Performance Analysis of "Content-Aware" Load Balancing Strategy FLEX: Two Case Studies

Ludmila Cherkasova, Mohan DeSouza*, Shankar Ponnekanti †
Hewlett-Packard Laboratories
1501 Page Mill Road,Palo Alto, CA 94303
e-mail: {cherkasova,mdesouza,shankar}@hpl.hp.com

Abstract *FLEX is a new cost effective, "locality aware" load balancing solution for a shared web hosting service implemented on a cluster of machines [3]. 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.*

Using a simulation 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 110-250% in average server throughput), getting within 8%-30% of optimal performance achievable for those traces. 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.

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.

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 (that is the RAM required to store all cached files without replication) is much less than the combined RAM of all the machines.

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, and static partitioning does not accommodate for this.

The observations above have led to the design of "locality-aware" balancing strategies [6] 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, "content-based" solution FLEX for load balancing and management of a shared Web hosting service [3]. 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 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 com-

*Work has been done while M.DeSouza worked at HP Labs during the summer of 1999. His current address is University of California, Dept of Computer Science, Riverside, CA 92521; mdesouza@cs.ucr.edu

†This work has been done while S.Ponnekanti worked at HPLabs during the summer 1999. His current address is Stanford University, Dept of Computer Science, Stanford, CA 94305, USA; pshankar@cs.stanford.edu

pare 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 250% in average server throughput), getting within 8%-30% of optimal performance achievable for those traces. 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 survey of existing load balancing 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 [8]. 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 approach;
- IP/TCP/HTTP redirection based approach.

Very good survey of load-balancing solutions can be found in [11], another related performance study on load-balancing topic is [2].

Round-Robin DNS [9] is built into the newer versions of DNS. RR-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, placing the different address first in the list for each successive request. Thus, different clients are mapped to different server nodes in the cluster. RR-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.

There is one, popular misconception about RR-DNS: that once a domain name of a web site (so-called canonical name, or CNAME) was resolved to a particular IP address (from the list) by the original DNS server, this domain_name-to-IP_address mapping might be cached by some intermediate DNS servers, and this particular domain_name-to-IP_address mapping will be provided for all the consequent requests served by this intermediate DNS. In fact, the mapping returned by the original DNS server is a list of IP addresses in the cluster. Client browser typically chooses the first IP address from the list. Thus, the intermediate DNS server caches not a domain_name-to-IP_address mapping, but rather a domain_name-to-list_of_IP_addresses mapping. This list is rotated for consequent resolutions by the intermediate DNS server, and it helps to avoid skewed load from the different domains to the original cluster.

To outline IP/TCP/HTTP redirection based approaches, we adopt here the classification proposed in [11]. These solutions can be classified in the following groups:

- layer four switching with layer two packet forwarding (L4/2);
- layer four switching with layer three packet forwarding (L4/3);
- layer seven switching (L7).

These terms refer to the techniques by which the systems in the cluster are configured together. In each of the technologies discussed here, the load-balancer sits on the network and acts as a proxy for incoming connections. The load-balancer is configured with a particular IP-address, called the cluster address.

In a L4/2 cluster, all the machines are identical above open systems integration (OSI) layer two (data link). That is, each system has a unique layer two address (i.e. MAC address), but identical layer three network address (cluster IP address). The traffic flow in L4/2 clustered environment is shown in Figure 1. The load-balancer determines the least loaded server (this decision is the job of the proprietary algorithms implemented in different products) to which server in a cluster the packet has to be sent (say, for example, to a server i). Then load-balancer sends the packet to the server i by changing the MAC address of the packet to MAC address of the server i. Since the network address (IP-address) of the server to which the packet was delivered is identical to the IP-address of the load-balancer which was used originally in the request packet, the server currently handling that connection may respond directly to the client rather than through the load-balancer. A restriction of L4/2

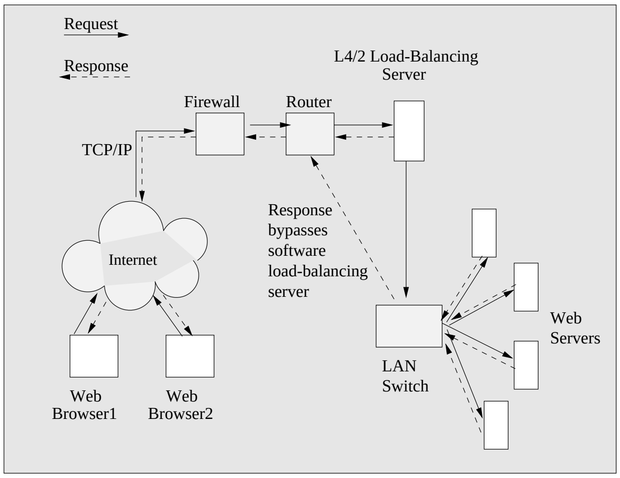

Figure 1: L4/2 Load-Balancing Cluster Solution.

clustering is that load-balancer must have a direct physical connection to all network segments of the servers in the cluster, because of layer two frame addressing. This contrasts with L4/3 clustering where the servers may be anywhere on any network (but in practice, servers in a cluster are likely to be connected via a single high-speed LAN anyway).

Among research and commercial products implementing L4/2 clustering are ONE-IP developed at Bell Labs, IBM's eNetwork Dispatcher, and ACEdirector from Alteon (currently acquired by Nortel).

L4/3 clustering technology slightly predate L4/2 approach. Each server in a cluster has a unique IP-address. 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. Thus an L4/3 load-balancer appears as a single host to a client. To the machines in the cluster, the load-balancer appears as a gateway. The traffic flow in L4/3 clustered environment is shown in Figure 2. Similar to L4/2 approach, the load-balancer determines to which server in a cluster the packet has to be sent (say, for example, to a server i). However, unlike in the earlier approach, the destination IP-address of the packet is rewritten to use the address of the i server which was selected to service this request. The responses sent from the servers in the cluster to the clients must travel through the load-balancer where the IP-address of i server is rewritten back to the original virtual cluster address represented by the load-balancer. Cisco's

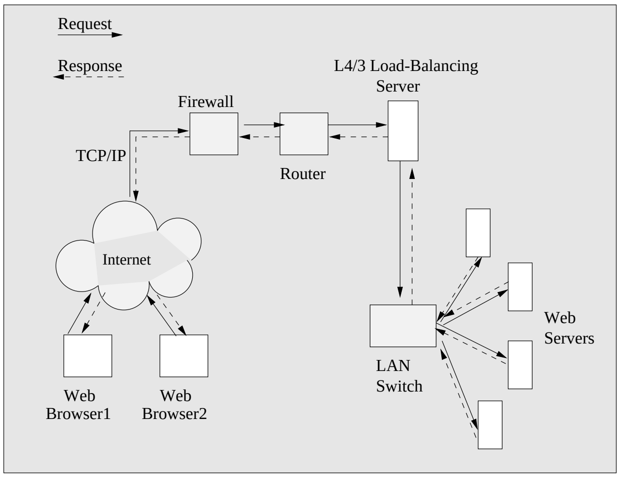

Figure 2: L4/3 Load-Balancing Cluster Solution. In this Figure, LAN switch connecting web servers in a cluster is not generally required.

LocalDirector is the most well-known commercial representative implementing L4/3 approach.

Traditional load balancing solutions (both L4/2 and L4/3) 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" (or content-based) balancing strategies [6] implemented as a L7 layer switching.

In the LARD approach, the cluster nodes are partitioned into two sets: front end and back ends. Front end accepts the connection as well as the request itself. The appropriate back-end server is selected based on the content of the request as well as the load information. The request distribution is done with the aid of modified kernel that supports a connection handoff protocol: after the connection has been established, and the server chosen, the LARD front-end informs the chosen back-end server of the status of the network connection, and the back-end server takes over that connection, communicating response back directly to the client. The authors in [6] show that on workloads with working sets that do not fit in a single node's RAM, LARD improves through-

[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.

put by a factor of two to four for a 16 node cluster (for the traces they experimented with).

3 FLEX Solution

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.

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 [5], we presented heuristic algorithms for solving these problems. For the details, we refer the readers to [5].

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 [2] 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. In the case of FLEX, TTL of one hour is reasonable, since FLEX tuning is done on a daily or weekly basis. Such a self-monitoring solution helps in observing changing site traffic patterns and helps predict future trends and plan for them.

4 First Case Study

4.1 Access Log and Sites Profile

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. Understanding the "one-timers" memory requirements is useful for "at-a-glance" capacity planning of the cluster RAM.

Month	Number of Requests	WS (MB)	AR MB	WS of Onetimers
April	1,674,215	994.2	14,860	370.0 MB
May	1,695,774	878.4	14,658	374.5 MB
June	1,805,762	884.9	13,909	311.2 MB
July	1,315,685	711.6	8,713	298.3 MB

(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)
April	242.7 MB / 362.1 MB / 556.3 MB
May	249.2 MB / 296.3 MB / 419.9 MB
June	196.1 MB / 304.8 MB / 475.1 MB
July	155.1 MB / 276.1 MB / 487.9 MB

(2)

Month	Working Set for 97/98/99% of all Requests (as % of Total WS)
April	24.4% / 36.4% / 56.0%
May	28.4% / 33.7% / 47.8%
June	22.2% / 34.4% / 53.7%
July	21.8% / 38.8% / 68.6%

(3)

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 4, 5, 6, 7 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 N^{00}	Largest WS	AR	Site N^{00}	WS	Largest AR
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

(4)

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.

[2]The detailed description of the "Closest" algorithm can be found in our HPLabs report HPL-2000-27, February, 2000. URL http://www.hpl.hp.com/techreports/2000/HPL-2000-27.html". It is a full version of [5].

May: Web Hosting Service had 74 hosted sites

Site N^{00}	Largest WS	AR	Site N^{00}	WS	Largest AR
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

(5)

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 N^{00}	Largest WS	AR	Site N^{00}	WS	Largest AR
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

(6)

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 N^{00}	Largest WS	AR	Site N^{00}	WS	Largest AR
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

(7)

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).

4.2 Simulation Model

Our simulation model was written using C++Sim [10]. Each web server consists of CPU and locally attached disk(s), with separate queues for each. In addition, each node maintains its own main memory cache (of configurable size) and replacement policy. For simplicity, caching is performed on a whole-file basis.

Processing a request requires the following steps: connection establishment, file transfer from the disk (if needed), file transfer to network, and connection teardown. The individual processing steps for a given request must be performed in sequence, but the CPU and disk times for different requests can be overlapped. Large file reads from the disk are blocked, such that the data transmission immediately follows the disk read for each block. Multiple requests waiting on the same file from disk can be satisfied with only one disk read, since all the requests can access the data once it is cached in memory.

For the cost of the basic request processing steps in our simulation, we decided to use the same parameters as the authors of LARD [6] used for their simulation. They derived these parameters by performing measurements on 300Mhz Pentium II machine running Free BSD 2.2.5 and aggressive experimental web server [3].

Connection establishment and tear down costs are set to 145 μ *sec* of CPU time each, while transmit processing incurs 40 μ *sec* per 512 bytes. (accordingly to these assumptions, the 8KByte document can be served from the main memory cache at a rate of approximately 1075 requests/sec).

If disk access is required, reading a file from the disk has a latency of 28ms (2seeks and rotational latency). The disk transfer time is 410 μ *sec* per 4 KByte (which results in approximately 10MBytes/sec paek transfer rate). For files larger than 44KBytes, an additional 14ms (seek plus rotational latency) is charged for each 44KBytes.

The cache replacement policy is LRU.

4.3 Simulation Results

Using our partitioning algorithm, a partition was generated for each month. [4] 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 in round-robin manner assigned among the servers designated 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.

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"

[3] We also performed similar experiments with a slightly different model and parameters, based on measurements of a web server running on HP 9000/899 running HP-UX 11.00. The performance results are quite similar to the results which use LARD model and parameters, and lead us to believe that the crucial assumption is a difference in web server throughput when it retrieves the files from the disk rather than from the RAM. The corresponding simulation results be found in [4]and [5].

[4] In this work, we used *Simple* strategy to define working set and access rate for "replicated" sites. For details see [5].

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.

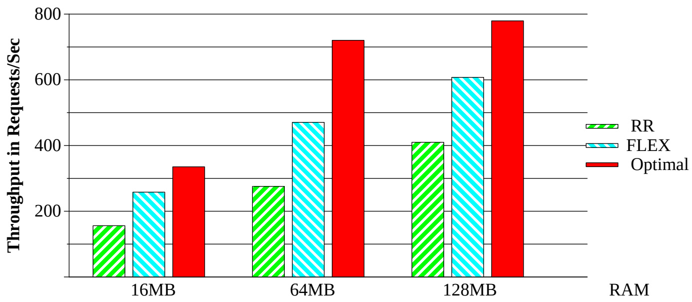

Figure 3: Average Server Throughput in a Four-Node Cluster for April.

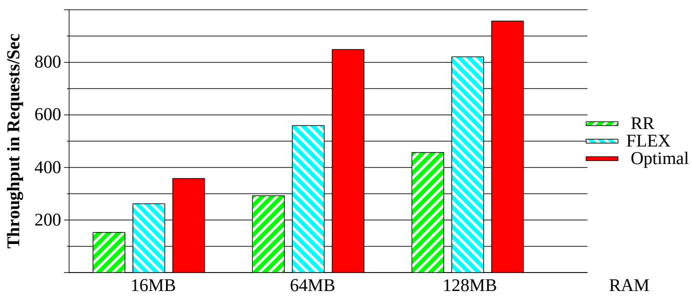

Figure 4: Average Server Throughput in a Four-Node Cluster for May.

Figures 3, 4, 5, 6 show the average server throughput for Round-Robin, FLEX and *Optimal* and different RAM sizes (the RAM sizes shown are per server). Note that

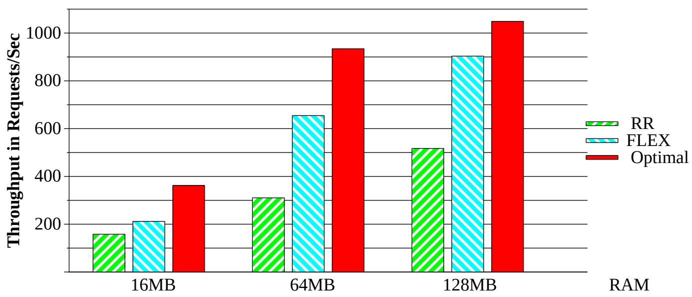

Figure 5: Average Server Throughput in a Four-Node Cluster for June.

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,

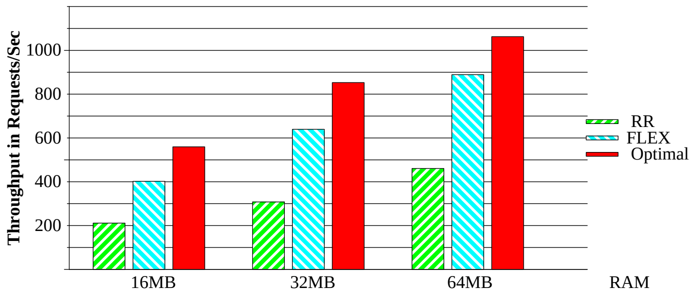

Figure 6: Average Server Throughput in a Four-Node Cluster for July.

FLEX performs best when there are no large sites and each site is allocated to exactly one server. In April, May and June one or more sites (site 62 in all three months and 57 in April) have to be necessarily replicated to achieve balanced partitions. No site had to be replicated in July. In general, FLEX outperforms Round-Robin in all cases, and for larger RAM, the benefits in throughput range from 50% to 110%. In general, performance of the FLEX is within 14-30% of *Optimal*.

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 Second Case Study

5.1 Access Log and Sites Profile

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.

Month	Number of Requests	WS (MB)	AR MB	WS of Onetimers
Day 1	6,893,537	1,249	17,774	381.6 MB
Day 2	6,709,091	1,073	17,056	359.2 MB
Day 3	6,822,680	1,177	17,356	404.4 MB
Day 4	6,591,683	1,243	17,080	390.9 MB

(8)

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)
Day 1	213.4 MB / 341.3 MB / 595.0 MB
Day 2	175.5 MB / 265.5 MB / 420.9 MB
Day 3	216.8 MB / 322.3 MB / 541.3 MB
Day 4	204.3 MB / 292.3 MB / 532.5 MB

(9)

Month	Working Set for 97/98/99% of all Requests (as % of Total WS)
Day 1	17.1% / 27.3% / 47.6%
Day 2	16.4% / 24.8% / 39.2%
Day 3	18.4% / 27.4% / 46.0%
Day 4	16.4% / 23.5% / 42.8%

(10)

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. The goal of the second case study is 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).

5.2 Simulation Results

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 11 show five sites with the largest working sets and with the largest access rate for Day 1.

Site N^{00}	Largest WS	AR	Site N^{00}	WS	Largest AR
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

(11)

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 11 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 7, 8 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).

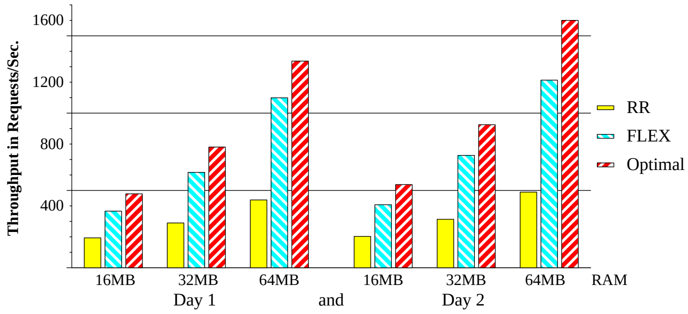

Figure 7: Average Server Throughput in a Four-Node Cluster for Days 1 and 2.

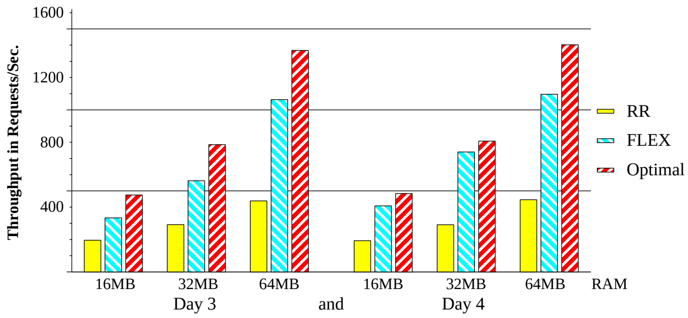

Figure 8: Average Server Throughput in a Four-Node Cluster for Days 3 and 4.

The results are consistent across all the four days. The performance improvements due to FLEX over Round-Robin are 90%-150% depending on the server RAM size (larger RAM gives greater performance improvements, because FLEX benefits from more efficient memory usage). FLEX is within 8%-25% of the *Optimal* strategy for all RAM sizes for the given traces.

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 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)

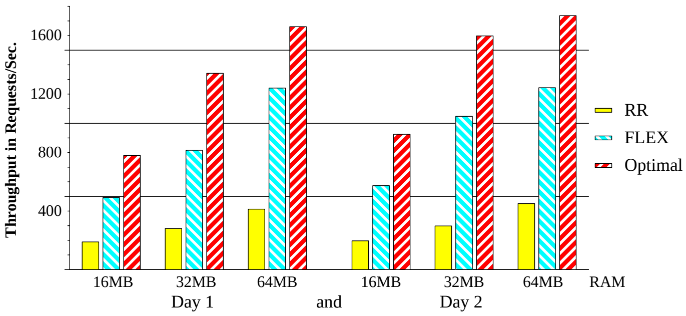

Figure 9: Average Server Throughput in an Eight-Node Cluster for Days 1 and 2.

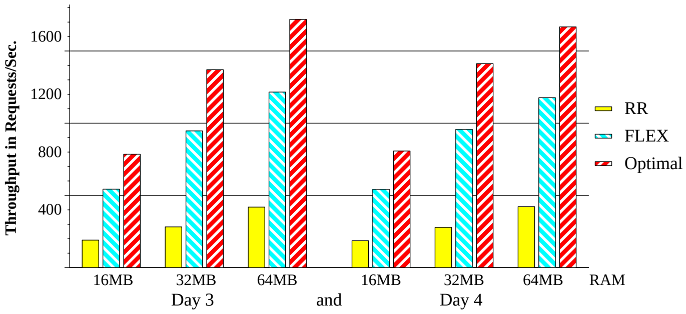

Figure 10: Average Server Throughput in an Eight-Node Cluster for Days 3 and 4.

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 160%-250% higher compared to the average web server throughput with Round-Robin strategy. FLEX is within 30% of the *Optimal* strategy for the given traces for various RAM sizes.

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 11, 12 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).

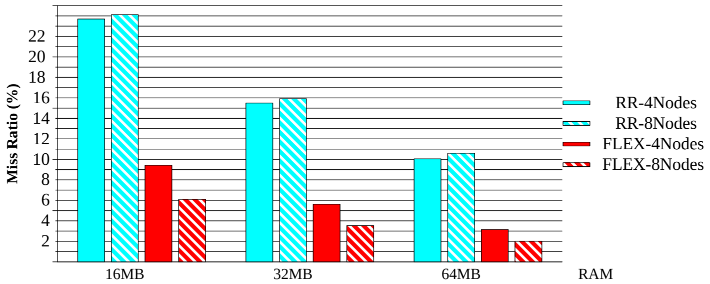

Figure 11: Average Miss Ratio per Server for Four- and Eight-Node Cluster during Day 1.

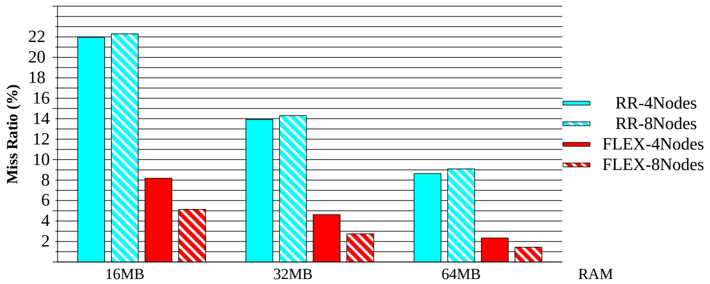

Figure 12: Average Miss Ratio per Server for Four- and Eight-Node Cluster during Day 2.

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 significantly 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, because client browser caches this ip-address for at least 15 min. HP site (which we used in our case study) runs on multiple web servers with RR-DNS for load balancing. We had four traces from four 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 [3] 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

Using two case studies (based on real traces), we evaluated the potential benefits of the new load-balancing solution FLEX. 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. FLEX solution shows superliner speedup when the number of nodes is increased (from four to eight in our case study) taking advantage of both the doubled processing power and memory.

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.

- 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.

We can see an interesting future work in extending the solution to be able to deal with dynamic pages.

Another direction is 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 some additional QoS requirements.

References

[1] M. Aron, D. Sanders, P. Druschel, W. Zwaenepoel. Scalable Content-Aware Request Distribution in Cluster-based Network Servers. In Proceedings of the USENIX 2000 Annual Technical Conference, San Diego, CA, June 2000.

[2] Bryhni H., Klovning E., Kure O. A comparison of load balancing techniques for scalable Web servers. IEEE Network. 14(4): 58-64, 2000 Jul-Aug.

[3] L. Cherkasova: FLEX: Load Balancing and Management Strategy for Scalable Web Hosting Service. In Proceedings of the Fifth International Symposium on Computers and Communications (ISCC'00), Antibes, France, July 3-7, 2000.

[4] L. Cherkasova, M. DeSouza, S. Ponnekanti: Performance Analysis of Scalable Web Hosting Service with FLEX: Two Case Studies. HP Laboratories Report No. HPL-2000-28, February, 2000.

[5] L. Cherkasova, S. Ponnekanti: Optimizing a "Content-Aware" Load Balancing Strategy for Shared Web Hosting Service. In Proceedings of the Eighth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'2000).

[6] 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.

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

[8] 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.

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

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

[11] T. Schroeder, S. Goddard, B. Ramamurthy, "Scalable Web Server Clustering Technologies", IEEE Network, May/June 2000.