Performance Analysis of Scalable Web Hosting Service with Flex: Two Case Studies

Ludmila Cherkasova, Mohan DeSouza[1],
Shankar Ponnekanti[2]
Computer Systems and Technology Laboratory
HP Laboratories Palo Alto
HPL-2000-28
February, 2000

E-mail: cherkasova@hpl.hp.com
mdesouza@cs.ucr.edu
pshankar@cs.stanford.edu

web hosting service, web server cluster, load balancing, scalability, super linear speedup, performance analysis

FLEX is a new cost effective, "locality aware" load balancing solution for a shared web hosting service implemented on a cluster of machines [C99]. FLEX allocates hosted web sites to different machines in the cluster based on the sites' processing and memory requirements which are estimated using the site logs.

Appropriate routing of requests can be achieved by using the DNS infrastructure, since each hosted web site has a unique domain name. FLEX is simple, inexpensive and can be easily implemented on top of the current infrastructure used by Web hosting service providers. The performance benefits achieved by FLEX from both load distribution and efficient memory usage make it a viable solution for low-end shared web hosting services.

Using two case studies (based on real traces), we evaluate the potential benefits of the new solution. We compare the performance of FLEX against *Round-Robin* and *Optimal* strategy. FLEX significantly outperforms *Round-Robin* (up to 130% in average server throughput), getting within 5%-15% of optimal performance achievable for those traces. Miss ratio is improved 2-6 times. FLEX solution shows superliner speedup when the number of nodes is increased from four to eight because it takes advantage of both the doubled processing power and memory.

Internal Accession Date Only

[1] University of California, Dept of Computer Science, Riverside, CA 92521, USA
[2] Stanford University, Dept of Computer Science, Stanford, CA 94305, USA

Contents

1 Introduction

Demand for Web hosting and e-commerce services continues to grow at a rapid pace. In Web content hosting, providers who have a large amount of resources (for example, bandwidth, disks, processors, memory, etc.) store and provide Web access to documents for institutions, companies and individuals who lack the resources, or the expertise to maintain a Web server, or are looking for a cost efficient, "no hassle" solution. According to Forrester Research Inc, more than two-thirds of all corporate web sites are now hosted (outsourced).

The shared hosting market targets small and medium size businesses. It is a robust, high volume, low-unit cost business. The most common purpose of a shared hosting Web site is marketing. In this case, many different sites are hosted on the same hardware. A shared Web hosting service creates a set of virtual servers on the same server. This supports the illusion that each host has its own web server, when in reality, multiple "logical hosts" share one physical host. Web server farms and clusters are used in Web hosting infrastructures for scalability and availability.

Traditional load balancing solutions try to distribute requests uniformly across all nodes regardless of the content. This interferes with efficient use of RAM in the cluster. The popular files tend to occupy RAM space in all the nodes. This redundant replication of content across the RAM of all the nodes leaves much less RAM available for the rest of the content, leading to a worse overall system performance. With these approaches, the effective net RAM of the cluster is much less than the combined RAM of all the machines. Further, some of these solutions also interfere with HTTP sessions.

A better approach would be to partition the content among the machines thus avoiding document replication in the RAMs. However, static partitioning will inevitably lead to an inefficient, suboptimal and inflexible solution, since the access patterns tend to vary dramatically over time, and static partitioning does not accommodate for this.

The observations above have led to the design of "locality aware" balancing strategies [LARD98] which aim to avoid unnecessary document replication across the RAM's of the nodes to improve the overall performance of the system. These are also known as content based routing strategies.

In this paper, we analyze a new scalable, "locality aware" solution FLEX for load balancing and management of an efficient Web hosting service [C99]. For each web site hosted on a cluster, FLEX evaluates (using web server access logs) the system resource requirements in terms of the memory (site's working set) and the load (site's access rate). The sites are then partitioned into N balanced groups based on their memory and load requirements and assigned to the N nodes of the cluster respectively. Since each hosted web site has a unique domain name, the desired routing of requests can be easily achieved by submitting appropriate configuration files to the DNS server.

Using two case studies (based on real traces), we evaluate the potential benefits of the new

solution. We compare the performance of FLEX against *Round-Robin* for a cluster with four nodes. We also compare FLEX with an *Optimal* strategy (which avoids all document replication and does perfect load balancing (at per-request granularity). FLEX significantly outperforms *Round-Robin* (up to 130% in average server throughput), getting within 5%-15% of optimal performance achievable for those traces. Miss ratio is improved 2-6 times. To study the scalability issues of *Round-Robin* versus FLEX strategy, we performed simulations for four- and eight-node clusters. The "speedup" under Round-Robin strategy for an eight-node cluster is only due to doubled "processing" power. FLEX solution shows superliner speedup when the number of nodes is increased from four to eight because it takes advantage of both the doubled processing power and memory.

The main attractions of the FLEX approach are ease of deployment and an extremely attractive cost/performance tradeoff. This solution requires no special hardware support or protocol changes. There is no single front end routing component. Such a component can easily become a bottleneck, especially if content based routing requires it to do such things as tcp connection hand-offs etc. FLEX can be easily implemented on top of the current infrastructure used by Web hosting service providers.

The paper is structured as follows. Section 2 outlines typical architecture platforms used for web hosting services and provides a detailed survey of existing load balancing strategies and solutions. Section 3 describes FLEX. Sections 4, 5 present simulation results and analysis for two different sets of traces.

2 Load Balancing Solutions

Different architectures have been used for multi-node web servers. One popular architecture is a farm of web servers with replicated disk content. Another popular architecture is the clustered architecture, which consists of a group of nodes connected by a fast interconnection network, such as a switch. It assumes some underlying software layer (e.g., virtual shared disk) which makes the interconnection architecture transparent to the nodes. The NSCA prototype of the scalable HTTP server based on two-tier architecture is described and studied in [NSCA96]. In all architectures, each web server has the access to all the content. Therefore, any server can satisfy any client request.

Traditional load balancing solutions can be partitioned in two major groups:

- DNS Based Approaches;
- IP/TCP/HTTP Redirection Based Approaches;
  - hardware load-balancers;
  - software load-balancers.

2.1 DNS Based Approaches

Round-Robin DNS [RRDNS95] is built into the newer versions of DNS. Round-Robin DNS

distribute the requests among the nodes in the cluster using the following technique: for a name resolution, the DNS server returns the IP address list (for example, list of nodes in a cluster which can serve this content, see Figure 1), placing the different address first in the list for each successive request. Thus, different clients are mapped to different server nodes in the cluster.

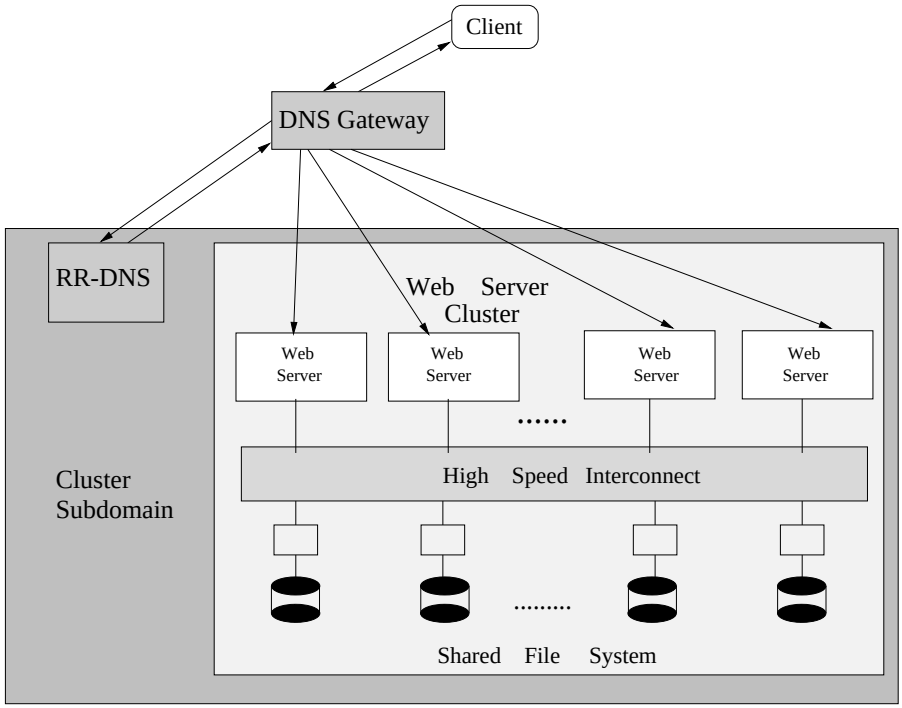

Figure 1: Web Server Cluster Balanced with Round-Robin DNS.

Round-Robin DNS is widely used: it is easy to set up and provides reasonable load balancing. Further, it uses the existing DNS infrastructure, i.e., there is no additional cost.

The Round-Robin schema has at least two drawbacks: it simply posts packets to the next server on the chain – without verifying whether it is already overloaded. The second drawback is that in the event of one of the nodes becoming unavailable, the DNS server still keeps sending the requests to this node as it has no way of detecting this. There are at least two commercial products that attempt to solve these problems associated with DNS load balancing: Cisco DistributedDirector [Cisco] and HP Network Connection Policy Manager.

2.2 IP/TCP/HTTP Redirection Based Approaches

There are several commercial hardware/software load-balancer solutions that can distribute incoming stream of requests among a group of Web servers. Load-balancing boxes are basically IP routers.

Hardware load-balancing servers are typically positioned between a router (connected to the Internet) and a LAN switch which fans traffic to the Web servers. Typical configuration

is shown in Figure 2.

In essence, they intercept incoming web requests and determine which web server should get each one. Making that decision is the job of the proprietary algorithms implemented in these products. This code takes into account the number of servers available, the resources (CPU speed and memory) of each, and the number of active TCP sessions being serviced. The algorithms also monitor each web server by time-stamping the packets and noticing how long it takes to for a request to be handled by the server and returned. In theory, faster servers are sent more requests than their slower counterparts. The balancing methods vary across the different load-balancing servers, but in general, the idea is to forward the request to the least loaded server in the cluster.

Load-balancing server acts as a fast regulating valve between the Internet and the pool of servers. The load balancer uses a virtual IP address to communicate with the router, masking the IP addresses of the individual servers. Only the virtual address is advertised to the Internet community, so the load balancer also acts as a safety net.

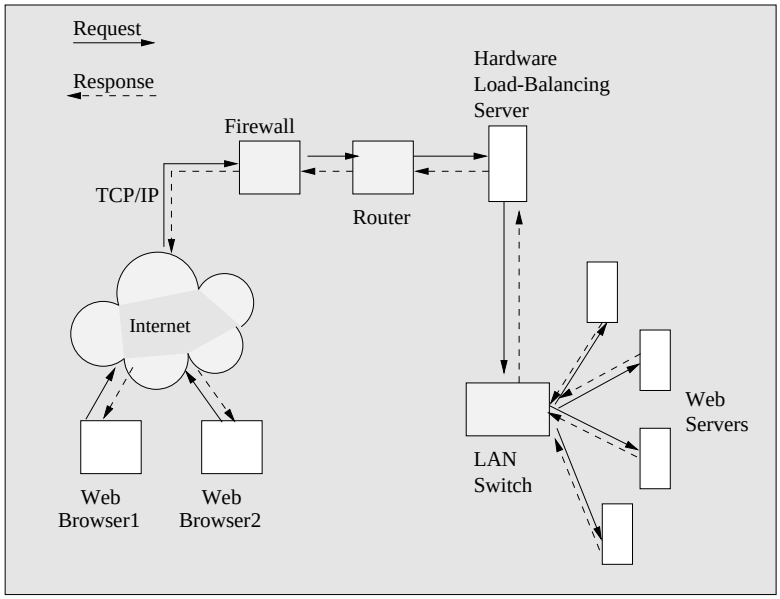

Figure 2: Web Server Farm with Hardware Load-Balancing Server.

The IP addresses of the individual servers are never sent back to the Web browser. If they were, the browser would try to establish a session with a specific Web server, rather than with the load-balancing server. This would defeat the entire purpose of deploying a load-balancing server to distribute the requests. Both inbound requests and outbound responses must pass through the balancing server, causing the load-balancer to become a potential bottleneck.

Four of the six commercial hardware load balancers we surveyed are built around Intel pentium processors: LocalDirector from Cisco Systems [Cisco], Fox Box from Flying Fox [FlyingFox], BigIP from F5 Labs [F5Labs], and Load Manager 1000 from Hydraweb Technologies Inc. [HydraWEB].

The other two load balancers employ a RISC chip: Web Server Director from RND Networks Inc. [RND] and ACEdirector from Alteon [Alteon]. All these boxes but for Cisco's and RND's run under Unix. Cisco's LocalDirector runs a derivative of the vendor's IOS software; RND's Web Server Director also runs a proprietary system.

The software load balancers take a different tack, handing off the TCP session once a request has been passed along to a particular server. In this case, the server responds directly to the browser (see Figure 3). Vendors claim that this improves performance: responses don't have to be rerouted through the balancing server, and there's no additional delay while an internal IP address of the server is retranslated into the advertised IP address of the load balancer.

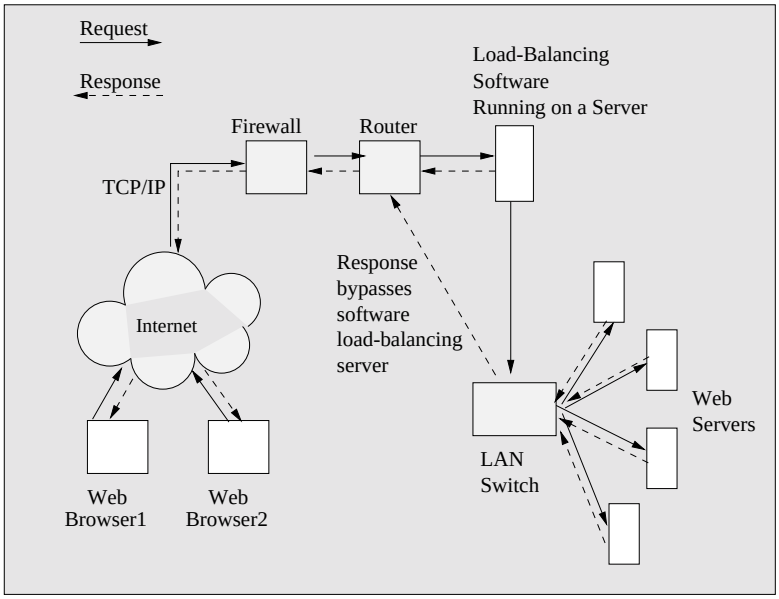

Figure 3: Web Server Farm with Load-Balancing Software Running on a Server.

Actually, the translation is handled by the Web server itself. Software load balancers are sold with agents that must be deployed on the Web server. It's up to the agent to put the right IP address on a packet before it's shipped back to a browser. If a browser makes another request, however, that's shunted through the load-balancing server.

Three commercial software load-balancing servers are available: ClusterCATS from Bright Tiger Technologies [BrightTiger], SecureWay Network Dispatcher from IBM [IBM-SWND], and Central Dispatch from Resonate Inc [Resonate]. These products can be used with Unix or Windows NT servers.

More complete survey on load balancing solutions can be found in [Bruno97, Roberts98, C99].

2.3 Locality Aware Balancing Strategies

Traditional load balancing solutions (both hardware and software) for a web server cluster try to distribute the requests uniformly on all the machines. As mentioned earlier, this adversely affects efficient memory usage because content is replicated across the caches of all the machines.[1] This may significantly decrease overall system performance. These observations have led the researchers to propose new "locality aware" balancing strategies [LARD98].

In the LARD approach, the cluster nodes are partitioned into two sets: front ends and back ends. Front ends act as smart routers or switches: their functionality is similar to load-balancing software servers described above. Front end nodes implement LARD to route the incoming requests to the appropriate node in a cluster. LARD takes into account both document locality and the current load. They show that on workloads with working sets that do not fit in a single node's RAM, LARD improves throughput by a factor of two to four for a 16 node cluster (for the traces they experimented with).

The elements of "content aware" strategies can be found in the commercial products of Alteon Networks Inc. (San Jose, Calif.) and Arrowpoint Communications Inc. (Westford, Mass).

3 New Scalable Web Hosting Solution: FLEX

3.1 Motivation for FLEX

The load balancing solutions surveyed in Section 2.2 are expensive, complicated, often require additional hardware and introduce additional latency. Some of these solutions also hinder http sessions which are an integral part of modern web applications. Almost all of them have a front end routing/ switching component that can become a bottleneck as the cluster size increases. Further, most of them do not take into account request locality. For a cluster of machines serving a single web site, taking request locality into account would require tcp connection handoffs or other such equivalent complicated techniques because the actual requests are seen only after the tcp connection is established and redirecting the request at this stage involves a lot of work to transfer the state associated with the tcp connection.

On the other hand, for a shared web hosting service where a cluster of machines serve a number of sites each with a different domain name, a locality aware solution can be achieved in a much simpler manner. Partitioning of content based on sites rather than files allows us to achieve content based routing (using DNS infrastructure) without the need for tcp connection handoffs.

The motivation for FLEX was to design a simple, inexpensive, easy to deploy, scalable, locality aware load distribution scheme for web hosting services. Such a scheme can also be used for serving content with different domain names within the same corporation. We compared the performance of FLEX with an ideal Round-Robin scheme to demonstrate the importance of

[1] We are interested in the case when the overall file set is greater than the RAM size of one node. If the entire file set completely fits to the RAM of a single machine, any of existing load balancing strategies provides a good solution.

efficient memory usage. We also compare FLEX with an optimal strategy that does perfect load balancing (at per request granularity) and most efficient memory usage. Results show that FLEX is not far off from the optimal strategy indicating that it does a very good job of load distribution and efficient memory usage.

3.2 FLEX system

The goal of FLEX is to assign sites to the nodes in the cluster to achieve both load balancing and efficient memory usage. Because different sites have different amounts of traffic, we need an intelligent algorithm for allocating sites to machines such that the load on each of the nodes is approximately the same. Our approach is to allocate sites to the nodes such that memory and processing requirements are approximately same on all the nodes. We use the working set sizes and access rates of sites as a metric for judging their memory and load requirements.

Let there be a total of S sites hosted on a cluster of N web servers. For each web site s, we build the initial "site profile" *SP_s* by evaluating the following characteristics:

- *A(s)* - the access rate to the content of a site s (in *bytes* transferred during the observed period P);
- *W(s)* - the combined size of all the accessed files of site s (in *bytes* during the observed period P, so-called "working set");

This site profile is entirely based on information which can be extracted from the web server access logs of the sites.

Next, we partition all the sites into N "equally balanced" groups: *S_1, ..., S_N* such that the cumulative access rates and cumulative working sets in each of those N groups are approximately the same. Finally, we assign one server for each group of sites *S_i*.

As mentioned earlier, static partitioning won't work because traffic patterns of sites change over time. FLEX monitors the sites' requirements periodically (at some prescribed time interval: daily, weekly etc). If the sites' requirements change, and the old allocation (partition) isn't good anymore – a new allocation (partition) of sites to machines has to be generated. If a new partition does not take into account the existing "old" partition, it could lead to temporary system performance degradation till the cluster memory usage stabilizes under the new partition. This is because when a site is allocated to be served by a new server, none of the content of this site is available in the RAM of this new server, and hence all the files would have to be downloaded from the disk. Thus, it is desirable to generate a new balanced partition which changes the assigned servers for a minimal number of sites from the existing, "old" partition.

FLEX depends on (fairly) accurate evaluation of the sites' working sets and access rates, especially in the presence of sites with **large** working set and/or **high** access rate requirements. A large site needs to be allocated to ("replicated on") more than one server when a single

server does not have enough resources to handle all the requests to this site. Thus, we need algorithms for:

- Estimating the working sets and access rates of all the sites from the access logs.
- Finding the number of servers each site should be allocated to. Small sites may be put on a single server but larger sites might have to be put on multiple servers.
- Finding the new working sets and access rates for sites put on multiple servers. When a site is put on k servers, the working set and access rate for this site on each of these k servers are (obviously) different from the total working set and access rate of this site.
- Finding a balanced allocation of sites to servers. The goal is to ensure that cumulative memory requirements and load are approximately same on each server. Further, the allocation must change the site-to-server mapping for a minimal number of sites. If a total remapping is done, there is heavy performance degradation during the switch from the old to new allocation.

In [CP2000], we presented heuristic algorithms for solving these problems. For the details, we refer the readers to [CP2000].

FLEX makes use of the DNS infrastructure to achieve the desired routing. Once the allocation of sites to machines is done, the corresponding information is submitted to the DNS server. All the sites in group S_i are mapped to the IP address of node i. Latest version of BIND 8.1.2 supports Dynamic Update standard described in RFC 2136. This allows authorized agents to update zone data by sending special update messages to add or delete resource records (without restarting DNS server). For a large site replicated on more than one node, DNS Round-Robin is used for routing requests to that site to the servers allocated to that site.

This solution is flexible and easy to manage. Tuning can be done on a daily or weekly basis. If server logs analysis shows enough changes in the working sets and access rates, the "Closest" algorithm finds a better partitioning of the sites to the nodes in the cluster (with minimal number of sites changed routing), and new DNS configuration files are generated. Once DNS server has updated its configuration tables, new requests are routed according to the new configuration files, and this leads to more efficient traffic balancing on the cluster. The entries from the old configuration tables can be cached by some servers and used for request routing without going to the primary DNS server. However, the cached entries are valid for a limited time dictated by the TTL (*time to live).* Once the TTL expires, the primary DNS server is requested for updated information. During the TTL interval, both types of routing: old and a new one, can exist. This does not lead to any problems since all servers have accesses to the whole content and can satisfy any request. The smaller the TTL, the more dynamically adaptive is the system.

FLEX system architecture is shown in Figure 4.

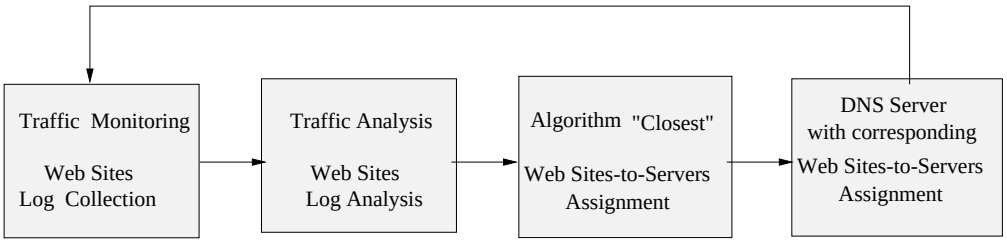

Figure 4: FLEX Strategy: Logic Outline.

Such a self-monitoring solution helps in observing changing site traffic patterns and helps predict future trends and plan for them. During special advertisement, promotional campaigns or events, when one could expect very high traffic rates for a certain site, the mappings can be manually changed (by the service administrator) such that the sites expecting much higher traffic are served by an increased number of servers. The algorithm can be easily made to generate an allocation such that a particular site is allocated to a certain minimum number of servers regardless of its requirements predicted from the logs.

4 Simulation Results: First Case Study

For our first experiment, we used the traces of the HP Web Hosting Service (provided to internal customers). We used the traces for a four-month period: from April 1999 to July 1999. In April, the service had 71 hosted sites. By the end of July, the service had 89 hosted web sites. The next table presents aggregate statistics characterizing the general memory and load requirements for the traces. We also estimate memory requirements for "one-timers" - files accessed only once. Note that "locality-aware" strategies give no benefit for a trace containing only one-timers.

	April	May	June	July
Number of Requests	1,674,215	1,695,774	1,805,762	1,315,685
Working Set (MB)	994.2 MB	878.4 MB	884.9 MB	711.6 MB
Working Set of "Onetimers"	370.0 MB	374.5 MB	311.2 MB	298.3 MB
Access Rate (MB)	14,860 MB	14,658 MB	13,909 MB	8,713 MB
Number of Targeted Files	17,955	16,305	17,915	20,341

(1)

To characterize the "locality" of the traces, we create a *Freq-Size* file: for each file in the trace, we store the number of times the file was accessed (frequency) and its size. *Freq-Size* file is sorted in the order of decreasing frequency. We then compute the cumulative fraction of requests and file sizes, normalized to the total number of requests and total data set size, respectively. The next table shows the locality characteristics of the trace:

Month	Working Set for 97/98/99% of all Requests (in MBytes)	Working Set for 97/98/99% of all Requests (as % of Total WS)
April	242.7 MB / 362.1 MB / 556.3 MB	24.4% / 36.4% / 56.0%
May	249.2 MB / 296.3 MB / 419.9 MB	28.4% / 33.7% / 47.8%
June	196.1 MB / 304.8 MB / 475.1 MB	22.2% / 34.4% / 53.7%
July	155.1 MB / 276.1 MB / 487.9 MB	21.8% / 38.8% / 68.6%

(2)

Smaller numbers for 97/98/99% of the working set indicate higher traffic locality: this means that a larger percentage of the requests target a smaller set of documents. These numbers help us characterize the possible benefits of locality-aware strategies in general. The more locality the trace has – the less benefits one could expect. This is because the main advantage of locality aware strategies is to increase the effective RAM size. The more the locality, the larger is the percentage of requests that can be satisfied from a smaller RAM and thus the benefits of a larger effective RAM are smaller.

In our simulations for the HP Web hosting service, we assumed that the sites are served by a web server cluster with four nodes. We normalize the requirements of the sites such that the total requirements of all the sites are 400 units of memory and 400 units of access rate. With the normalization, each machine has to be allocated a set of sites whose combined access rate and working set both total to 100 units each. Tables 3, 4, 5, 6 show five sites with the largest working sets and with the largest access rate for April, May, June, and July, respectively.

April: Web Hosting Service had 71 hosted sites

Site Number	Largest Working Set (units)	Access Rate (units)	Site Number	Working Set (units)	Largest Access Rate (units)
62	213.9	40.2	57	56.8	95.6
57	56.8	95.6	20	2.7	46.7
17	14.3	3.43	62	213.9	40.2
42	12.2	10.0	67	2.4	34.2
60	12.1	9.8	51	2.7	28.3

(3)

It is interesting to note that the site 62 has a very high working set and accounts for 213 units out of the total of 400 units for all 71 sites! However, the site 62' accounts for only 40.2 units of access rate. Further, there are sites, such as site 20, which have a very small working set (2.7 units of total) but "attract" a large number of accesses (40.2 units of access rate). Such sites have a small number of extremely "hot" files.

May: Web Hosting Service had 74 hosted sites

Site Number	Largest Working Set (units)	Access Rate (units)	Site Number	Working Set (units)	Largest Access Rate (units)
62	135.8	28.4	10	37.7	50.7
57	68.7	47.3	57	68.7	47.3
10	37.7	50.7	20	7.6	43.8
60	19.0	10.7	67	3.2	28.8
31	13.1	22.9	62	135.8	28.4

(4)

Data for May shows that the service' aggregate profile had changed: some of the larger sites in April account for less memory and load requirements, while a few other sites require more system resources.

June: Web Hosting Service had 84 hosted sites

Site Number	Largest Working Set (units)	Access Rate (units)	Site Number	Working Set (units)	Largest Access Rate (units)
62	136.4	35.4	57	74.5	50.6
57	74.5	50.6	20	4.6	42.7
10	18.6	41.0	10	18.6	41.0
60	14.1	10.4	62	136.4	35.4
13	12.9	25.6	67	2.9	32.0

(5)

Data for June shows further trends in the changing site traffic patterns: the memory and load requirements for sites 10 and 20 continue to grow steadily while some other sites disappear from the list of "leaders".

July: Web Hosting Service had 89 hosted sites

Site Number	Largest Working Set (units)	Access Rate (units)	Site Number	Working Set (units)	Largest Access Rate (units)
10	64.2	43.6	20	8.1	46.1
57	49.4	12.6	10	64.2	43.6
5	38.7	6.3	13	12.4	34.5
34	28.4	5.0	1	0.7	34.1
60	19.6	16.6	21	3.1	17.1

(6)

Data for July shows a clear change of "leading sites": sites 10 and 20 became the largest sites with respect to working set and access rate requirements respectively. Site 62's contribution diminishes (it does not appear among the five "leading sites"). In July, the whole service profile became more balanced: there were no sites with excessively large working sets (memory requirement) or access rates (load on the system).

Our simulation model was written using C++Sim [Schwetman95]. The model makes the following assumptions about the capacity of each web server in the cluster:

- Web server throughput is 1000 Ops/sec (or requests/sec) when retrieving files of size 14.6Kbytes from the RAM (14.6Kbytes is the average file size for the SpecWeb96 benchmark, which is an industry standard for measuring web server performance).
- Web server throughput is 10 times lower (i.e., 100 Ops/sec) when it retrieves the files from the disk rather than from the RAM.[2]

[2]We measured web server throughput (on HP 9000/899 running HP-UX 11.00) when it supplied files from the RAM (i.e., the files were already downloaded from disk and resided in the File Buffer Cache), and compared it against the web server throughput when it supplied files from the disk. Difference in throughput was a factor of 10. For machines with different configurations, this factor can be different).

- The service time for a file is proportional to the file size.
- The cache replacement policy is LRU.

Using our partitioning algorithm, a partition was generated for each month. The requests from the original trace for the month were split into four sub-traces based on the strategy. For Round-Robin, the first sub-trace had requests 1, 5, 9 etc, the second sub-trace had requests 2, 6, 10 etc and so on. For FLEX, the first sub-trace had all the requests to all the sites allocated to the first server in that month's partition etc. FLEX might replicate a large site onto several servers. In this case, each request to this site was randomly assigned to one of the servers assigned to this site. The four sub-traces were then "fed" to the respective servers. Each server picks up the next request from its sub-trace as soon as it is finished with the previous request. We measured two parameters: 1) throughput (averaged across 4 servers) in processing all the requests; and 2) miss ratio.

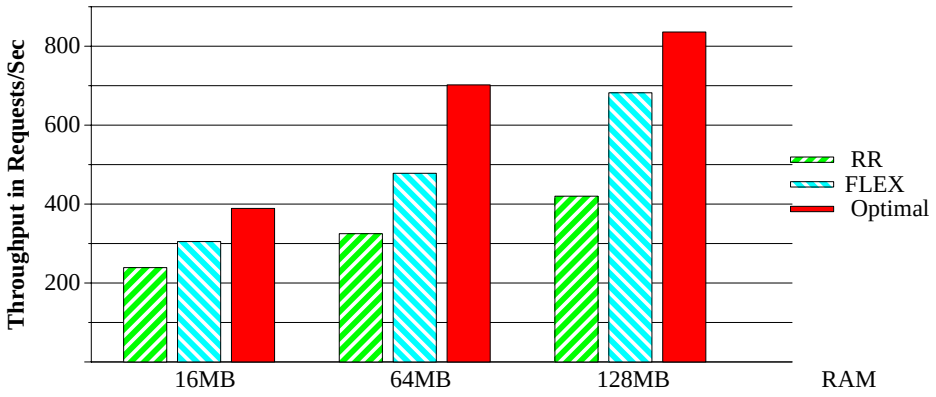

Figure 5: Average Server Throughput in a Four-Node Cluster for April.

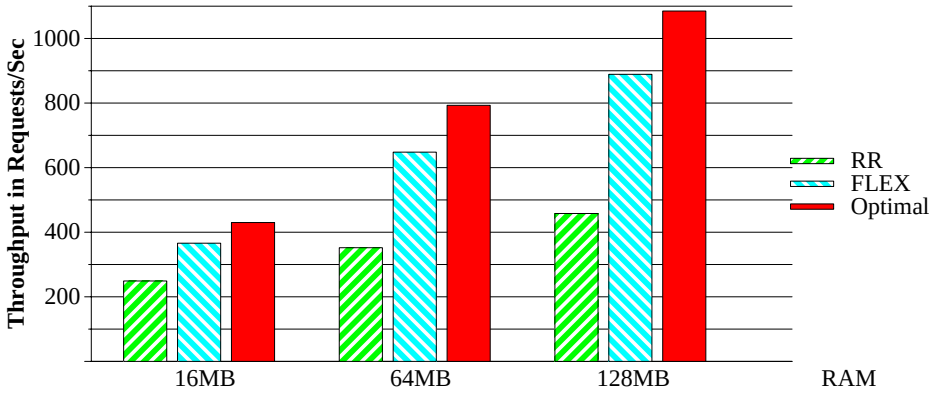

Figure 6: Average Server Throughput in a Four-Node Cluster for May.

To better appreciate the results, we decided to get an upper-bound on the best achievable performance for the particular trace we had. We implemented the "optimal" strategy *Optimal* which has the four servers operating with their RAMs combined. Each request from the

original trace can be served by any server (or rather CPU, since memories are now combined). *Optimal* sets an absolute performance limit for the given trace and the given cache replacement policy because it has perfect load balancing and no replication of content in memory.

Figures 5, 6, 7, 8 show the average server throughput for Round-Robin, FLEX and *Optimal* and different RAM sizes (the RAM sizes shown are per server).

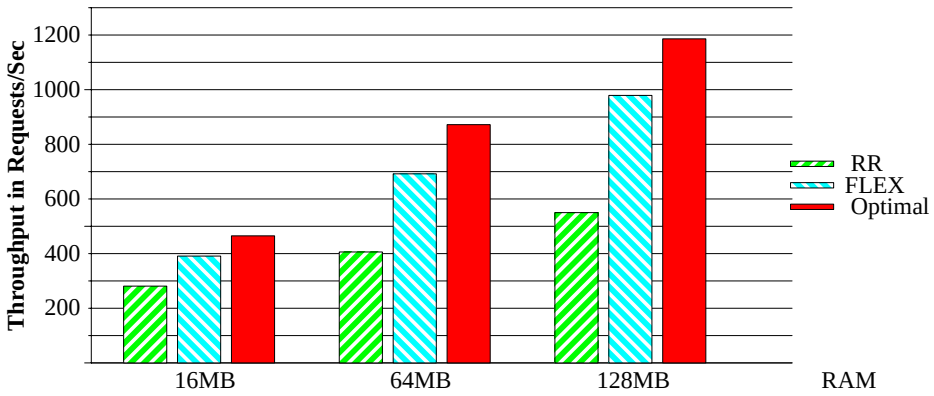

Figure 7: Average Server Throughput in a Four-Node Cluster for June.

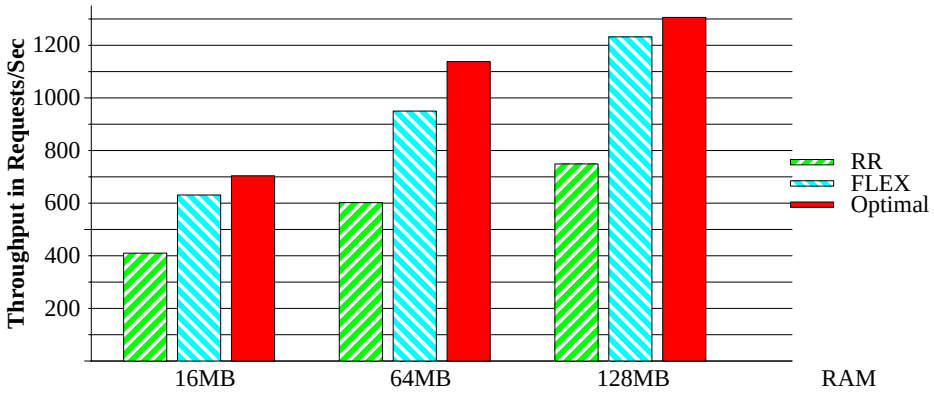

Figure 8: Average Server Throughput in a Four-Node Cluster for July.

Note that when a large site is assigned to multiple servers, there is some loss of memory usage efficiency because the content of this site is now replicated on multiple servers. Thus, FLEX performs best when there are no large sites and each site is allocated to exactly one server. The performance of FLEX is comparatively poorer than *Optimal* in April, May and June because one or more sites (site 62 in all three months and 57 in April) have to be necessarily replicated in these months to achieve balanced partitions. On the other hand, no site had to be replicated in July and the performance of FLEX is much better. In general, FLEX outperforms Round-Robin in all cases, and for larger RAM, the benefits in throughput range from 50% to 100%. For May, June, and July, performance of FLEX strategies is within 5-18% of *Optimal*.

For the traces of April and RAM sizes of 16 MB and 64 MB, FLEX's performance benefits are

modest. One of the reasons for this is a number of accesses to very large files (sizes of 10MB to 15MB) in April. We plan to repeat these simulations for different replacement policies (for example, not caching files larger than a certain size) to study the impact of large files.

Due to the lack of space, we omit the graphs showing the miss ratios: in all cases, miss ratio (FLEX vs Round-Robin) is improved dramatically, by a factor of 2-3. *Optimal* strategy shows the minimum miss ratio, and miss ratios for FLEX strategies are very close to that of *Optimal*.

All our results were for a cluster of four nodes. As the cluster size increases, locality aware strategies like FLEX do even better compared to non-locality aware strategies like Round-Robin. The next case study examines the scalability of FLEX.

5 Simulation Results: Second Case Study

In our second experiment, we used the traces of the busy HP's external web site. These traces represent a large, single web site. To create a web hosting like service, we treat the set of first level directories of this single site as independent web sites. Traces were collected during four days in October, 1999. During these days, the "web hosting service" had 145 hosted "sites". The next table presents aggregate statistics characterizing the general memory and load requirements for the traces.

	Day 1	Day 2	Day 3	Day 4
Number of Requests	6,893,537	6,709,091	6,822,680	6,591,683
Working Set (MB)	1,249.1 MB	1,072.5 MB	1,177.0 MB	1,242.8 MB
Working Set of "Onetimers"	381.6 MB	359.2 MB	404.4 MB	390.9 MB
Access Rate (MB)	17,774 MB	17,056 MB	17,356 MB	17,080 MB
Number of Targeted Files	60,203	50,004	60,707	54,887

(7)

These traces have much higher number of requests (two orders of magnitude higher compared to the first trace), and represent a much busier service. The next table shows the locality characteristics of the traces:

Month	Working Set for 97/98/99% of all Requests (in MBytes)	Working Set for 97/98/99% of all Requests (as % of Total WS)
Day 1	213.4 MB / 341.3 MB / 595.0 MB	17.1% / 27.3% / 47.6%
Day 2	175.5 MB / 265.5 MB / 420.9 MB	16.4% / 24.8% / 39.2%
Day 3	216.8 MB / 322.3 MB / 541.3 MB	18.4% / 27.4% / 46.0%
Day 4	204.3 MB / 292.3 MB / 532.5 MB	16.4% / 23.5% / 42.8%

(8)

These traces have more locality than the traces in the first case study. One of the reasons could be that these traces belong to a single large web site (although artificially partitioned into multiple sites) and not multiple sites as in the first trace. As explained earlier, "locality" is an important characteristic which helps us estimate in advance the potential benefits of locality-aware strategies. In this case, we might expect a lower "return" from FLEX due to

the high locality inherent in the traces. However, the second case study is still very interesting because of higher traffic rate which gives us an opportunity to study the performance benefits of FLEX for different (larger) cluster sizes.

The other distinct feature is that the overall service is more balanced: there are no "sites" with excessively large working sets (memory requirement) or access rates (load on the system).

We performed simulations for two different cluster sizes to study the impact of increasing cluster size. First, we assume that the sites are served by a web server cluster with four nodes. In this case, we normalize the requirements of the sites such that the total requirements of all the sites are 400 units of memory and 400 units of access rate. With the normalization, each machine has to be allocated a set of sites whose combined access rate and working set both total to 100 units each. Next Table 9 show five sites with the largest working sets and with the largest access rate for Day 1.

Site Number	Largest Working Set (units)	Access Rate (units)	Site Number	Working Set (units)	Largest Access Rate (units)
24	35.5	18.1	82	0.1	40.8
98	25.9	4.7	33	24.5	37.4
33	24.5	37.4	108	18.5	30.4
63	21.4	19.2	41	14.2	26.2
135	19.3	9.9	63	21.4	19.2

(9)

Due to space limitations, we show the statistics for Day 1 only. The statistics for the other days differ slightly (by around 5-10 units).

For example, site 24 had the largest working set requirement for Day 1: 35.5 units (with 18.1 units of access rate). During the observed four day period, these requirements varied between 34 units (with 19.9 units of access rate) and 39.8 units (with 17.2 units of access rate).

Site 82 was responsible for highest access rate: 40.8 units (with working set of 0.1 units). During the observed four day period, these requirements varied between 40.1 units (with working set of 0.3 units) and 45.1 units (with working set of 0.09 units).

REMARK: For a cluster with eight nodes, similar analysis holds. However, we normalize the total requirements of all the sites to 800 units. Thus, the numbers from Table 9 are doubled for the eight-node case.

Using our partitioning algorithm, a partition was generated for each day based on the traffic in the **previous** day. For example, the partition used for Day 2 was based on the traffic patterns of Day 1, etc.

Figures 9, 10 show the average achievable server throughput for Round-Robin, FLEX and *Optimal* strategies for Days 1-2 and Days 3-4 respectively for different RAM sizes (per server RAM sizes).

The results are consistent across all the four days. The performance improvements due to FLEX over Round-Robin are 30%-80% depending on the server RAM size (larger RAM gives

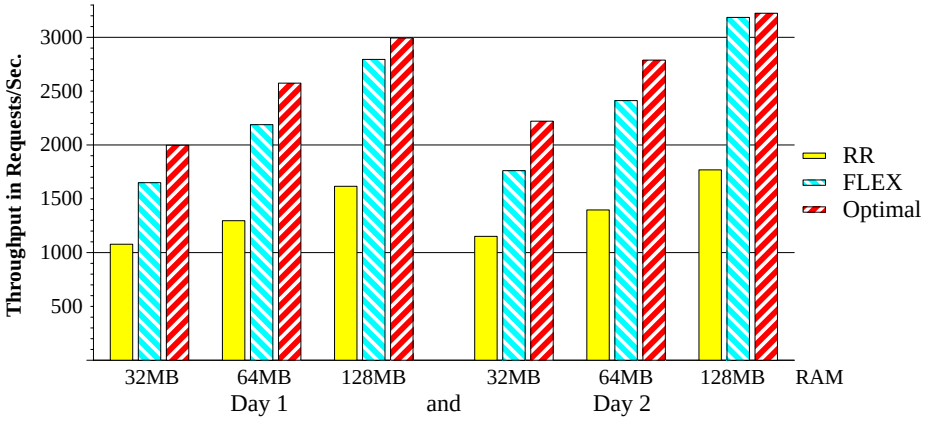

Figure 9: Average Server Throughput in a Four-Node Cluster for Days 1 and 2.

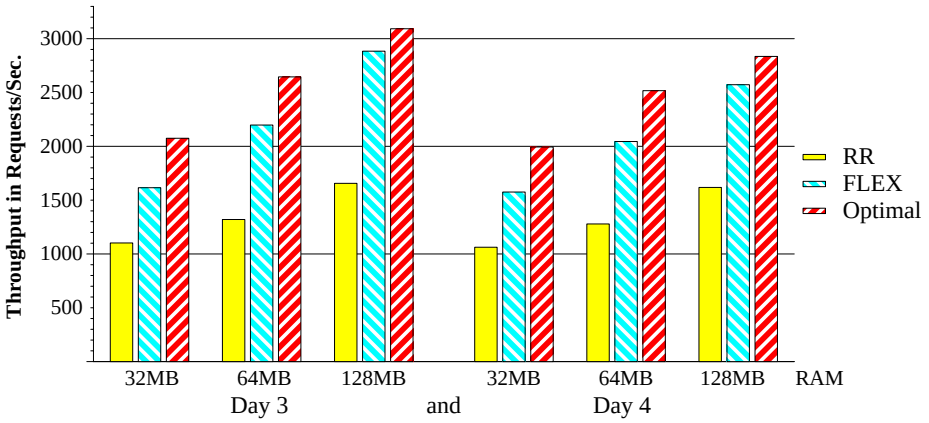

Figure 10: Average Server Throughput in a Four-Node Cluster for Days 3 and 4.

greater performance improvements, because FLEX benefits from more efficient memory usage). FLEX is within 1%-17% of the *Optimal* strategy for all RAM sizes for the given traces. For a RAM size of 128 MBytes, FLEX achieves near optimal performance.

For the second set of experiments on the same traces, we assume that the sites are served by a web server cluster with eight nodes. Using our partitioning algorithm, a partition was generated for a cluster of eight nodes, again based on the **previous** day's traffic patterns. Figures 11, 12 show the average achievable server throughput for Round-Robin, FLEX and *Optimal* strategies for Days 1-2 and Days 3-4 respectively for different RAM sizes (per server RAM sizes)

Performance improvements of FLEX over Round-Robin are much higher for an eight-node cluster than for a cluster with four nodes. Average web server throughput with FLEX is 85%-130% higher compared to the average web server throughput with Round-Robin strategy. FLEX is within 1%-15% of the *Optimal* strategy for the given traces for various RAM sizes. For a RAM size of 128 MBytes, FLEX again achieves nearly optimal performance.

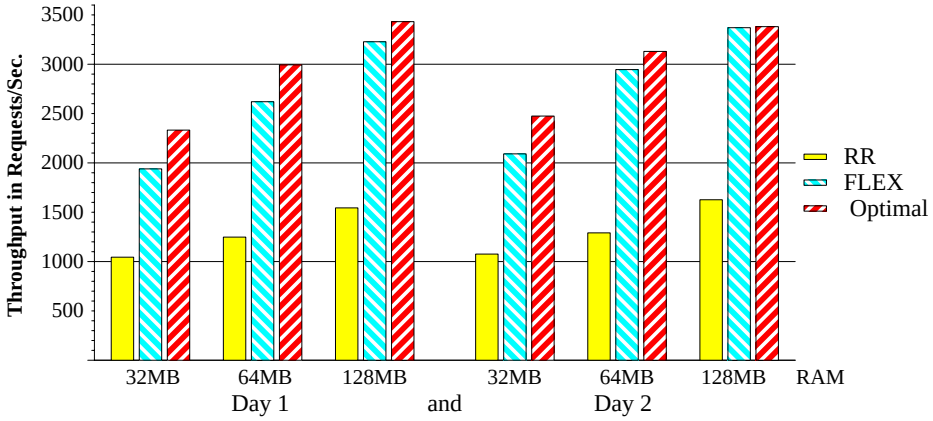

Figure 11: Average Server Throughput in an Eight-Node Cluster for Days 1 and 2.

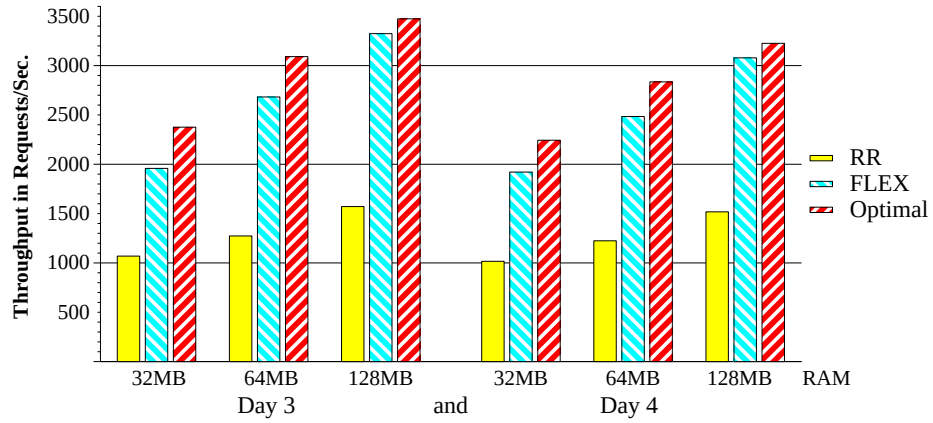

Figure 12: Average Server Throughput in an Eight-Node Cluster for Days 3 and 4.

Next, we analyze the performance of Round-Robin and FLEX for different cluster sizes. The average per-server throughput with Round-Robin strategy for an eight-node cluster is almost same as (in fact, slightly worse than) that of a four-node cluster. Thus a cluster with eight nodes processes the same trace about twice as fast as a cluster with four nodes. However, a cluster with eight nodes has twice the memory in addition to twice the processing power compared to a cluster with four nodes. The speedup under Round-Robin strategy is due to doubled processing power only. FLEX takes advantage of both the doubled memory and processing power and thus demonstrates a superlinear speedup as the cluster size increases from four to eight nodes.

The following Figures 13, 14 show the average miss ratio per node with Round-Robin and FLEX strategies in clusters with four and eight nodes. (We omit miss ratio figures for Days 3 and 4, because the results are very similar to Days 1 and 2).

The miss ratio with Round-Robin strategy doesn't improve as the cluster size increases from four to eight nodes. This again emphasizes that Round-Robin is incapable of taking advantage of doubled overall memory in the cluster. With an eight-node cluster, FLEX shows signif-

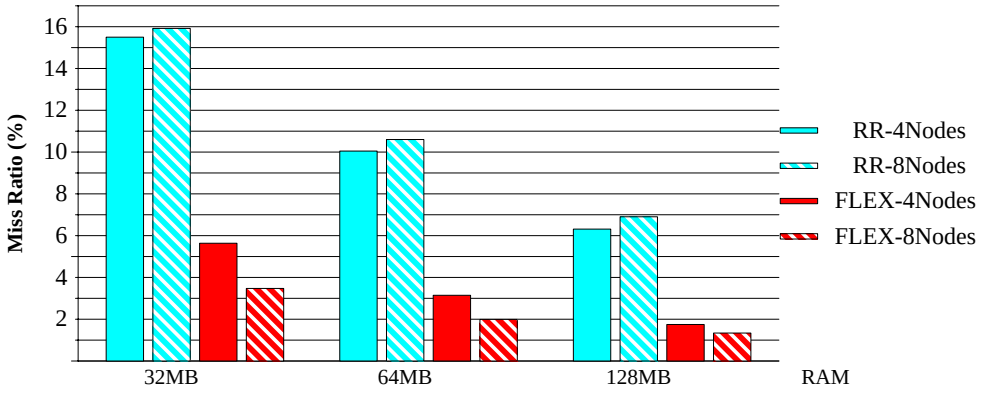

Figure 13: Average Miss Ratio per Server for Four- and Eight-Node Cluster during Day 1.

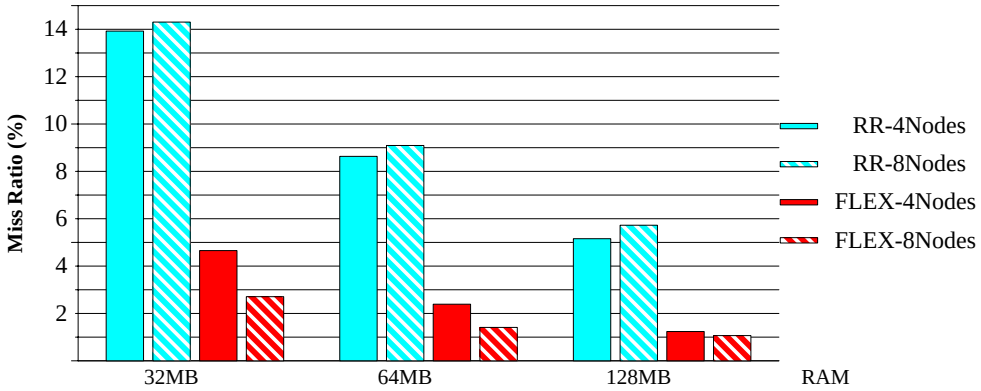

Figure 14: Average Miss Ratio per Server for Four- and Eight-Node Cluster during Day 2.

icantly lower miss ratio, especially for smaller RAM sizes. It explains why FLEX achieves superlinear speedup for larger cluster sizes.

Note that the simulation results above are based on an "ideal Round-Robin" strategy, i.e., we route each consecutive request from the trace to the next web server in a cluster in an ideal round-robin manner. However, in reality once a particular client resolves the domain name to an ip-address of some server in a cluster, all consecutive requests from this client are sent to the same server. The HP site (which we used in our case study) runs on four web servers with RR-DNS for load balancing. We had four traces from those servers ("real Round-Robin") to compare against the "ideal Round-Robin" strategy we used in experiment. Average server throughput under "real Round-Robin" was about 5%-13% worse than under "ideal Round-Robin". These results could be interpreted twofold:

1. "Real Round-Robin" does remarkably good job of load-balancing: it is within 5%-13% of intended load balancing which is an "ideal Round-Robin".

2. The FLEX performance benefits, if compared against "real Round-Robin", will be about 5%-13% higher.

We also did some experimentation with synthetic traces. The performance gain provided by FLEX depends on locality characteristics of the traffic. Preliminary analysis [C99] based on synthetic traces with less locality demonstrated that FLEX can outperform Round-Robin by a factor of 5 for a 16 node cluster.

6 Conclusions and Future Research

In this paper, we analyzed several commercial hardware/software load-balancing solutions used to distribute requests among a group of Web servers. We described a low cost, easy to deploy and scalable solution FLEX. The advantages of the FLEX can be summarized as follows:

- FLEX is a low cost balancing solution unlike most other solutions. It does not require installation of any additional hardware or complex software. The resources spent for additional hardware may be spent in increasing the cluster size thus providing better performance than special hardware based solutions.
- FLEX is a scalable solution because there is no front end switch or a centralized component that could become a bottleneck as the cluster size increases. All the other solutions examined earlier that did dynamic load balancing have centralized routing/ switching components that can become a bottleneck as the cluster size increases.
- FLEX is not based on static inflexible partitioning and can adapt to gradual changes in sites' traffic patterns.
- As emphasized earlier, it is easy to deploy.

Interesting future work will be to make the solution more dynamically adaptable, to extend the solution and the algorithm to work with heterogeneous nodes in a cluster, to take into account SLA (Service Level Agreement) and some additional QoS requirements.

7 References

[Alteon] URL: http://www.alteon.com/products/acelerate-data.html

[BrightTiger] URL: http://www.brighttiger.com

[Bruno97] L. Bruno: Balancing the Load On Web Servers.
CMPnet, September 21, 1997. URL: http://www.data.com/roundups/balance.html

[C99] L. Cherkasova: FLEX: Design and Management Strategy for Scalable Web Hosting Service. HP Laboratories Report No. HPL-1999-64R1,1999.

[CP2000] L. Cherkasova, S. Ponnekanti: Achieving Load Balancing and Efficient Memory Usage in A Web Hosting Service Cluster. HP Laboratories Report No. HPL-2000-27, February, 2000.

[Cisco] URL: http://www.cisco.com/warp/public/751/lodir/

[F5Labs] URL: http://www.f5labs.com/

[FlyingFox] URL: http://www.flyingfox.com/

[HydraWEB] URL: http://www.hydraweb.com/z2_index.html

[IBM-SWND] http://www.software.ibm.com/network/dispatcher/

[LARD98] V.Pai, M.Aron, G.Banga, M.Svendsen, P.Drushel, W. Zwaenepoel, E.Nahum: Locality-Aware Request Distribution in Cluster-Based Network Servers. In Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS VIII), ACM SIGPLAN,1998, pp.205-216.

[MS97] S.Manley and M.Seltzer: Web Facts and Fantasy. Proceedings of USENIX Symposium on Internet Technologies and Systems, 1997, pp.125-133.

[NSCA96] D. Dias, W. Kish, R. Mukherjee, R. Tewari: A Scalable and Highly Available Web Server. Proceedings of COMPCON'96, Santa Clara, 1996, pp.85-92.

[Resonate] URL: http://www.resonate.com/products/

[RND] URL: http://www.rndnetworks.com/

[Roberts98] E. Roberts: Load Balancing: On a Different Track. CMPnet, May 1998. URL: http://www.data.com/roundups/load.html

[RRDNS95] T. Brisco: DNS Support for Load Balancing.
RFC 1794, Rutgers University, April 1995.

[Schwetman95] Schwetman, H. Object-oriented simulation modeling with C++/CSIM. In Proceedings of 1995 Winter Simulation Conference, Washington, D.C., pp.529-533, 1995.

[SpecWeb96] The Workload for the SPECweb96 Benchmark.
URL http://www.specbench.org/osg/web96/workload.html