Achieving Load Balancing and Efficient Memory Usage in A Web Hosting Service Cluster

Ludmila Cherkasova, Shankar Ponnekanti*
Computer Systems and Technology Laboratory
HP Laboratories Palo Alto
HPL-2000-27
February, 2000

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

web hosting service, web server cluster, web traffic characteristics, load balancing, scalability, performance analysis.

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 [C99]. 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, Simple+, Advanced, and Advanced+) to improve the allocation of the web sites to different machines. We also design algorithm "Closest" which creates a new partition for the given sites' requirements which is "closest" to a specified "previous" sites' partition. "Improved" FLEX outperforms traditional load balancing solutions 50% to 100% (in throughput) even for a four node cluster. Miss ratio is improved 2-3 times. These figures increase with the size of the cluster. Ease of deployment and low cost are the other attractions of FLEX.

Internal Accession Date Only
* 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 to the Internet, disks, processors, memory, etc.) offer to 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.

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 a Web hosting infrastructure as a way to create scalable and highly available solutions. Good surveys of most typical architectural solutions may be found in [B97, R98, C99]. In this paper, we assume that each web server in a cluster or web server farm has access to all the web content. Therefore, any server can satisfy any client request.

Ideally, a cluster (farm) of N web servers should be N times more powerful than one web server. However, to realize this, one has to achieve the two often conflicting goals: load balancing and efficient memory usage. 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 a 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 [LARD98] which aim to avoid unnecessary document replication to improve the overall performance of the system. In [C99], FLEX - a new scalable "locality aware" solution for the design 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.

One of the main attractions of this approach is economy and ease of deployment. This solution requires no special hardware support or protocol changes. The resources required for special

[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.

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 very 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.

FLEX also modifies and adjusts the allocation of the 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 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.

In this paper, we propose a set of methods and algorithms (of varying complexity and expected accuracy) to solve the following problems that were raised above:

- computing the site's working set and access rate requirements when the site is replicated across several servers;
- creating a new partition for the given sites' requirements which is "closest" to a specified "previous" sites' partition.

The first problem is addressed in Section 2. The solution of the second problem is described in Section 3. Section 4 analyze the simulation results for the proposed methods.

2 Working Set and Access Rate Evaluation Methods

All the methods described in this Section are entirely based 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:

- $A(s)$ - 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");
- $FR(s)$ - 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).

The access rate $A(s)$ is the principal factor that characterizes the load on a system due to the traffic to site s. The working set $W(s)$ characterizes the memory requirements of the site s. These parameters provide a high level characterization of the hosted web sites and their system resource requirements.

Let N be the number of nodes in the web server cluster. 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 each of these N groups is approximately the same.

An important issue that needs to be addressed is dealing with large sites. These are sites with large working set and access rate requirements. 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.

In what follows, we assume that all the machines are identical. However, our approach can be extended to the case where machines have different capacities.

2.1 Basic Definitions and Denotations

Let the working set and access rate requirements of a site s be $W(s)$ and $A(s)$. When the 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, we expect the access rate to this site on each of the k servers to be k times smaller than the original access rate, $A(s)$. We denote the working set on each of the k servers of a site s replicated on these k servers by $W(s\#k)$. Similarly, we denote the access rate of each of these k logical sites by $A(s\#k)$. 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 $k * A(s\#k)$.

Note that all the expressions involving $s\#k$ that appear in this and later Sections are consistent for the case $k = 1$. However, in most expressions, at the expense of some redundancy, we mention the case when the site is put on a single server separately, for the sake of clarity. This also helps the reader take better notice of the consequences of replication on more than one

server.

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:

$$TotalW = \sum_{\text{all sites } s} NewW(s) \qquad \text{and} \qquad TotalA = \sum_{\text{all sites } s} NewA(s)$$

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

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

where N is the number of servers in the cluster.

Next, we describe when and on how many servers a site is replicated. We replicate a site s when its working set $W(s)$ or access rate $A(s)$ exceeds a certain limit. Specifically, we replicate a site s when either

$$W(s) > alpha * MeanW \qquad \text{or} \qquad A(s) > beta * MeanA,$$

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.

Let $Copies(s)$ denote the number of times a site is replicated. Further, recall that $s\#1$ is the same as s. Initially, we have $Copies(s) = 1$ for all the sites s.

Our algorithm for deciding the amount of replication for each site s is as follows:

```
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), MeanW and 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

$$W(s\#Copies(s)) < alpha * MeanW \qquad \text{and} \qquad NewA(s\#Copies(s)) < beta * MeanA.$$

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 put on a single server, we set

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

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

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

Similarly,

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

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 Four Different Methods to Compute Working Set and Access Rate Requirements for Replicated Sites

2.2.1 Simple

The simplest but least accurate method is to simply set:

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

where the k is the number of servers the site s is replicated across.

Intuitively, it means that each of k servers experiences $\frac{1}{k}$-th of the total load (traffic) to the site s. However, the working set (memory requirements) of the site s for each of k servers is the same (i.e., the original working set with no reduction).

2.2.2 Simple+

This method is similar to the *Simple* method. However, unlike in *Simple* method, we estimate the possible reduction of the working sets 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 was replicated across diminishes as k increases. This leads to a smaller working set on each of the k servers.

In this method, for each of the replicated sites s, we use additional information - the frequency for all accessed files. That is, for each file f of site s, we know the access frequency fr_f (the number of times this file was accessed during the observed period P) and the file size $size_f$ in bytes, as described at the beginning of Section 2.

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. (Intuitively, if the file f is accessed at least once on a given server, this file "contributes" its size to the site working set on this server).

Assuming independence of accesses and that all accesses are equally likely to go to any server, 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 have

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

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 accuracy and precision of the site allocation method as well.

2.2.3 Advanced

Methods *Simple* and *Simple+* can be further improved, if the memory (RAM) size of each node in the cluster is available.

Let ram be the size of the memory (RAM) in each node. Recall that there are N nodes in a cluster. 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 then make the following assumptions:

- Best performance is achieved when the "most popular 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.

REMARK: In principle, these assumptions are not true. Replacement policies, such as LRU, etc., do not ensure the second assumption. Moreover, the best performance can be achieved from the (ideal, unpractical) policy where a file that is accessed furthest in future is the one that is evicted, regardless of the popularity of the file. However, we believe that these assumptions are good approximations to the best practical policies.

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, we evaluate the working set requirement of a site s as $W(s) = C(s, fr^{opt})$.

Let $Y(s)$ be the total number of bytes of site s transferred during period P. Recall that we set access rate $A(S)$ to this value in Methods *Simple* and *Simple+*. Note that $Y(s) = \sum_{fr} fr * B(s, fr)$

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 $A(s) = Y(s) + D(s, fr^{opt}) * (DiskWeight - 1)$. Note that $A(s) = Y(s)$ when $DiskWeight = 1$.

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

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

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 in cases where one or more sites are replicated.

2.2.4 Advanced+

This method is essentially identical to Method 2, except for the details of modeling.

Here again, we compute $B(s, fr)$ for each site s and all frequencies fr.

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. 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.

The probability that a byte, that is accessed fr times (expected) in period P, is accessed at least once in period T is given by

$$1 - e^{-\frac{fr*T}{P}}$$

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.

Note that a byte of site s is expected to be in memory if it is accessed at least once in the period T. So, we compute the working set of site s as

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

Again, let $Y(s)$ be a total number of bytes of site s transferred during period P. Recall that $Y(s) = \sum_{fr} fr * B(s,fr)$.

All the bytes of site s that are not in $W(s)$ come from the disk. The number of bytes of the site s transferred from disk is given as:

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

Another way of looking at this is the following: consider a reference to a file f at time t. File f is in memory if it was at least accessed once in the time interval $(t-T,t)$ and has to be downloaded from the disk otherwise. Thus, we assume that the probability that a file is accessed from the disk is equal to the probability that it is not accessed in time T.

We define the site s access rate as: $A(s) = Y(s) + (DiskWeight - 1) * D(s)$.

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

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

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

$$Y(s\#k) = \frac{Y(s)}{k}$$

$$A(s\#k) = Y(s\#k) + (DiskWeight - 1) * D(s\#k)$$

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 in time T, the same byte of each replica site (s#k) is accessed an expected $\frac{fr*T}{k*P}$ times in time T.

3 Algorithm for Generating the "Closest" Balanced Partition

As stated earlier, FLEX monitors the sites requirements periodically (at some prescribed time interval: daily, weekly, or monthly). Because of changed site requirements, the old allocation (partition) may not be good anymore and a new allocation (partition) of sites to machines might be needed. Further, as seen earlier, the new partition must move a minimal number sites to new servers to avoid temporary performance degradation during the reassignment of sites to servers. In this Section, we propose 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.

3.1 Algorithm Inputs

The algorithm *closest* generates a new balanced partition R for the given sites' requirements $Reqs$ such that R is the "closest" to a specified"previous" sites' partition R_{old}.

ALGORITHM *closest*(Partition R_{old}, Requirements $Reqs$)
where

1. Partition R_{old} is specified as follows:
   - For each server S, it contains $sites(S)$- the list of sites assigned to this server;
   - Similarly, for each site s, it contains
     - $servers(s)$ - the list of servers on which this site s is replicated, and
     - $count(s)$ - the number of servers on which site s is replicated, i.e. $count(s) = Copies(s)$.
2. Sites' requirements $Reqs$ contain the following information for each site s:
   - $count(s)$ - the number of servers on which site s is replicated;
   - $W(s\#k)$, where $k = count(s)$;
   - $A(s\#k)$, where $k = count(s)$.

3.2 Basic Definitions and Notation Used in the Algorithm

Let R be a partition, and $Reqs$ be the given sites' requirements.

We use capital letters X, Y, etc, to refer to servers and small letters p, q, r, s etc to refer to sites. Note that "site s" is used here for either s or $s\#k$.

1. For a partition R, site p and server X,

$$R.contains(X, p) = true$$

   if and only if server X is one of the servers allocated to the site p.

2. Let $R.W(X)$ denote the total memory needed for all the sites assigned to server X, and $R.A(X)$ denote the sum of access rates of all the sites assigned to server X.

$$R.W(X) = \sum_{\text{all } s \text{ such that } R.contains(X,s)} Reqs.W(s\#count(s)).$$

$$R.A(X) = \sum_{\text{all } s \text{ such that } R.contains(X,s)} Reqs.A(s\#count(s)).$$

3. For servers X and Y, the $R.Deviation(X, Y)$ is defined as follows

$$|R.W(X) - R.W(Y)| + |R.A(X) - R.A(Y)|.$$

4. For servers X, Y and sites p, q, $R.swap(p, q, X, Y)$ is the new partition obtained by swapping the sites p and q between servers X and Y.
   This operation is valid only if

$$R.contains(X, p) = true \quad \text{and} \quad R.contains(X, q) = false$$

   and

$$R.contains(Y, q) = true \quad \text{and} \quad R.contains(Y, p) = false$$

   (see case 4 below for an exception).
   To understand the conditions

$$R.contains(X, q) = false \qquad \text{and} \qquad R.contains(Y, p) = false,$$

   note that a site s could be replicated on $k > 1$ servers, and each of the k logical "subsite" $s\#k$ has to be kept on a different server.

5. Additionally, $R.swap(p, q, X, Y)$ is also a valid operation if:

   - $p = 0$ and site q is on server Y, in this case, $R.swap(p, q, X, Y)$ refers to the partition obtained by moving site q from server Y to server X.

- $q = 0$ and site p is on server X, in this case, $R.swap(p, q, X, Y)$ refers to the partition obtained by moving site p from server X to server Y.

6. For servers X, Y and sites p, q on servers X and Y respectively, the function *Benefit* is defined as follows:

$$R.Benefit(p, q, X, Y) = R.swap(p, q, X, Y).Deviation(X, Y) - R.Deviation(X, Y)$$

   This definition also holds if one of p or q is 0 (refer to case 4 above).

7. For partitions R and Q

$$distance(R, Q) = cardinality(M) \quad \text{where}$$

$$M = \{(p, X) | \text{ such that } Q.contains(X, p) = true \text{ and } R.contains(X, p) = false\}.$$

   In words, $distance(R, Q)$ measures the number of times a site is present on a server in partition R such that the same site is not present on the same server in partition Q.

8. For a partition R and a real number dev, we define a relation *satisfies* as follows:

$$R\ satisfies\ dev = true \quad \text{if} \quad 100 - dev \leq R.W(X) \leq 100 + dev \quad \text{and}$$

$$100 - dev \leq R.A(X) \leq 100 + dev \quad \text{for all servers } X \text{ in a cluster.}$$

   This means that allocation to all servers is within $dev\%$ of their quotas. Recall that the working sets and access rates were normalized such that each server has a quota of 100 for both access rate and working set.

9. *SET* contains the set of all "good" partitions found so far. Its meaning will be clear from the algorithm pseudocode given below.

3.3 Algorithm *closest*

The informal, short description of the algorithm is as follows.

The algorithm makes *NumSolutions* iterations to find partitions satisfying deviation *dev* and then picks that partition among the found ones that has the least distance from the existing partition.

In each iteration to find a partition satisfying *dev*, the algorithm starts with the existing partition and swaps sites (at most *NumAttempts* times) across servers trying to obtain a partition that satisfies *dev*. If, in less than or equal to *NumAttempts* swaps, a partition is obtained that satisfies *dev*, then this iteration is successful. Otherwise, this iteration is a failure.

After *NumSolutions* iterations, say K partitions satisfying *dev* are found. Note that $K \leq$ *NumSolutions*. If $K = 0$ (that is, no partition satisfying *dev* was found), then *dev* is increased and the whole process is repeated.

The pseudocode for the algorithm *closest* is given below:

```
begin Algorithm
dev = 0
SET = NULL
while (SET == NULL)
    dev += 5;
    for i=1 to NumSolutions do
        R = P; /* make a copy of P */

        if (Reqs.count(p) != R.count(p) for some site p)
           randomly add/drop servers from R.servers(p) to make
                   R.count(p) = Reqs.count(p);
        for j=1 to NumAttempts do

            if i <= NumSolutions/10
                pick X and Y such that R.Deviation(X,Y) is maximum;
                pick two sites p and q (one of them could be 0) such that
                     R.benefit(p,q,X,Y) is maximum;
                R = R.swap(p,q,X,Y);

            else if (i <= 4 * NumSolutions)/10)
                pick X and Y such that R.Deviation(X,Y) is maximum;
                pick two sites  (one of them could be 0) such that
                     probability of picking p and q is proportional
                     to R.benefit(p,q,X,Y) ;
                R = R.swap(p,q,X,Y);

            else if (i <= 7 * NumSolutions/10)
                pick two servers such that the probability of picking X and Y
                     is proportional to R.Deviation(X,Y);
                pick two sites p and q (one of them could be 0) such that
                     R.benefit(p,q,X,Y) is maximum;
                R = R.swap(p,q,X,Y);

            else
                rand = random number in 0..1;
                if (rand > 0.9)
                   pick X and Y such that R.Deviation(X,Y) is maximum

                else
                   pick two servers such that the probability of picking
                             X and Y is proportional to R.Deviation(X,Y);
```

```
                if (rand > 0.9)
                    pick two sites p and q (one of them could be 0)
                             such that R.benefit(p,q,X,Y) is maximum;
                    R = R.swap(p,q,X,Y);

                else
                    pick two sites  (one of them could be 0) such that
                         probability of picking p and q is proportional to
                         R.benefit(p,q,X,Y)
                    R = R.swap(p,q,X,Y)

        endfor

        if (R satisfies dev)
           SET = SET union {R}

    endfor

endwhile
R' = {R | R belongs to SET and distance(R_old,R) is minimum}
end Algorithm
```

There are few places in the pseudocode above which need additional explanation:

- The pseudocode has some "magic" numbers. For example "$NumSolutions/10$", "$4 * NumSolutions/10$" etc. These magic numbers are not very important. We have simply described the exact algorithm we used.
- In principle, we could have used any optimization algorithm, such as a genetic algorithm or hill climbing etc. However, we found that the above algorithm was good enough for our purposes.
- When the pseudocode says:
  "probability of picking sites p and q is proportional to $R.Benefit(p, q, X, Y)$",
  it means that the probability of picking sites p and q is given by

$$P(p, q) = \frac{R.Benefit(p, q, X, Y)}{R.TotalBenefit(X, Y)}.$$

  Here

$$R.TotalBenefit(X, Y) = \sum_{\text{all site pairs } r,s} R.Benefit(r, s, X, Y)$$

  where sites r and s are such that $R.swap(r, s, X, Y)$ is valid.

- When the pseudocode says :
  "probability of picking X and Y is proportional to $R.Deviation(X,Y)$",
  it means that the probability of picking servers X and Y is given by

$$P(X,Y) = \frac{R.Deviation(X,Y)}{R.TotalDeviation} \text{where}$$

$$R.TotalDeviation = \sum_{\text{all server pairs } X,Y} R.Deviation(X,Y).$$

Note that X and Y cannot be the same server.

4 Simulation Results

For our experiments, 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 created 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. As stated earlier, 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 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 "leaders"). 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 14.5K from the RAM (14.5k 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.[1]
- 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

[1] 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).

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.

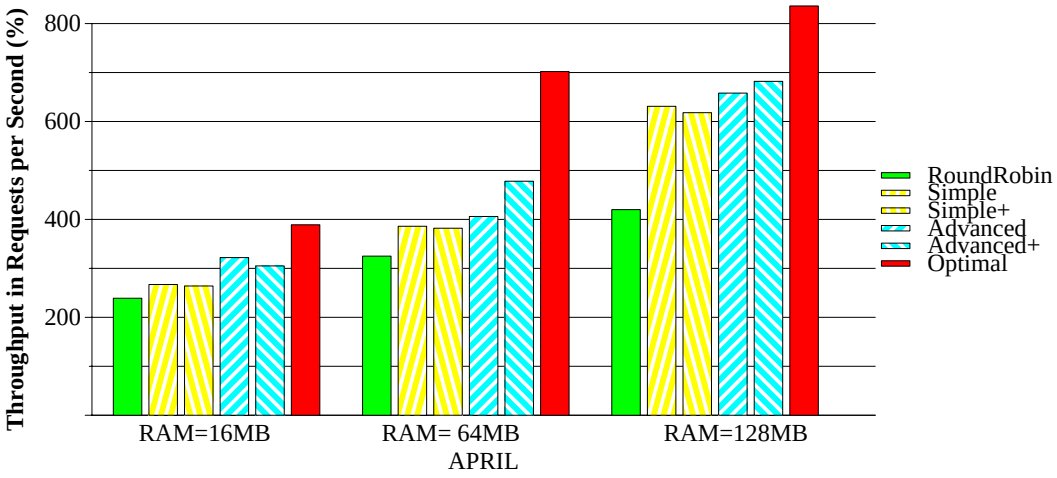

Figure 1: April.

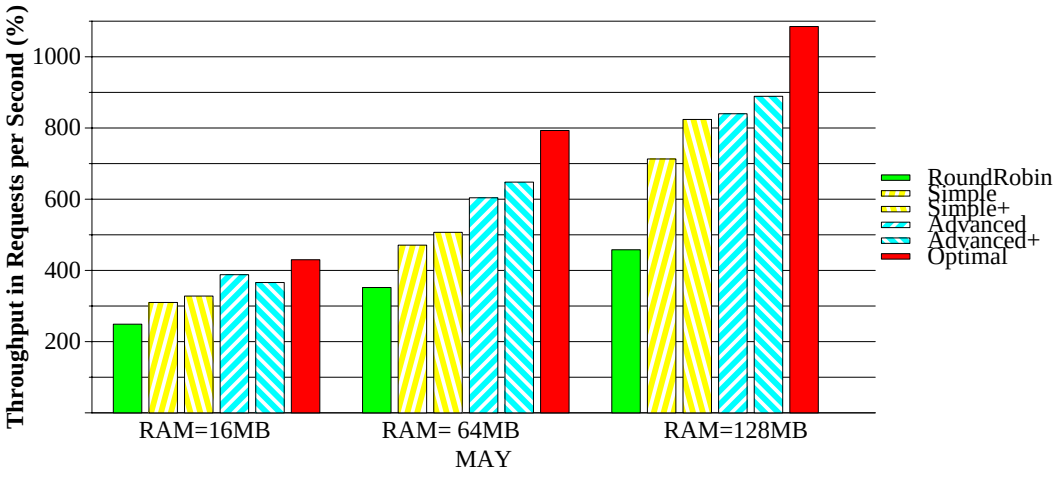

Figure 2: May.

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).[2] Figures 1, 2, 3, 4 show the maximum achievable throughput for Round-Robin, different FLEX strategies and *Opt* for different RAM sizes (the RAM sizes shown are per server).

[2]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 memory.

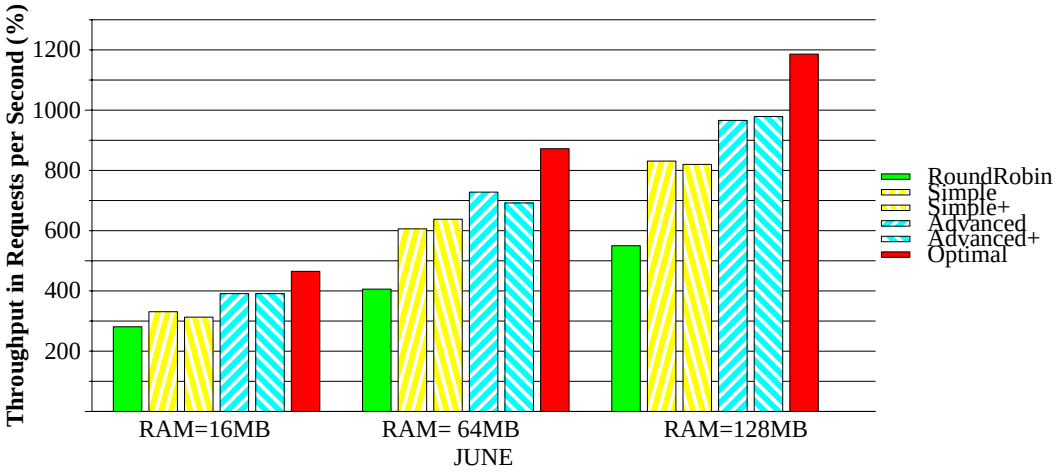

Figure 3: June.

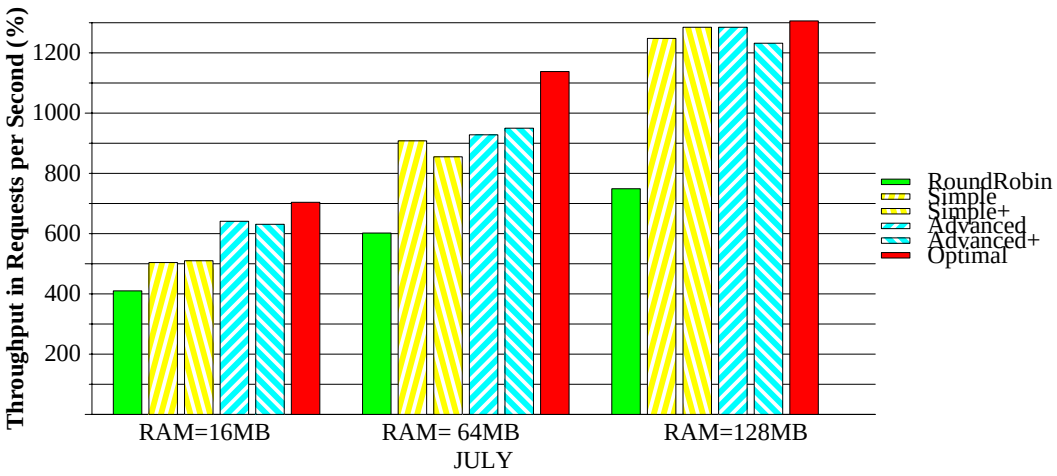

Figure 4: July.

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 *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.

For the traces of April and RAM sizes of 16 MB and 64 MB, FLEX's performance benefits are modest. One of the reason for this is a number of accesses to very large files (10MB to 15MB size) 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.

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).

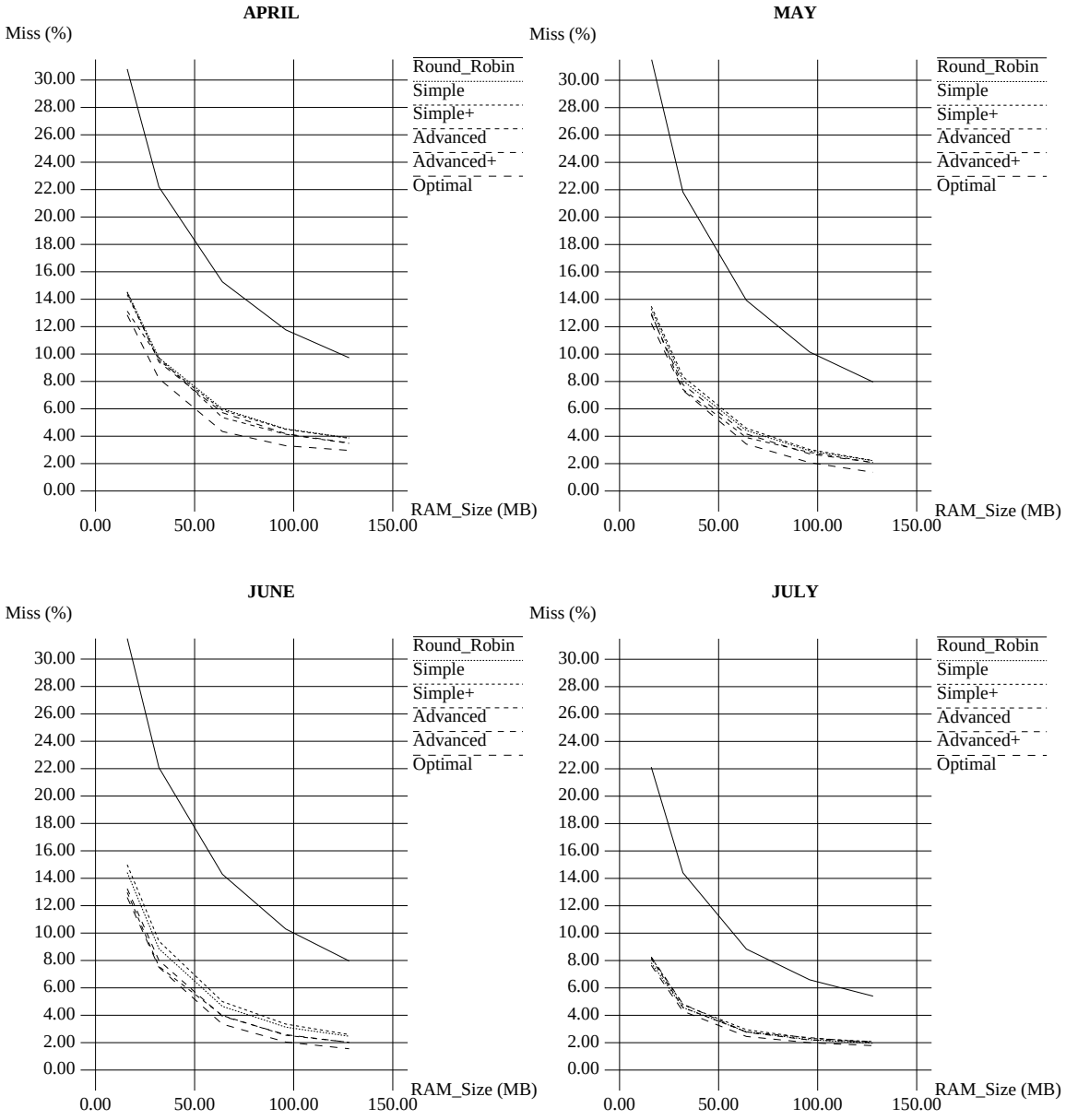

Figure 5: April, May, June, July.

Figure 5 shows the miss ratio of Round-Robin, different FLEX strategies and *Opt* for different RAM sizes. As usual, the RAM sizes shown are per server.

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.

5 Conclusions and Future Work

Our simulation results confirm that FLEX achieves its twin goals of efficient memory usage and load balancing. Further, FLEX has the following advantages compared to other *locality aware* strategies:

- Ease of deployment and low cost.
- 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.

Future work will investigate above mentioned ideas and also extending the solution and the algorithm to work with heterogeneous nodes in a cluster, SLA (Service Level Agreement), and some additional QoS requirements.

6 References

[B97] 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.

[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.

[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.

[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

[R98] 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.