ϵ, which reflects the amount of load imbalance a load distribution policy is willing to trade for cache affinity. If a requested object is not cached, the request is simply sent to the server with the lightest load. If, however, the object is cached (in one or more local caches), the most lightly-loaded server that has the object cached is selected, provided that its load is within ϵ percent of the most lightly loaded of all the servers. Thus, an ϵ value of "infinity" leads to load distribution on the basis of cache affinity alone (note that there is no replication of objects in multiple caches in this case), whenever the object requested is present in some cache, while an ϵ value of 0 leads to load distribution on the basis of only server load.

Sample caching performance results for the affinity-based policy are shown in Figure 4. There are four pairs of lines, and each pair of lines represents a fixed aggregate cache size, in both 2-server and 4-server configurations. The values plotted are the cache hit ratios for objects as functions of the ϵ values used for the affinity policy (the results for byte hit ratio are qualitatively similar).

Even small values of ϵ, such as 1%, result in a significant increase in the cache hit ratio for the affinity policy, suggesting that the loads on the individual server nodes are close to equal much of the time.



(a) (b)

Figure 3: *Caching Performance as a Function of Aggregate Cache Size and Number of Servers in the Cluster (Load): (a) Object Hit Ratio; (b) Byte Hit Ratio*

The cache hit ratio tends to increase with ϵ, but not always monotonically, since small changes in ϵ can affect the dispatch decision for a given request, and thus the placement (and replacement) of documents in server caches. The cache hit ratio is highest for $\epsilon = \infty$, since the "pure affinity" policy does not allow document replication in the caches.

Intermediate values of ϵ affect the caching performance in subtle ways. When ϵ is small (say, less than 30%), the 2-server configuration invariably outperforms the 4-server configuration with the same aggregate cache size. The reason for this, as noted earlier, is the higher cache hit ratio achievable with a larger per-server cache size. This effect is most pronounced on small aggregate cache sizes, and less pronounced as the aggregate cache size increases. When ϵ is large (say, greater than 60%), the cache hit ratio of the 4-server configuration approaches that of the 2-server configuration, and even outperforms it in some instances. The latter effect is most evident on the larger aggregate cache sizes. In other words, the use of affinity information can (at the very least) recoup the caching performance that is typically lost when additional server nodes (with a fixed aggregate cache size) are added to a server configuration.

With respect to load balancing performance, the effect of ϵ on the LBM is small for ϵ in the range of 0 to 100% (Table 4). Thus, it is possible to achieve both good cache performance and good load balance with a distribution policy that considers both load information and cache affinity information. It is worthwhile to send a request to a server that has the object cached, even if this violates load balance in the immediate term, because there is a good chance that the load imbalance can be improved by sending the next request to a different server.

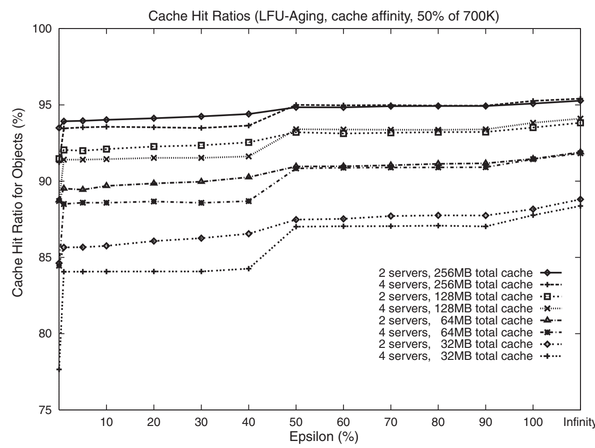


Figure 4: *Effect on Cache Performance (Objects) of Using Cache Affinity Information in Load Distribution (Load/Affinity), per-request bandwidth at 50% of 700 KB/sec*

Furthermore, an omniscient cache manager with knowledge of all current cache contents is not necessarily required. Similar cache performance can be achieved with a static mapping of Web objects to server caches, using hashing on URL names (compare Figure 5 and Figure 4). Load balancing performance suffers only modestly in this case (Table 5).

ϵ	LBM	
	Bandwidth	Requests
0	1.059	1.142
20	1.062	1.150
40	1.064	1.152
60	1.067	1.158
80	1.068	1.160
100	1.071	1.165
∞	1.160	1.293

Table 4: *Effect of Epsilon Value on Load Balancing (Load/Affinity, 2 servers, 32 MB aggregate cache size, per-server bandwidth of 700 KB/sec, per-request bandwidth constraint 50% (350 KB/sec))*

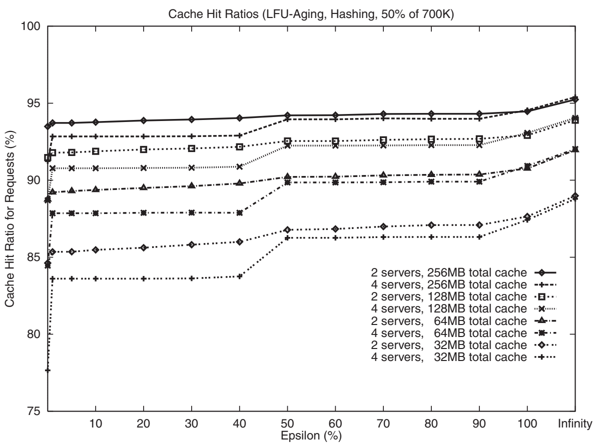


Figure 5: *Effect on Cache Performance (Objects) of Using Static Hashing of URL Names in Load Distribution (Load/Affinity, static hashing, per-server bandwidth of 700 KB/sec, per-request bandwidth constraint 50% (350 KB/sec))*

ϵ	LBM	
	Bandwidth	Requests
0	1.059	1.142
20	1.076	1.168
40	1.098	1.202
60	1.122	1.241
80	1.139	1.266
100	1.161	1.299
∞	1.310	1.484

Table 5: *Effect of Epsilon Value on Load Balancing (Load/Affinity, static hashing, 2 servers, 32 MB aggregate cache size, per-server bandwidth of 700 KB/sec, per-request bandwidth constraint 50% (350 KB/sec))*

4.3 Impact of Using Cache Affinity on Load vs. Round Robin

Table 6 compares the relative improvement in load balancing performance of Load vs. Round-Robin to that of Load/Affinity vs. Round-Robin/Affinity, for varying per-request bandwidth constraints. With Load/Affinity and Round-Robin/Affinity an "infinite" value of ϵ is selected, implying that when the requested object is present in some cache, both load distribution policies will allocate the request to the respective server. If no server has the object cached, the Load/Affinity policy will send the request to a server with the lightest load, while the Round-Robin/Affinity policy is defined to send the request to a server to which it has sent the fewest requests.

The results in Table 6 show that Round-Robin/Affinity is largely ineffective. Similar behaviour (although less dramatic) is observed when decisions are based on intermediate combinations of load balance and affinity considerations, rather than pure affinity. It thus appears that current server load information must be employed for load balancing purposes, when cache affinity information is used as well. This is because when load distribution employs cache affinity information, decisions are based less frequently on load balancing considerations. It therefore becomes necessary to check current conditions explicitly, rather than relying solely on the history of past load distribution decisions.

Maximum Per-Request Bandwidth (%)	Without Affinity	With Affinity
10	9.8	43.1
20	12.4	44.7
40	15.1	48.7
60	16.4	49.3
80	17.1	48.7
100	17.6	49.0

Table 6: *Relative Performance Improvements of Load as Compared to Round-Robin: % difference in LBM Bandwidth (2 servers, 128 MB aggregate cache, per-server bandwidth of 700 KB/sec)*

5 Relation to Other Work

There is a vast and growing literature on scalable, high performance Web servers. For brevity, we comment here on only two recent works that we consider most relevant to our own, and discuss how this work differs from ours.

Goldszmidt *et al.* [12] have done significant work on the design, implementation, and deployment of large-scale Web servers, most notably for the 1996 and 1998 Olympics. A major focus in their work has been the design and implementation of a high performance TCP connection router, for which the primary metric is connection throughput (i.e., the maximum number of TCP connections per second that can be supported by the server). A secondary focus has been on resource utilization (e.g., TCP connection state and memory buffer consumption by slow modem users), load balancing performance, and end user response time. To the best of our knowledge, their work did not consider affinity-based caching and its related tradeoffs with caching system performance.

One paper that has proposed a cache affinity approach is [19]. In particular, the Locality-Aware Request Distribution (LARD) policy is proposed as an example of a content-aware request distribution policy. The approach is evaluated using simulation, with a prototype implementation for validation. They find that their approach offers significant improvements in server throughput, and significant reductions in server idle time (i.e., load imbalance). The notions of affinity and controlled replication of documents in the LARD work are similar to ours, but the simulation models (and the traces used) are different, in several ways. For example, our simulation model presents the workload to the server based on the timestamps in the trace; we do not adjust the request arrival rate so as to necessarily ensure a steady flow of requests. This is important in the evaluation of load balance, which depends greatly on the timing of arrivals. Furthermore, we do not assume infinite network bandwidth; rather, we make the per-request bandwidth a parameter to our simulation, orthogonal to the server bandwidth. This allows us to explore a broader spectrum of the server design parameter space. The TCP connection handoff procedure described in [19] is complementary to our work, and would be a valuable component in a prototype implementation of our own clustered server.

6 Conclusions

Trace-driven simulation was used to compare various policies for load distribution in clustered Web servers. Our results suggest that very simple policies such as round-robin may yield good load balance if the achievable per-request bandwidth is strongly network or client limited, otherwise policies that take into account current server loads are required. As well, the relative importance of using information on current server loads for load balancing purposes increases with the number of servers.

Achieving both good cache performance and good load balance is possible, but it requires use of policies that take both objectives into consideration and that make use of information concerning server loads and the cache affinities of incoming requests. Cache affinities may be effectively estimated through hashed assignment of the URL space.

This paper considers only the case of multiple machines cooperating to provide the Web server function at a single physical location. Work in progress concerns the load distribution problem in systems with geographically distributed caching/replication, in which a number of new considerations arise.

7 Acknowledgements

The authors are grateful to Hewlett-Packard for providing two of the Web server access logs used in this research, as well as for an equipment donation in support of this research. The authors also thank Martin Arlitt for his substantial contribution to the design and implementation of our simulator for clustered Web server architectures, while employed as a research assistant at the University of Saskatchewan.

Financial support for this research was provided by CANARIE, through the Advanced Networks Applications (ANA) program, and by Telecommunications Research Laboratories (TR*Labs*) in Saskatoon. Additional support was provided through NSERC research grants OGP0003707, OGP0000264, and OGP0121969.

References

[1] D. Andresen, T. Yang, V. Holmedahl, and O. Ibarra, "SWEB: Towards a Scalable World Wide Web Server on Multicomputers", *Proceedings of the 10th International Parallel Processing Symposium (IPPS '96)*, Hawaii, April 1996.

[2] M. Arlitt and C. Williamson, "Internet Web Servers: Workload Characterization and Performance Implications", *IEEE Transactions on Networking*, Vol. 5, No. 5, pp. 631-645, October 1997.

[3] H. Balakrishnan, S. Seshan, M. Stemm, and R. Katz, "Analyzing Stability in Wide-Area Network Performance", *Proceedings of the 1997 ACM SIGMETRICS Conference*, Seattle, WA, pp. 2-12, June 1997.

[4] A. Bestavros, "WWW Traffic Reduction and Load Balancing through Server-Based Caching", *IEEE Concurrency*, Vol. 5, No. 1, pp. 55-67, January-March 1997.

[5] R. Caceres, F. Douglis, A. Feldmann, G. Glass, and M. Rabinovich, "Web Proxy Caching: The Devil is in the Details", *Performance Evaluation Review*, Vol. 26, No. 3, pp. 11-15, December 1998.

[6] P. Cao, L. Fan, and Q. Jacobson, "Web Prefetching Between Low-Bandwidth Clients and Proxies: Potential and Performance", to appear, *Proceedings of the 1999 ACM SIGMETRICS Conference*, Atlanta, GA, May 1999.

[7] A. Chankhunthod, P. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell, "A Hierarchical Internet Object Cache", *Proceedings 1996 USENIX Conference*, January 1996.

[8] M. Crovella, M. Harchol-Balter, and C. Murta, "Task Assignment in a Distributed System: Improving Performance by Unbalancing Load", *Proceedings of the 1998 ACM SIGMETRICS Conference*, Madison, WI, pp. 268-269, June 1998. (Poster paper – extended abstract only.)

[9] D. Dias, W. Kish, R. Mukherjee, and R. Tewari, "A Scalable and Highly Available Web Server", *Proceedings IEEE Computer Conference (COMPCON)*, Santa Clara, CA, pp. 85-92, March 1996.

[10] A. Fox, S. Gribble, Y. Chawathe, E. Brewer, and P. Gauthier. "Cluster-based scalable network services", *Proceedings of the Sixteenth ACM Symposium on Operating System Principles*, San Malo, France, October 1997.

[11] M. Garland, S. Grassia, R. Monroe and S. Puri, "Implementing Distributed Server Groups for the World Wide Web", Technical Report CMU-CS-95-114, Carnegie Mellon University School of Computer Science, January 1995.

[12] G. Goldszmidt and G. Hunt, "Shock Absorber: A TCP Connection Router", *IEEE GLOBECOM 97*, Nov. 3-8, 1997, Phoenix, AZ.

[13] J. Gwertzman, "Autonomous Replication in Wide-Area Distributed Information Systems", Technical Report TR-95-17, Harvard University Division of Applied Sciences, Center for Research in Computing Technology, 1995.

[14] IBM Womplex Home Page. Available from `http://www.womplex.ibm.com`

[15] T. Kwan, R. McGrath, and D. Reed, "NCSA's World Wide Web Server: Design and Performance", *IEEE Computer*, Vol. 28, No. 11, pp. 68-74, November 1995.

[16] E. Lazowska, J. Zahorjan, G. Graham, and K. Sevcik, *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*, Prentice-Hall, 1984.

[17] D. Mosedale, W. Foss, R. McCool, "Lessons Learned Administering Netscape's Internet Site", *IEEE Internet Computing*, Vol. 1, No. 2, pp. 28-35, 1997.

[18] V. Padmanabhan and J. Mogul, "Using Predictive Prefetching to Improve World-Wide Web Latency", *ACM SIGCOMM, Computer Communication Review*, Vol. 26, No. 3, pp. 22-36, July 1996.

[19] V. Pai, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. Nahum, "Locality-Aware Request Distribution in Cluster-based Network Servers", *Proceedings of the Eighth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VIII)*, San Jose, CA, October 1998.

[20] V. Valloppillil and K. Ross, "Cache Array Routing Protocol v1.0", Internet Draft, February, 1998. Available at URL: `http://ds.internic.net/internet-drafts/draft-vinod-carp-v1-03.txt`

[21] D. Wessels and K. Claffy, "Evolution of the NLANR Cache Hierarchy: Global Configuration Challenges", 1996. Available at URL: `http://www.nlanr.net/Papers/Cache96/`

[22] S. Williams, M. Abrams, C. Standridge, G. Abdulla and E. Fox, "Removal Policies in Network Caches for World-Wide Web Documents", *Proceedings of the 1996 ACM SIGCOMM Conference*, Stanford, CA, pp. 293-305, August 1996.

[23] C. Yoshikawa, B. Chun, P. Eastham, A. Vahdat, T. Anderson, and D. Culler, "Using Smart Clients to Build Scalable Services", *Proceedings of the 1997 USENIX Technical Conference*, 1997.

[24] H. Zhu, B. Smith, and T. Yang, "A Scheduling Framework for Web Server Clusters with Intensive Dynamic Content Processing", to appear, *Proceedings of the 1999 ACM SIGMETRICS Conference*, Atlanta, GA, May 1999. (Poster paper – extended abstract only.)