Optimizing a "Content-Aware" Load Balancing Strategy for Shared Web Hosting Service

Ludmila Cherkasova
Hewlett-Packard Laboratories
1501 Page Mill Road,
Palo Alto, CA 94303
cherkasova@hpl.hp.com

Shankar R Ponnekanti *
Stanford University,
Dept of Computer Science
Stanford, CA 94305, USA
pshankar@cs.stanford.edu

Abstract *FLEX is a new scalable "locality aware" solution for achieving both load balancing and efficient memory usage on a cluster of machines hosting several web sites [1, 2]. FLEX allocates the sites to different machines in the cluster based on their traffic characteristics. Here, we propose a set of new methods and algorithms (Simple+, Advanced, and Advanced+) to improve the allocation of the web sites to different machines. New methods show additional performance benefits compared to the original Simple strategy. Experiments show that FLEX outperforms traditional load balancing solutions by 50% to 100% (in throughput) even for a four node cluster. Miss ratio is improved 2-3 times.*

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 to the Internet, disks, processors, memory, etc.) offer to store and provide Web access to documents for institutions, companies and individuals looking for a cost efficient, "no hassle" solution.

A shared Web hosting service creates a set of virtual servers on the same server. This creates 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 to create scalable and highly available solutions. In this paper, we assume that each web server in a cluster has access to all the web content. Therefore, any server can satisfy any client request.

Traditional load balancing solutions (both hardware and software) for a web server cluster try to distribute the requests uniformly on all the machines. However, this adversely affects efficient memory usage because content is replicated across the caches of all the machines. With this approach, a cluster having N times bigger RAM (which is the combined RAM of N nodes) might effectively have almost the same RAM as one node due to replicated popular content throughout the RAMs in the cluster.[1]

A better approach is to partition the content so that memory is used more efficiently. However, static partitioning will inevitably lead to an inefficient, suboptimal and inflexible solution, since the changes in access rates as well as access patterns tend to vary dramatically over time, and static partitioning does not accommodate for this.

The observations above led researchers to design "locality-aware" balancing strategies like LARD (*locality aware request distribution*) [4] which aim to avoid unnecessary document replication to improve the overall performance of the system.

In [1, 2], FLEX - a new scalable "locality aware" solution for load balancing and management of an efficient Web hosting service, was introduced. 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.

The main attractions of this approach are econ-

*This work was done while Shankar Ponnekanti was working at HPLabs in summer 1999.

[1] We are interested in the case when the overall file set is greater than the RAM 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.

omy and ease of deployment. This solution requires no special hardware support or protocol changes. The resources required for special hardware based solution can instead be invested in adding more machines to the cluster. 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. Further, most web hosting systems have built in facilities for web server log analysis for various other reasons. Our solution can thus be integrated easily into the current infrastructure.

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 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. Several questions have to be answered when dealing with large sites that need to be replicated: how many servers should be assigned to a particular large site? What are the memory requirements and the load due to this site on each of the assigned servers? Evaluation of the memory requirements for the replicated site on each of the assigned servers is a non-trivial task. In Section 2, we propose a set of methods and algorithms (of varying complexity and expected accuracy) to solve this problem. Section 3 introduces workload we used to compare performance of the designed algorithms and strategies. Section 4 describes the simulation model and its parameters. Section 5 analyzes the simulation results for the proposed methods.

2 Working Set and Access Rate Evaluation Methods

We use $W(s)$ to denote the *working set* of site s, which stands for the memory requirements of site s. Similarly, we use $A(s)$ to denote the *access rate* of site s, which stands for the load due to site s. Let N be the number of nodes in the web server cluster. [1] Our goal is to partition all the hosted web sites in N "equally balanced" groups: $C_1, ..., C_N$ such that the cumulative access rates and cumulative working sets of these N groups are approximately the same.

It is important to note that *access rate* is used in this paper to denote the load, not the number of bytes accessed in a period. The latter is called *raw access rate* and is a well defined quantity. On the other hand, each of the methods to be proposed in this section calculates the *access rate* or load differently. A similar remark applies to *working set*.

[1] We assume that all the machines are identical. However, our approach can be extended to the case where machines have different capacities.

2.1 Replication of Large Sites

An important issue that needs to be addressed is dealing with large sites. A large site has to be served by more than one server when a single server does not have enough resources to handle all the requests to this site. When a site s is replicated on k servers, we replace the site s by k identical logical sites $s\#k$, where each of these sites is assigned to a different server. Note that $s\#1$ is the same as s.

When a site is replicated on k servers, the working set of this site on each of these k servers may not be the same as $W(s)$. In fact, we expect the working set on each of the servers to be less than the working set of the unreplicated site $W(s)$, because some files of this site might never be accessed on some of these k servers. Similarly, the access rate to this site on each of the k servers is smaller too. We denote the working set (and access rate) on each of the k servers of a site s replicated on these k servers by $W(s\#k)$ (and respectively $A(s\#k)$). Thus, the total working set and access rate of a replicated site s on all the k servers is given by $k * W(s\#k)$ and $k * A(s\#k)$.

The new working set and the new access rate for each site s are thus defined defined as

$$NewW(s) = \begin{cases} W(s) & \text{if the site is put on one server} \\ k * W(s\#k) & \text{if the site is put on } k > 1 \text{ servers} \end{cases}$$

Similarly,

$$NewA(s) = \begin{cases} A(s) & \text{if the site is put on one server} \\ k * A(s\#k) & \text{if the site is put on } k > 1 \text{ servers} \end{cases}$$

The total working set and the total access rate of all the sites is computed as follows:

$$TotalW = \sum_{\text{all sites } s} NewW(s)$$

$$TotalA = \sum_{\text{all sites } s} NewA(s)$$

Thus, the mean working set and the mean access rate per server are given by:

$$MeanW = \frac{TotalW}{N} \qquad \text{and} \qquad Mean = \frac{TotalA}{N}$$

where N is the number of servers in the cluster.

Next, we describe when and on how many servers is a site replicated. We replicate a site s when its working set $W(s)$ or access rate $A(s)$ exceeds a certain limit: $W(s) > alpha * MeanW$ or $A(s) > beta * MeanA$, where *alpha* and *beta* are two thresholds in the range between 0 and 1. Typical values of *alpha* and *beta* to create a good balanced solution are in the range of 0.7.

Our algorithm for deciding the amount of replication for each site s is as follows. Let $Copies(s)$ denote the number of times a site is replicated. Initially, we have $Copies(s) = 1$ for all the sites s.

```
find MeanW and MeanA
do
  done = true
  for s = 1 to NumSites
    if ((W(s#Copies(s))>alpha*MeanW or
       A(s#Copies(s))>beta*MeanA)  and
       Copies(s)<N)  {
         Copies(s) = Copies(s) + 1;
         done = false;
         recompute NewW(s), NewA(s);
         recompute MeanW,  MeanA;
    }
while done = false
```

Note that when the algorithm is finished, the following condition is met: for each site s, either s is replicated across all the N servers or

$$W(s\#Copies(s)) < alpha * MeanW$$

and

$$A(s\#Copies(s)) < beta * MeanA.$$

To simplify our algorithms and get a better representation of the working sets and access rates for each site, we "normalize" the working sets and access rates as given below.

If a site s is replicated across k servers, we set

$$W(s\#k) = \frac{W(s\#k) * N * 100}{TotalW}.$$

Similarly,

$$A(s\#k) = \frac{A(s\#k) * N * 100}{TotalA}.$$

These formulae also hold for unreplicated sites with $k = 1$ (recall that $s\#1$ is the same as s). With the normalization, both the total access rate and the total working set of all the sites is equal to $N * 100$. Thus, our task is to partition the web sites in N balanced groups with cumulative (normalized) working sets and access rates of 100 units each. Each of these balanced group is then assigned to a server.

2.2 Overview of Methods

All the methods proposed are based only on information that can be extracted from the web server access logs of the sites. For each hosted web site s, we build the initial "site profile" by evaluating the following characteristics:

- $RA(s)$ - the *raw access rate* to the content of a site s (in *bytes* transferred during the observed period P);
- $RW(s)$ - the combined size of all the accessed files of site s (in *bytes* during the observed period P, called *raw working set*);
- $FR(s)$ - the table of all accessed files with their frequency (number of times a file was accessed during the observed period P) and size, i.e., a set of triples $(f, fr_f, size_f)$, where fr_f and $size_f$ are the frequency and the size of the file f respectively.

Unless specified, our methods are independent of what the period P is (one day, one month etc).

We propose four different methods to estimate $W(s)$ and $A(s)$. Recall that $W(s)$ stands for the memory requirements and $A(s)$ stands for load. There are several ways of estimating memory requirements and load of a site. Each method uses a different definition of $W(s)$ and $A(s)$. The goal of each method is to compute the memory and load requirements, but the methods differ in complexity of modeling and expected accuracy of estimates.

- *Simple* is the simplest method and is based on the **sites' raw working set and raw access rate** extracted from server logs. *Simple* method was proposed and used in [2].
- *Simple+* method requires **the files' access frequency information** in addition to the raw working sets and raw access rates. Based on this additional knowledge, we estimate the possible reduction of the raw working sets caused by replication.
- *Advanced* method calculates the working set of a site as the number of bytes of that site that are expected to be in the RAM assuming that **the memory (RAM) size of each node in the cluster is available.** Similarly, the access rate of a site is calculated as a weighted sum of the site's bytes accessed from memory and transferred from disk.

- *Advanced+* method is similar to *Advanced* method, but it has a more realistic and complicated analytical model.

2.3 Simple

The simplest but least accurate method is to set:

$$A(s\#k) = \frac{RA(s)}{k} \qquad \text{and} \qquad W(s\#k) = RW(s)$$

where the k is the number of servers the site s is replicated across. The formulae hold for unreplicated sites also, with $k = 1$. Thus, this method simply estimates load as the raw access rate and memory requirements as the sum of sizes of all the accessed files of the site. For a replicated site, the load on each of the k servers is set to $\frac{1}{k}$-th of the total load to s. However, memory requirements are unchanged.

2.4 Simple+

This method is the same as *Simple* except for replicated sites. Here, we estimate the possible reduction of the working sets (or memory requirements) caused by replication. Intuitively, if some files of the site s are accessed only a few times, then the probability that these files are accessed on all the k servers the site s is replicated across diminishes as k increases. This leads to a smaller working set on each of the k servers.

In this method, we use **additional information - the frequency for all accessed files**, for each of the replicated sites s. Let a site s be replicated across k servers. In order to estimate $W(s\#k)$, we evaluate the probability $p(k, f)$ that the file f is accessed at least once on one of these k servers in the period P. Assuming independence of accesses, this probability is given by

$$p(k, f) = 1 - (1 - \frac{1}{k})^{fr_f}.$$

Thus, we calculate the working set as

$$W(s\#k) = \sum_{\text{all files } f \text{ of site } s} [1 - (1 - \frac{1}{k})^{fr_f}] * size_f$$

and we calculate the access rate as

$$A(s\#k) = \frac{A(s)}{k}.$$

Thus, we estimate the possible reduction in working set size due to the site's replication. This improves the accuracy of the working set estimates and hence, the expected accuracy of the site allocation method as well.

2.5 Advanced

The *Advanced* method improves upon the *Simple* and *Simple+* methods by using additional information - the memory size of each node. This method makes the following assumptions:

- Best performance is achieved when the most popular bytes (most frequently accessed bytes) reside in memory.
- The OS replacement policy ensures that the most popular bytes are the ones that are kept in memory at all times.

While these assumptions are not strictly true, we believe they are reasonable. Given these assumptions, it can be easily seen that when the sites are distributed on a cluster of identical nodes with total memory Ram, the best performance is achieved when the most popular Ram bytes are distributed equally on all nodes. Thus, the *Advanced* method first identifies the most popular Ram bytes from all sites taken together. This method then defines the working set of each site as its contribution to the most popular Ram bytes.

Let ram be the size of the memory (RAM) of each node. Hence we have, total cluster memory $Ram = ram * N$. Let $B(s, fr)$ be the number of bytes of site s that are accessed with frequency fr in the period P. In other words, $B(s, fr)$ is the sum of sizes of files that are accessed with frequency fr in period P.

Let $C(s, fr) = \sum_{\text{all } fr' \geq fr} B(s, fr')$. We can compute $C(s, fr)$ for all the sites and frequencies. In practice, we compute $C(s, fr)$ only till some frequency limit fr_{large}.

Then, we find the smallest frequency fr^{opt} such that $\sum_{\text{all sites } s} C(s, fr^{opt}) \leq Ram$. Essentially, by computing fr^{opt}, we have identified the most popular Ram bytes from all the sites. We evaluate the working set requirement of a site s as $W(s) = C(s, fr^{opt})$. As explained earlier, this is the site's contribution to the most popular Ram bytes.

According to our assumptions, we have $C(s, fr^{opt})$ bytes of site s in memory while the rest of the bytes of this site come from disk. Thus, the number of bytes of site s that would come from disk is

$$D(s, fr^{opt}) = \sum_{fr < fr^{opt}} fr * B(s, fr).$$

In this method, we make a distinction in estimating the site load depending on whether the accessed bytes come from memory or disk. Accesses from the disk cause more load on the system that accesses from memory. So, we weigh the accesses from the disk by a factor *DiskWeight*. In other words, for the accesses coming from the disk, we add an additional cost of $(DiskWeight - 1)$ per byte.

Thus, we set $A(s) = RA(s) + D(s, fr^{opt}) * (DiskWeight - 1)$. Recall that $RA(s)$ is the raw access rate of site s. Note that $A(s) = RA(s)$ when $DiskWeight = 1$.

If a site s is replicated on k servers, we approximate the number of bytes (of s) accessed with frequency greater than or equal to fr on each server as follows:

$$C(s\#k, fr) = C(s, k * fr)$$

and we have

$$B(s\#k, fr) = C(s\#k, fr) - C(s\#k, fr + 1).$$

We use the above equations to calculate fr^{opt}, $D(s, fr^{opt})$ and the working sets and access rates for all the sites and the sites' replicas.

2.6 Advanced+

In *Advanced*, we make the simplifying assumption that the OS replacement policy is such that it always keeps the most popular *ram* bytes in memory. In *Advanced+*, we make a more realistic assumption - OS has LRU replacement policy. We retain the assumption that for best performance, the most popular *Ram* bytes have to be distributed equally among all nodes. We design a simple analytical model to calculate a time period T such that the sum of sizes of distinct files accessed (from all the sites) in time T is equal to *Ram* bytes. In other words, this is the period for one LRU cycle if we had a single machine with *Ram* bytes of memory serving all the requests. Since the most popular *Ram* bytes are distributed equally on all the nodes, we assume that T is also the period for one LRU cycle for each node. That is, a file that is accessed at time t is expected to be evicted at time $t + T$ if it is not accessed again.

In order to calculate T, we assume that the arrival distribution is Poisson. That is, for the $B(s, fr)$ bytes of site s that were accessed with frequency fr, we assume that their arrival rate is Poisson with fr expected arrivals in period P. So, we have

$$\sum_{s,fr} B(s, fr) * (1 - e^{-\frac{fr*T}{P}}) = Ram.$$

Using the above formula , we can find T/P.

The bytes of a site s that are expected to be in the RAM at any time t are the the bytes which are accessed atleast once in the interval $(t - T, t)$. Thus, the working set of a site s is defined as the number of bytes of site s that are expected to be accessed atleast once in a period T. Once T is found, we set

$$W(s\#k) = \sum_{fr} B(s, fr) * (1 - e^{-\frac{fr*T}{k*P}})$$

$$A(s\#k) = \frac{RA(s)}{k} + (DiskWeight - 1) * D(s\#k)$$

$$D(s\#k) = \sum_{fr} fr * B(s, fr) * e^{-\frac{fr*T}{k*P}}$$

In other words, when a site s is replicated across k servers, if a byte of site s was accessed an expected $\frac{fr*T}{P}$ times during the interval T, the same byte of each replica site (s#k) is accessed an expected $\frac{fr*T}{k*P}$ times during the interval T. As usual, these formulae also work for unreplicated sites (set k = 1). The formula for $D(s\#k)$ is based on the following reasoning: A file accessed at time t is expected to be available in the RAM if it was accessed atleast once in the interval $(t - T, t)$. If not, it has to be fetched from the disk.

2.7 Algorithm "Closest"

FLEX modifies and adjusts the allocation of sites to the nodes of the cluster when traffic patterns change. FLEX monitors the sites' requirements periodically (at some prescribed time interval: daily, weekly, or monthly). If the sites' requirements change, the old allocation (partition) may not be good anymore and a new allocation (partition) of sites to machines has to be generated. If generating 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, we need to generate a new balanced partition which changes the assigned server for a minimal number of sites from the existing, "old" partition, to avoid temporary performance degradation during the reassignment of sites to servers.

We designed a heuristic approximate algorithm *Closest* (exact algorithm is unlikely to be polynomial as this problem can be proved to be NP-complete) for the following problem:

- create a new balanced partition R for the given sites requirements *Reqs* which is "closest" to a specified "previous" sites' partition R_{old}

where "closest" means that minimal number of sites are moved to new servers.

While the algorithm *Closest* doesn't guarantee the best balancing or the closest partition to the previous partition, it does pretty well in practice.

For a detailed description of the algorithm, we refer the readers to the full version of the paper [3].

3 Workload Description

For our experiments, we used traces from the HP Web Hosting Service (provided to internal customers). We collected 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.

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

(1)

To characterize the "locality" of the traces, we created a table called *Freq-Size*. For each file in the trace, we stored: 1) the number of times the file was accessed (frequency), and 2) the file size in bytes. *Freq-Size* is sorted in the order of decreasing frequency. For various percentages x, we then computed the sum of sizes of the most popular files that accounted for $x\%$ of all the requests. The next two tables show 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, that is, a larger percentage of the requests target a small set of documents.

In our simulations for the HP Web hosting service, we consider 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. Hence, each machine has to be allocated a set of sites with combined access rate and working set of 100 units each. Tables 4, 5, 6, 7 show the five sites with the largest working sets and the five sites with the largest access rate for each of April, May, June, and July.

April:

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) but attract a large number of accesses (40.2 units of access rate). Such sites have a small number of extremely "hot" files.

May:

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 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:

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 leading sites.

July:

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 significant change in the leading sites: sites 10 and 20 became the largest sites with respect to working set and access rate requirements respectively. Site 1 still has a very small working set (0.7 units) but now accounts for 34 units of the load! 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 or access rates.

4 Simulation Model

Our simulation model was written using C++Sim [5]. The model makes the following assumptions about the capacity of each web server in the cluster:

1. Web server throughput is 1000 requests/sec when retrieving files of size 14.5K from the RAM.
2. Web server throughput is 10 times lower (i.e., 100 requests/sec) when it retrieves the files from the disk rather than from the RAM.[2]
3. The service time for a file is proportional to the file size.
4. The cache replacement policy is LRU.

Using our partitioning algorithm [3], 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. 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.

We also implemented the "optimal" strategy *Opt* 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).[3]

[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).

[3] Opt 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 the main memory.

5 Simulation Results

Figure 1 shows the achievable throughput for Round-Robin, different FLEX strategies and *Opt* for different RAM sizes (the RAM sizes shown are per server). Note that when a large site is assigned to multiple

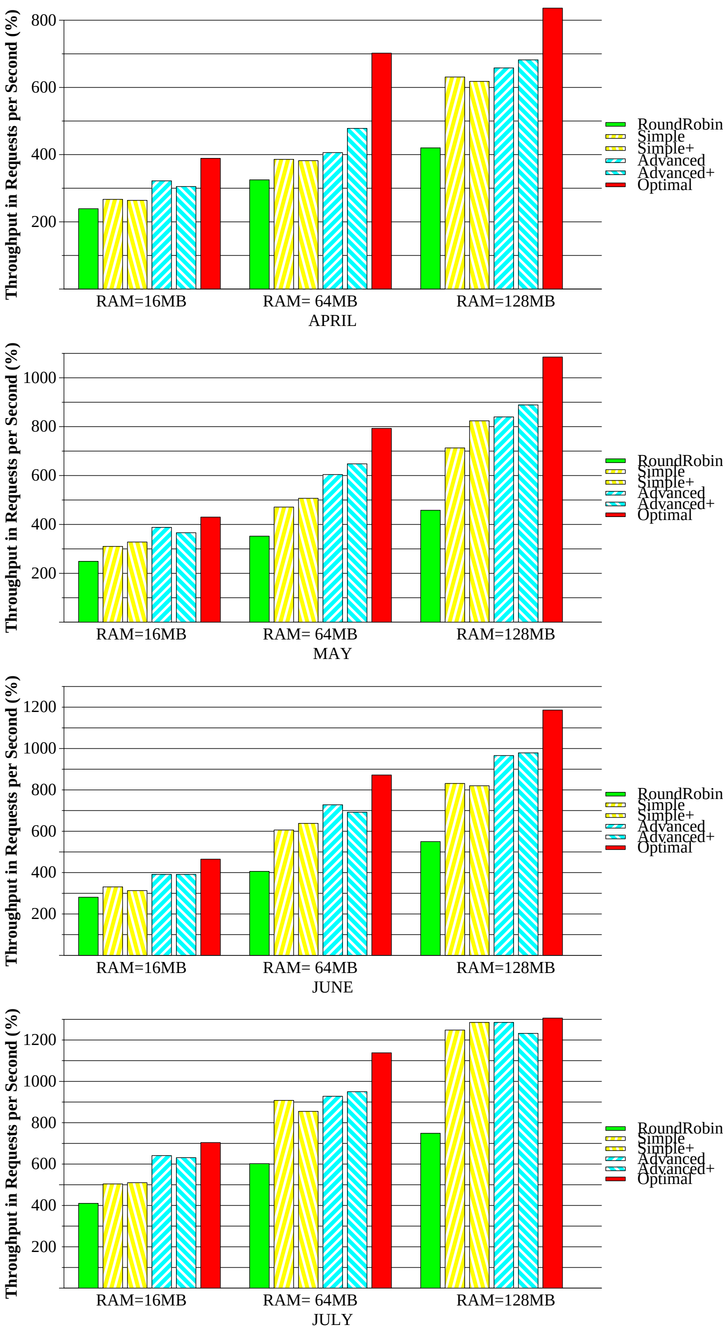

Figure 1: Server Throughput for Round-Robin, different FLEX strategies and *Opt* for April, May, June, and July respectively.

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 *Opt* in April, May and June because one or more sites (sites 62 in all three 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 thus performance of FLEX was much better. In some cases (especially 128 MB) it was near-optimal.

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 5-15% of *Opt*. In all the cases, *Advanced* and *Advanced+* outperform *Simple* and *Simple+*. Difference between *Simple* and *Simple+*, as well as *Advanced* and *Advanced+* does not seem to be significant (at least for the traces we studied). Due to lack of space, we do not include figures for miss ratio. In all cases, miss ratio (FLEX vs Round-Robin) is improved dramatically, by a factor of 2-3. *Opt* shows the minimum miss ratio, and miss ratios for FLEX strategies are very close to that of *Opt*. 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.

6 Conclusions and Future Work

We proposed a set of new methods (Simple+, Advanced, and Advanced+) to evaluate the sites' working sets and access rates. Our simulation results confirm that the new methods significantly improve performance of FLEX in the presence of sites with large working sets and access rates. FLEX is a low cost balancing solution unlike most other solutions. FLEX has the following advantages compared to other *locality aware* strategies:

- Ease of deployment, flexibility, and ability to adapt to gradual changes in sites' traffic patterns, facilitating an efficient capacity planning approach.
- No special hardware support needed.
- No complicated state management/connection handoffs etc.
- No front-end routing component that can become a potential bottleneck as the cluster size increases.

The primary disadvantage of FLEX is its inability to adjust to temporary surges in request arrivals. However, FLEX has an extremely favorable cost-benefit tradeoff. Further, combining the ideas of FLEX with a dynamic load balancing approach could result in a solution that has the advantages of both worlds. This will be part of future work.

References

[1] L. Cherkasova: FLEX: Design and Management Strategy for Scalable Web Hosting Service. HP Laboratories Report No. HPL-1999-64R1,1999. http://www.hpl.hp.com/techreports/1999/HPL-1999-64R1.html.

[2] L. Cherkasova: FLEX: Load Balancing and Management Strategy for Scalable Web Hosting Service. In Proceedings of The Fifth IEEE Symposium on Computers and Communications (ISCC'2000).

[3] 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. http://www.hpl.hp.com/techreports/2000/HPL-2000-27.html

[4] 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.

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